A study of the semi-dynamic traffic assignment using the sensitivity analysis

Dissertation

A study of the semi-dynamic traffic assignment using the sensitivity analysis

Graduate School of
Natural Science & Technology
Kanazawa University

Division of Environmental Design

Student ID No. 1724052013
Name: Bui Tien Thiem
Chief advisor: Professor Shoichiro Nakayama
Date of Submission June 25th, 2020

Contents

Chapter 1

Introduction

1.1 Background

For the purpose of understanding and developing an optimal transport network with the conductive movement of traffic and minimum traffic congestion problems, traffic flow researchers have studied interactions between users and infrastructure. In other words, the traffic flow study's purpose is to create and apply an optimal model that would enable vehicles to reach their destination in the shortest possible time using the maximum roadway capacity. Traffic flow has traditionally followed the sequential four-step model or urban transportation planning procedure, including trip generation, trip distribution, mode choice, and traffic assignment, first executed at the Detroit Metropolitan Area Traffic Study and Chicago Area Transportation Study in the 1950s. In the urban transportation planning procedure, traffic assignment has concerned the selection of routes between origins and destinations in the transport network and the estimation of network performance. These last step results are long recognized as a key component for future planning scenarios such as junctions design, proposing transportation policies to minimize congestion, predicting the change of travel demand and travel time caused by road maintenance projects, and so on.

However, the term traffic assignment has caused confusion for practitioners in recent years because of various distinctly different model concepts and implementations. There are many types of traffic assignment models, but they can be divided into static and dynamic traffic assignment models. In a static traffic assignment model (STA) defined on a relatively long time-of-day period, such as the peak period, the traffic condition is described by the average or steady-state travel time on a link as a function of the volume of traffic on the link (volume-delay function or link time-performance function). However, because traffic conditions of most cities change significantly with time of day, STA could not adequately represent traffic flow, especially time-varying congestion phenomena. Dynamic traffic assignment model (DTA) models that aim to describe such time-varying network and demand interaction using a behaviorally sound approach

seem more reasonable. DTA can be divided into simulation models and mathematical theory models, depending on the approach. In the case of dynamically analyzing traffic flow on a network, many microsimulation models with a large number of parameters and assumptions have been proposed to fit the model to reality. However, it is easy to fall into the over-fitting state and the stability of the model is not guaranteed when changing the input parameters. On the other hand, the mathematical theory model should play a benchmark role in solving problems that can occur when using the simulation model described above. There are many studies on the mathematical theory of DTA in recent years, but it is hard to say that a general solution has been established, and it is still developing. Besides, when trying to apply to the whole metropolitan area, we face problems of insufficient data and computational burden. Detailed origin-destination (OD) matrix data and large calculation capacity are the requirements of DTA models. Available traffic OD data which is obtained by road traffic census and person trip survey will be relatively coarse, finely divided at 60 minutes.

As a combination of STA and DTA models, a semi-DTA model is one of the efficient alternatives for describing within-day traffic dynamics for large-scale traffic networks. In this approach, with a day is split into several periods, static network equilibrium is reached in each period, but the flow propagation between periods is added. While the ordinary STA technique and algorithm can be applied to the semi-dynamic model, it is also an alternative for describing daily traffic dynamics of large-scale networks, simple and easy to understand concepts, and is with small computation load. This approach assumed that all vehicles departing from their origin do not reach their destinations. The flow on a link that cannot exit the link (called “residual flow”) in a given period is propagated to the next period as travel demand between the end node of that link and the original destination; this effect is known as flow propagation. In the present period, this residual flow should be removed on the subsequent links. The magnitude of residual flow on a link that depends on the inflow and the link travel time in this study. In the techniques and algorithms of the STA models, the stochastic user equilibrium (SUE) traffic assignment model with a typical logit-based model has been commonly used because of its practicality and applicability. If the logit-based route choice is assumed in the semi-DTA model and the link travel time is a function of its inflow, the double-looped fixed-point problem needs to be solved for the

existence and uniqueness of semi-dynamic SUE flow.

Nevertheless, the huge cost of computing for solving the double-looped fixed-point problem is still a big problem when applying the semi-DTA model to a large traffic network. In our study, the above problem will be solved by the approximation method using sensitivity analysis and the accuracy of the approximate results as well as the calculation efficiency will be shown.

Besides, if the residual flow is eliminated and propagated to the next period, the number of OD pairs will escalate dramatically and therefore the number of routes will increase greatly and the computer may be overloaded. Thus, we will also propose a link-based and node-based variable approach for saving storage.

Also, the multinomial logit-based (MNL) traffic assignment has also been criticized because of independent distribution and identical variance assumptions. Various models have been proposed to address those two weaknesses, of which the cross-nested logit (CNL) model and the *q*-generalized logit model are essential proposals with many advantages. Thus, this semi-DTA model needs to be extended for applying to the various extensions of the logit model, such as the CNL model and the *q*-generalized logit model.

1.2 Purpose

With the desire to propose a traffic assignment model with high applicability and efficient calculation method, contributing to the science of traffic assignment models, this study includes the following purposes:

- The sensitivity analysis method is considered as an effective tool and a key calculation method for this study. Therefore the ambitious goal is to propose an efficient sensitivity analysis method with link-based and node-based variables to achieve an approximate equilibrium solution that ensures accuracy and reduces computation time in the static logit- based SUE traffic assignment models.

- Proposing the semi-dynamic SUE traffic assignment model that takes into account the spatial-temporal movement of congestion to a real-scale network. To reduce computational cost and increase the applicability of the semi-DTA model, the sensitivity analysis method will be used. Besides, the target of applying the algorithm with only link-based and node-based approaches to the research model to increase the calculation efficiency will be taken into account.

- To generalize the model and calculation method, this study also proposes the sensitivity analysis method and the semi-DTA model with sensitivity analysis for these models including the CNL model and the *q*-generalized logit model. This is a corollary to the fact that these expansion logit-based models could fix the shortcomings of the traditional MNL model. These expansion models also need to reach the aim of reducing calculational time and computer capacity.

The effectiveness of the model and calculation method will be illustrated by applications to simple examples and Kanazawa City road networks.

1.3 Structure of the dissertation

This dissertation consists of 6 chapters including an introduction. The remainder includes the following chapters:

Chapter 2 will summarize the historical concept and related researches. After organizing these past studies, the position of this research will be located.

The sensitivity analysis method is often of interest when looking for an approximate traffic assignment equilibrium solution that guarantees accuracy and reduces computation time. Chapter 3 will propose the sensitivity analysis method for the static logit-based SUE traffic assignment model. The efficient calculation and validity of the proposed method will also be depicted through the applications to a small network and Kanazawa city road network.

The concept and formulation of a semi-DTA will be described in Chapter 4. Besides, the sensitivity analysis method for solving the SUE of this model will be proposed with the link-based algorithm. Small virtual and Kanazawa city road networks will also be considered as case studies to illustrate the proposed models.

The sensitivity analysis method and a semi-DTA model with sensitivity analysis for the CNL model and the *q*-generalized logit model would be conducted in Chapter 5. The comparisons and the best solution for representing the traffic situation of the Kanazawa city road network will be also shown in this chapter.

Chapter 6 will summarize the research and consider future works.

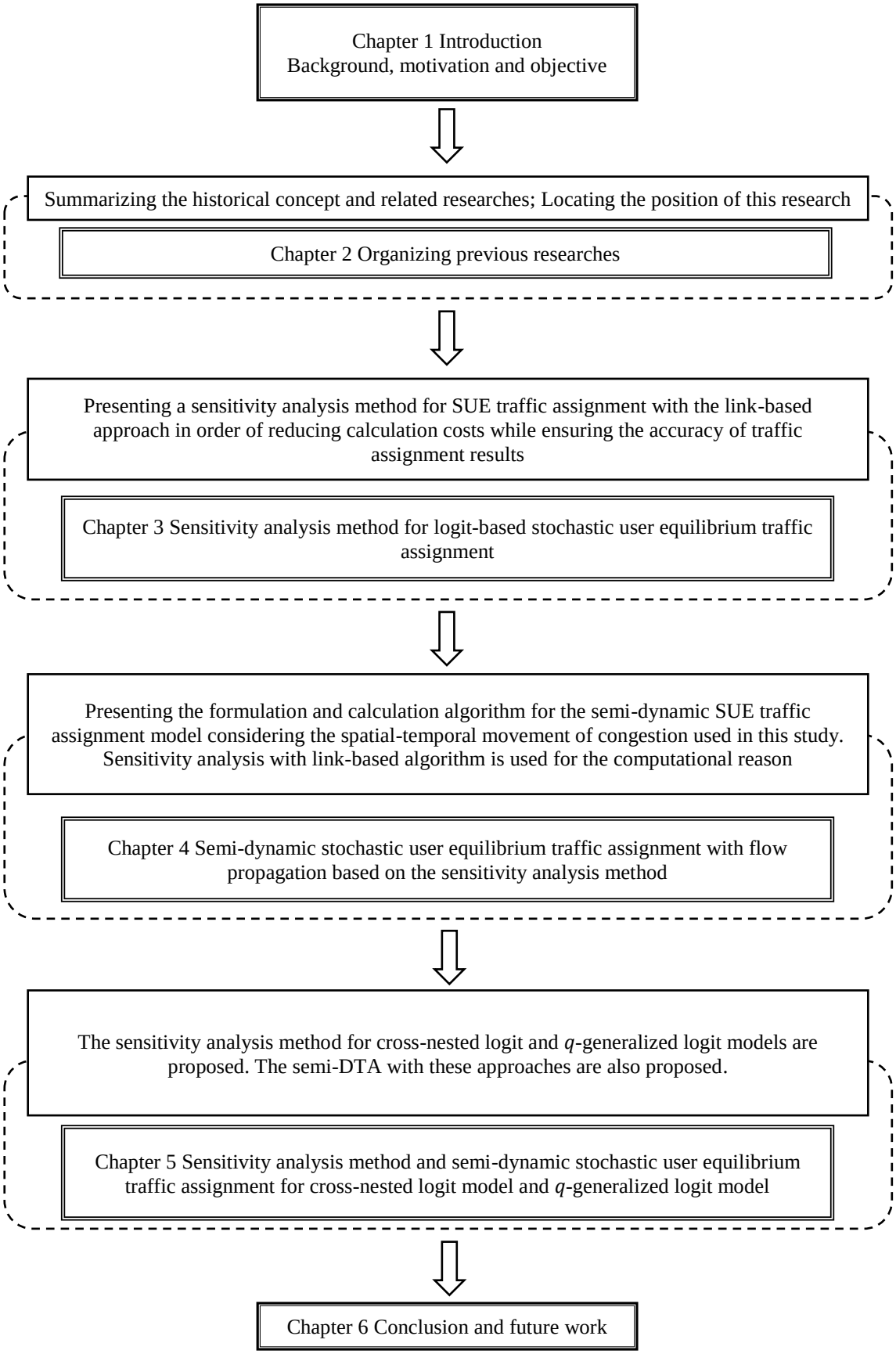


Figure 1.1 Structure of this dissertation

Chapter 2

Organizing previous researches

2.1 Overview

This chapter will summarize the basic concept of traffic assignment models and organize existing studies related to this research. Then, the position of this study will be presented.

2.2. History of the development of traffic assignment model

An actual traffic network is abstractly represented by nodes and links connecting nodes. The pattern of observed user's movements represented by the trip matrix (OD traffic volume) will be reproduced by a traffic assignment model. In other words, the traffic assignment model aims to predict the volume of traffic that flows to each link on the network given level of travel demand. The results of traffic assignment depict reasonable link flows and heavily congested links. To make it more user-friendly, it has been collaborated with the behavior model to consider the route selection behavior between each OD pair. It is difficult to build a universal model because users always select routes by subjective judgment. Therefore, the principle of allocation has been introduced, and models have built based on it. Traffic assignment models can be broadly divided into STA and DTA depending on how traffic is handled. The concepts and basic theories of typical traffic assignment model will be summarized as follows:

2.2.1 User equilibrium model

STA consists of two major directions; the deterministic user equilibrium (UE) and SUE. The UE model is the most basic model among traffic assignment models. Based on the existence of a balance between supply and demand developed in the field of neoclassical economics, Wardrop[1)] attempted to clarify the equilibrium concept of supply and demand of traffic over the network in 1952. The proposed Wardrop's principles including two principles.

Wardrop's first principle

Wardrop[1)] proposed an equilibrium principle in traffic network as follows: The travel times on all routes used are equal, and less than those which would be experienced by a single vehicle on any unused route. In other words, under equilibrium conditions, no user can reduce his/her travel time by changing the route. This is referred to as Wardrop's first principle[2)]. This traffic equilibrium state is known to be equivalent to the Nash equilibrium of the N-person game from the viewpoint of game theory. The first principle is also called the UE principle because its contents mean the equilibrium state reached by road users as a result of optimizing their route selection behavior. For this equilibrium principle to hold, the following two assumptions must be satisfied. One assumes that all users always

act to minimize travel time. Another assumption is users always have perfect information about available routes (no stochastic effects).

Wardrop's second principle

An alternative principle of assigning traffic onto a network was also proposed by Wardrop. His second principle aims to optimize the entire system and is called a system optimization assignment. Under social equilibrium conditions, the total travel time on the road networks is minimized. This implies that each user cooperatively active in choosing his/her route to achieve the most efficient use of the whole. The traffic equilibrium state established here is a state in which not all users pass through their shortest route. Therefore, the equilibrium is different from the first principle and the flows resulting from the two principles are also different.

Consequently, mathematical expressions are formulated based on Wardrop's UE. The first principle stipulates that users of the traffic network will choose the minimum cost route between each OD pair, and through this process, the routes that are used (i.e., have positive flows) will have equal costs; moreover, routes with costs higher than the minimum will have no flow. In this study, we consider the traffic network with N nodes, A links, and W OD pairs, where $N = \{1,2,\dots,i,j,\dots m,n\}$ is the set of nodes, $A = \{12,\dots,ij,\dots,gh,\dots,mn\}$ is the set of links and $W = \{12,\dots,rs,\dots,uv\}$ is the set of OD pairs. Here we consider the case where the travel cost function on each link is monotone and the travel demand for each OD pair is fixed. The fixed travel demand of each OD pair rs is denoted by Q_0^{rs} and the set of routes connecting each OD rs is denoted by K^{rs}. The traffic volume and the required travel time on the route k between a certain OD pair rs is denoted by f_k^{rs}, and c_k^{rs}. The set of route-flows, Ω^{rs}, must satisfy (flow conservation condition)

$$\sum_{k \in K^{rs}} f_k^{rs} = Q_0^{rs}, \quad f_k^{rs} \geq 0 \ \forall\, k \in K^{rs}, rs \in W. \tag{2.1}$$

If the minimum travel time between OD pair rs in is λ^{rs}, the first principle is expressed as follows:

$$\begin{aligned} &c_k^{rs} - \lambda^{rs} = 0 \; if \; f_k^{rs} \geq 0, \ \forall\, k \in K^{rs}, rs \in W, \\ &c_k^{rs} - \lambda^{rs} \geq 0 \; if \; f_k^{rs} = 0, \ \forall\, k \in K^{rs}, rs \in W. \end{aligned} \tag{2.2}$$

Above expression is written in a complementary form as follows:

$$\begin{cases} 0 \le c_k^{rs} - \lambda^{rs} \perp f_k^{rs} \ge 0, \quad \forall k \in K^{rs}, rs \in W, \\ f_k^{rs} \in \Omega^{rs}. \end{cases} \tag{2.3}$$

When these are applied to an actual network, it is more convenient to calculate traffic flow on links. The number of drivers using link ij can be expressed as the sum of the total route traffic of all OD pairs that include link ij in that route. This is formulated as the following equation and vector-valued function:

$$x_{ij} = \sum_{rs \in W} \sum_{k \in K^{rs}} f_k^{rs} \delta_{ij,k}^{rs}, \forall ij \in A, k \in K^{rs}, rs \in W, \tag{2.4}$$

$$\mathbf{x} = \Delta \mathbf{f}, \tag{2.5}$$

here, x_{ij} is the traffic flow on link ij; $\delta_{ij,k}^{rs}$ is the link-route connection variable, which is 1 if the link ij exists on the route k between the OD pair rs, and 0 if it does not exist. $\mathbf{x} = (x_{12}, \dots, x_{ij}, \dots, x_{mn})^T$ and $\mathbf{f} = (f_1^{12}, \dots, f_{|K^{rs}|}^{rs}, \dots, f_{|K^W|}^{uv})^T$ are the vector forms of link flow and route flow, respectively. $\Delta = \{\delta_{ij,k}^{rs}\}$ is the link-route incidence matrix and ^{T} is the transpose. The route traffic f_k^{rs} and the link traffic x_{ij} are non-negative. Next, the travel time will be considered. Since the route travel time is expressed as the sum of the link travel times of the links existing on the route, it is as follows:

$$c_k^{rs} = \sum_{ij \in A} t_{ij} \delta_{ij,k}^{rs}, \forall ij \in A, k \in K^{rs}, rs \in W, \tag{2.6}$$

here, t_{ij} is the travel time of link ij, but since it changes depending on the traffic passing through the link, it can be regarded as a function of link traffic x_{ij}, so it is expressed as follows.

$$t_{ij} = t_{ij}(x_{ij}), \forall ij \in A. \tag{2.7}$$

Equations (2.1) to (2.7) could be rewritten into an equivalent mathematical optimization problem as follows:

$$\min. Z(\mathbf{x}(\mathbf{f})) = \sum_{ij \in A} \int_0^{x_{ij}} t_{ij}(z)\, dz$$

$$s.t. \begin{cases} x_{ij} = \sum_{rs \in W} \sum_{k \in K^{rs}} f_k^{rs} \delta_{ij,k}^{rs} \quad , x_{ij} \geq 0 \; \forall \; ij \in A, \\ \sum_{k \in K^{rs}} f_k^{rs} = Q_0^{rs}, \quad f_k^{rs} \geq 0 \; \forall \; k \in K^{rs}, rs \in W. \end{cases} \tag{2.8}$$

The equivalence of the UE and the problem (2.8) was demonstrated by Sheffi[3]. And Sheffi[3] also presented the convexity and uniqueness solution of the problem (2.8). The convex combinations method (also known as Frank-Wolfe (FW) method) was demonstrated for solving the UE model[3]. The characteristic of the UE assignment is that the link traffic volume is uniquely determined, but the route traffic volume is not uniquely determined. For solving the convex optimization problem (2.8), given a current feasible link flows solution, $\mathbf{x}^{(l)}$, the lth iteration of the FW will find the descent direction $\mathbf{d}^{(l)}$and move size in this direction with step size α. $\mathbf{d}^{(l)}$ is the line connecting the current solution, $\mathbf{x}^{(l)}$, with the solution of a linear approximation problem (denoted $\mathbf{yx}^{(l)}$). This approximation is given by:

$$\min Z^{(l)}(\mathbf{yx}) = Z\left(\mathbf{x}^{(l)}\right) + \nabla Z\left(\mathbf{x}^{(l)}\right)(\mathbf{yx} - \mathbf{x}^{(l)})$$

$$s.t. \begin{cases} yx_{ij} = \sum_{rs \in W} \sum_{k \in K^{rs}} f_k^{rs} \delta_{ij,k}^{rs} \quad , yx_{ij} \geq 0 \; \forall \; ij \in A, \\ \sum_{k \in K^{rs}} f_k^{rs} = Q_0^{rs}, \quad f_k^{rs} \geq 0 \; \forall \; k \in K^{rs}, rs \in W, \end{cases} \tag{2.9}$$

where $\mathbf{x}^{(l)}, Z\left(\mathbf{x}^{(l)}\right), \nabla Z\left(\mathbf{x}^{(l)}\right)$ are constant in which $\nabla Z\left(\mathbf{x}^{(l)}\right) = \left(\frac{\partial Z(\mathbf{x}^{(l)})}{\partial x_{12}}, \dots, \frac{\partial Z(\mathbf{x}^{(l)})}{\partial x_{ij}}, \dots, \frac{\partial Z(\mathbf{x}^{(l)})}{\partial x_{mn}}\right) = \left(t_{12}\left(x_{12}{}^{(l)}\right), \dots, t_{ij}\left(x_{ij}{}^{(l)}\right), \dots, t_{mn}\left(x_{mn}{}^{(l)}\right)\right)$.

The above minimization problem becomes:

$$\min Z^{(l)}(\mathbf{yx}) = \sum_{ij \in A} t_{ij}\left(x_{ij}{}^{(l)}\right) yx_{ij}$$

$$s.t. \begin{cases} yx_{ij} = \sum_{rs \in W} \sum_{k \in K^{rs}} f_k^{rs} \delta_{ij,k}^{rs} \quad , yx_{ij} \geq 0 \; \forall \; ij \in A, \\ \sum_{k \in K^{rs}} f_k^{rs} = Q_0^{rs}, \quad f_k^{rs} \geq 0 \; \forall \, k \in K^{rs}, rs \in W. \end{cases} \tag{2.10}$$

It would be tantamount to finding the flow pattern that minimizes the total travel time over the network with the fixed link travel times and OD travel demands. The solution to this problem is that all the flow for a given OD pair rs is assigned all to the shortest route connecting this pair (also known as the "all-or-nothing" assignment). Shortest route search methods such as the well-known Dijkstra method[4] play an important role in this step.

Because $\mathbf{yx}^{(l)}$ are a solution of a linear program and the convexity of Z, the new solution, $\mathbf{x}^{(l+1)}$, must lie between $\mathbf{x}^{(l)}$ and $\mathbf{yx}^{(l)}$. Equivalently, we must find a step size, λ, by solving the following problem:

$$\min_{0 \leq \lambda \leq 1} Z\left[\mathbf{x}^{(l)} + \lambda\left(\mathbf{yx}^{(l)} - \mathbf{x}^{(l)}\right)\right]. \tag{2.11}$$

One-dimensional search methods such as the golden section method (proposed by Kiefer, 1953) can be used for solving the above problem[3]. The new solution is generated by:

$$\mathbf{x}^{(l+1)} = \mathbf{x}^{(l)} + \lambda\left(\mathbf{yx}^{(l)} - \mathbf{x}^{(l)}\right). \tag{2.12}$$

The FW method can be summarized as follows:

Step 1: Initialization

Set $l := 1$. Choose convergence error σ ($\sigma := 10^{-3}$). Based on the free-flow travel time pattern $\left\{t_{ij}^0\right\}$, perform an all-or-nothing assignment to obtain $\mathbf{x}^{(l)}$.

Step 2: Direction finding

Update $t_{ij}{}^{(l)} = t_{ij}\left(x_{ij}{}^{(l)}\right) \forall ij \in A$. Perform all-or-nothing assignment based on $\left\{t_{ij}{}^{(l)}\right\}$ to obtain $\mathbf{yx}^{(l)}$.

Step 3: Step-size determination

Find λ by solving $\min_{0\le\lambda\le1} Z[\mathbf{x}^{(l)} + \lambda(\mathbf{y}\mathbf{x}^{(l)} - \mathbf{x}^{(l)})]$.

Step 4: Move

Set $\mathbf{x}^{(l+1)} = \mathbf{x}^{(l)} + \lambda(\mathbf{y}\mathbf{x}^{(l)} - \mathbf{x}^{(l)})$

Step 5: Convergence test

If $\|\mathbf{x}^{(l+1)} - \mathbf{x}^{(l)}\| \le \sigma$, stop. Otherwise, let $l := l + 1$ and go to Step 2

2.2.2 Stochastic user equilibrium

In the UE assignment model described above, it is assumed that all drivers have the same travel cost perception and perfect information about the network situation. Thus, they will select only their shortest route. However, assuming an actual situation such as there is a different perceived travel cost between users or the user does not have enough information about the situation of travel, the route considered to be the shortest for each user will be different and the UE assignment model is based on unrealistic assumptions.

It is natural to think that the actual route selection varies depending on the user. Therefore, a stochastic SUE has been developed as a model that can take into account the variation in driver's route selection behavior and can take into account the congestion phenomenon. The equilibrium state established by the SUE assignment model can be expressed as follows[5]: In a SUE, each user chooses the route that seems to have the lowest travel cost for him/her. Travel cost here is often considered travel time. As a result, any user no longer believes that changing his/her route can reduce his/her travel time. Routes with a higher cost than the shortest route will also be utilized because of the difference in travel cost perception among users. In the SUE assignment model, an error term is introduced into the user's perceived travel time (travel time recognized by the user). This is similar to the discrete choice model based on the random utility theory.

To formulate the SUE model, let $\mathbf{U} = (U_1, \dots, U_K)$ be the vector of random utilities associated with a given set of alternatives $\{1, \dots, K\}$. The utility of each alternative to a specific decision-maker is a function of the observed attributes of the alternatives and the observed characteristics of the decision-maker. These characteristics and attributes variables are denoted by vector **a.** Thus, $U_k = U_k(\mathbf{a})$.

For simplify of presentation, this dependence is omitted from the notation. To consider the effects of unobserved characteristics and attributes, U_k is expressed as a random variable including a deterministic component, V_k, and an additive random component (the error term), ε_k, that is

$$U_k = V_k + \varepsilon_k, \tag{2.13}$$

where U_k is the overall utility of choice k, V_k is deterministic utility and ε_k is the random utility component (the error term). The error term allows for two cases: two individuals with the same measured attributes and facing the same choice set make different decisions and some of them do not select the best alternative (it showed irrational behavior). Given the distribution of possible utility values, the probability that a specific alternative will be selected by a decision-maker can be calculated. We consider that p_k is the probability of choosing choice k. The choice probability is the probability that U_k is higher than the utility of any other alternative and

$$p_k = \Pr[U_k > U_i, \ \forall i \in \{1, \dots, K\} \ and \ i \neq k \ ] \ \forall k \in \{1, \dots, K\}. \tag{2.14}$$

Once the distribution of the error term is specified, the distribution of the utilities can be showed and the probability of choosing an alternative can be calculated. One of the most popular discrete choice models is the logit model in which the random terms of each utility function are assumed independently and identically distributed Gumbel variates. Besides, by assuming that the random error term of each utility is normally distributed, the probit model has been also proposed. In route choice context, the SUE model could be also divided into two particular interests: the logit model (Dial[6]; Bell[7]) and the probit model with the Monte-Carlo procedure (Burrel[8]; Daganzo and Sheffi[5]). As compared to the probit model, the logit model with closed-form structure is more practical. The assumed logit model will be described. The utility of route k between OD pair rs is assumed as

$$U_k^{rs} = V_k^{rs} + \varepsilon_k^{rs}, \tag{2.15}$$

where $V_k^{rs} = -\theta c_k^{rs}$ in which θ is the positive dispersion parameter. The larger θ is, the smaller the perceived travel cost error is, and the closer to deterministic UE model the SUE model is. Note that $V_k^{rs} = -\theta c_k^{rs}$ does not mean that other

variables such as price cannot be considered, and other variables can be added to the utility V_k^{rs} without losing generality. Consequently, the probability p_k^{rs} that the route k is selected from the route set K^{rs} between the OD pair rs can be expressed as follows

$$p_k^{rs} = \frac{\exp(-\theta c_k^{rs})}{\sum_{k \epsilon K^{rs}} \exp(-\theta c_k^{rs})} \tag{2.16}$$

Using the above equation, the route flow f_k^{rs} is calculated as the following equation:

$$f_k^{rs} = Q_0^{rs} p_k^{rs} = Q_0^{rs} \frac{\exp(-\theta c_k^{rs})}{\sum_{k \epsilon K^{rs}} \exp(-\theta c_k^{rs})}. \tag{2.17}$$

Using the above expression, the flow conservation and the nonnegative of flow conditions are automatically satisfied. Similar to the UE traffic assignment, the relationships between the route flow and link flow, the route travel time, and link travel time are given by **Equations (2.4) to (2.6)**.

For solving the logit-based SUE model, a general formulation as a mathematical program regarding the evaluation of the objective function and the stochastic network loading models with the method of successive averages (MSA) were proposed. Fisk[9)] proposed the model as the unique solution of the convex minimization program and Daganzo[10)] proposed an unconstrained minimization model. However, they suffered the slow convergence and computer capacity problems because the objective function evaluation of these models needed to enumerate all routes. Powell and Sheffi[11)] based on Fisk's objective function and presented a link-based MSA algorithm with a predetermined step size sequence. Furthermore, in the case of the link travel times depended on traffic flows, the equilibrium state of traffic network using MSA with stabilized the definition of efficient routes converged to minimization programs was proved by Powell and Sheffi. Nevertheless, it also had slow convergence because of the appearance of the entropy function defined by route flows in Fisk's objective function. This entropy function was decomposed into a function consisting of only links and

nodes variables by Akamatsu[12]. Among them, the well-known mathematical optimization problem under the constraints proposed by Fisk[9] have been commonly used, that is

$$\min_{\mathbf{x},\mathbf{f}} \sum_{ij\in A} \int_0^{x_{ij}} \mathrm{t}_{ij}(z)dz + \frac{1}{\theta} \sum_{rs\in W} \sum_{k\in K^{rs}} f_k^{rs} \ln \frac{f_k^{rs}}{Q_0^{rs}}$$

$$s.t. \begin{cases} \sum_{k\in K^{rs}} f_k^{rs} = Q_0^{rs}, \quad f_k^{rs} \geq 0 \ \forall\, k \in K^{rs}, rs \in W \\ x_{ij} = \sum_{rs\in W} \sum_{k\in K^{rs}} f_k^{rs} \delta_{ij,k}^{rs} \quad , \forall\, ij \in A. \end{cases} \tag{2.18}$$

Solving these mathematical optimization problems by numerical methods gives solutions[13]. The typical solution is a simplicial decomposition method (Damberg et al. [14]). The detail of this algorithm is omitted here.

In another approach, the solution of the SUE problem is iterative in the sense of a fixed-point problem and the MSA can be used to achieve an equilibrium state. From **Equations (2.1) to (2.7)**, the fixed-point problem with the route-traffic flow is considered as

$$f_k^{rs} = Q_0^{rs} \frac{\exp\left(-\theta c_k^{rs}(\mathbf{f})\right)}{\sum_{k\in K^{rs}} \exp\left(-\theta c_k^{rs}(\mathbf{f})\right)}. \tag{2.19}$$

The logit-based model, however, when applied to large facing difficulties is that the enumeration of possible alternatives routes is implausible. To avoid route enumeration in a logit stochastic loading, the logit-based stochastic network loading algorithms obviating route enumeration (Dial[6]; Bell[7]) have been widely used and the well-known MSA algorithm has been used for achieving SUE solution. Dial's algorithm including STOCH and STOCH2 procedure has been preferred to use (Sheffi[3]). The travel demand has been assigned to *"efficient"* routes connecting each OD pair in this algorithm. The difference between STOCH and STOCH2 procedure lies in the definition of *"efficient"* routes. STOCH algorithm (double-pass algorithm) consider an efficient route between each OD pair is the route consisting of links that take the user further away from the origin and closer to the destination, meanwhile, STOCH2 algorithm (single-pass

algorithm) defines an efficient route is a route including links that have their initial node closer to the origin than their final node. However, travel time may be changed from one iteration to the next because of flow-dependent assumption and these definitions are unstable (Leurent[15]). Based on Dial's algorithm, Leurent proposed the STOCH3 algorithm and the definition of the STOCH3-efficient route. A *"STOCH3-efficient"* (or reasonable, or available) route is the route satisfying three conditions: It does not include the same node more than once, it consists of links have their initial node closer to the origin than their final node, and every link of this route is *"reasonable enough"* compared to a reference shortest route. He also introduced a definition of a maximum *"elongation ratio"* for a link concerning the origin, h_{ij}^r, and confirmed that $h_{ij}^r \in [1.3; 1.5]$ in urban studies (in a result due to Tagliacozzo and Pirzio[16]). If we assume that C_{rn}^0 denotes the reference shortest generalized travel time from origin r to node n, based on the free-flow travel time $\{t_{ij}^0\}$ or travel times pattern at UE solution, the *"reasonable enough"* of route k can be expressed as the following condition:

$$\left(1 + h_{ij}^r\right)\left(C_{rj}^0 - C_{ri}^0\right) \geq t_{ij}^0 \quad \forall ij \in A. \tag{2.20}$$

Using the STOCH3-efficient route, Leurent performed the effectiveness of the STOCH3 logit model with MSA and stated that better behavioral models and computationally useful would be attained with stable route identification. The STOCH3 algorithm with the MSA method for each origin is summarized as follows:

Program variables:

Ω_{ij}^r: Indicator variable ($\Omega_{ij}^r := 1$ if link ij is efficient from origin node r and $\Omega_{ij}^r := 0$ otherwise).

O_i^r: The ith node in the order of increasing access time C_{ri}^0 from origin node r.

a_{ij}: Likelihood of link ij ($a_{ij} = \exp(-\theta t_{ij})$).

wl_{ij}^r: Link weight that presents the importance of link ij in contributing to an efficient route from the origin node r.

wn_n^r: Node weight.

xn_n^r: Flow passing through node n from the current origin node r.

xl_{ij}^r: Flow of link ij from the current origin node r.

The STOCH3 procedure for each origin node with MSA method:

Step 0: Preliminaries

(a) Set iteration counter $l := 1$ and maximum value of the change in link flow from the previous iteration $l-1$ to the current one l ($\sigma := 10^{-3}$)

(b) Based on the free-flow travel time $\{t_{ij}^0\}$, for each origin node r, calculate the minimum travel time to all other nodes. Determine C_{rn}^0 for each node n and the order of increasing access time from origin node r.

(c) For each link ij, set $\Omega_{ij}^r := 1$ if $(1 + h_{ij}^r)(C_{rj}^0 - C_{ri}^0) \geq t_{ij}^0$, otherwise $\Omega_{ij}^r := 0$ and set $t_{ij} = t_{ij}^0$ and $x_{ij}^{(l)} = 0$.

Step 1: Set the link likelihood

For each link ij, set $a_{ij} := \exp(-\theta\, t_{ij})$

Steps 2, 3, and 4 are to be run for each origin node r.

Step 2: Forward pass

(a) Set all wl_{ij}^r and wn_n^r to 0. Set wn_r^r:=1

(b) For each link ij taken in the order of increasing reference access time from r, if $\Omega_{ij}^r = 1$ then compute $wl_{ij}^r := a_{ij} wn_i^r$ and add wl_{ij}^r to wn_j^r, otherwise do nothing

Step 3: Backward pass

(a) Set $xn_s^r := Q_0^{rs}$ for the destination node s.

(b) For each link ij taken in the order of decreasing reference access time from r, if $\Omega_{ij}^r = 1$ then compute $xl_{ij}^r := xn_j^r wl_{ij}^r / wn_j^r$ and add xl_{ij}^r to xn_i^r, otherwise do nothing.

Step 4: Contribution to total link-flows

Calculate $x_{ij}^{(l)} := x_{ij}^{(l)} + xl_{ij}^{r}$.

Step 5: Link travel time and link impedances update

Set $t_{ij}^{(l)} \coloneqq t_{ij}(x_{ij}^{(l)})$ and $a_{ij} \coloneqq \exp(-\theta\, t_{ij}^{(l)})$.

Step 6: Direction finding

Based on link travel time $\{t_{ij}^{(l)}\}$ and a_{ij}, steps 6.1, 6.2, 6.3 are the same as steps 2, 3, and 4 for each OD pair, respectively. Yielding an auxiliary link flow pattern $\{yx_{ij}^{(l)}\}$.

Step 7: Link flow update

(a) Choose a sequence of real numbers such that $(0 < \lambda^{(l)} < 1)$.

(b) Let $rg_{ij}^{(l)} = yx_{ij}^{(l)} - x_{ij}^{(l)}$.

(c) Set $x_{ij}^{(l+1)} = x_{ij}^{(l)} + \lambda^{(l)} g_{ij}^{(l)}$.

Step 8: Stopping the test

If $max_{ij}\{rg_{ij}^{(l)}\} \leq \sigma$, stop. The solution is $\{x_{ij}^{(l)}\}$. Otherwise, set $l := l + 1$ and go to step 5.

2.2.3 Overview of extensions logit-based traffic assignment

Although widely used, assumptions of the logit-based traffic assignment consisting of independent distribution and identical variance have also been criticized. The independent distribution causes the route overlapping problem and the identical-variance assumption triggers route length problem in the route choice context (Chikaraishi and Yaginuma[17]).

The first drawback of the MNL is also called the IIA property (Independence of Irrelevant Alternative) that causes significant erroneous results on overlapping or correlated routes. The flow on the overlapping routes are overestimated because the network topology is not considered in the MNL model (Sheffi[3]). Considering the simple network with an OD pair and four-link shown in **Figure 2.1**. There are three possible routes, two of them having a common link. Assuming the travel time on each of these routes is 20 min. The flow on all the routes would be equal by using the MNL model. There is no problem with this result if the amount of overlap

between two bottom routes is relatively small (**Figure 2.1a**) in which most users perceive the three routes as three distinct alternatives. However, if the amount of overlap is large, shown in **Figure 2.1b**, users may perceive the bottom two routes as a single alternative. As a consequence, the flow on heavily overlapping routes is overestimated.

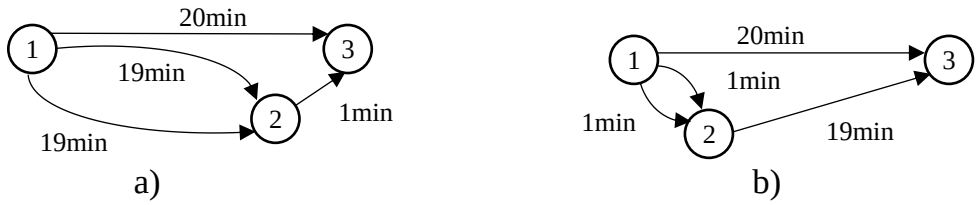


Figure 2.1 A network representing the overlapping problem of the MNL model

The second drawback of the MNL can cause the route length problem in the route choice probability context. For example, consider two routes of one origin-destination (OD) pair in which the difference in travel times between the routes is 10 minutes shown in **Figure 2.2**.

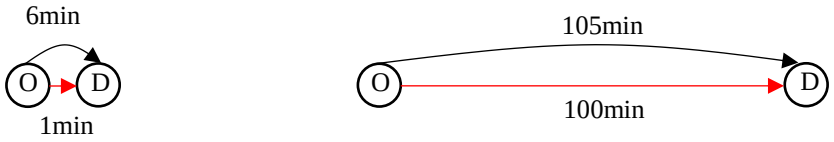


Figure 2.2 A network representing the route length problem of the MNL model

The logit model assumes that a user can perceive the 5-minutes' difference independently from the length of the route. There is no problem if the travel time is small (e.g., when the two routes' travel time is 1 and 6, most users may use the first route). However, when the travel times are 100 and 105, there may be no difference between the two routes' choice probabilities because the variance in perceived travel time becomes larger as the travel time lengthens[3)].

To overcome the route overlapping problem, modifying the single-level tree structure of the original MNL model has been widely used. For example, the C-logit model (Cascetta et al.[18)]; Russo and Vitetta[19)]) and path-size logit model (Ben-Avika and Bierlaire[20)]; Ramming[21)]) are typical cases. However, the very similar patterns of MNL and its modifications were shown by Prashker and Bekhor[22)]. In

another approach, the error component of the utility function capturing the similarity among routes has been assumed. Logit kernel (or mixed logit) model for route choice presented by Bekhor et al.[23] is an example of this approach in which either Monte Carlo simulation or numerical integration methods are required for dealing with the choice probability. However, the mixed logit route choice model with a path-based approach was solved in an ineffective method (Pravinvongvuth et al.[24]). Once the multivariate extreme value (MEV) theory with the framework of random utility maximization was proposed by McFadden[25], MEV-based route choice models consisting of the CNL model (Prashker and Bekhor[26]; Vovsha and Bekhor[27]; Bekhor et al. [28]; Papola[29]), the generalized nested logit (GNL) model (Bekhor and Prashker[30]), and the paired combinatorial logit (PCL) model (Prashker and Bekhor[26),31),32]; Gliebe et al.[33]) have been commonly applied. This approach used a two-level tree structure including the marginal and conditional probabilities and allowed a route being belong to more than one nest. While the CNL and the GNL considered a nest was a link (that is why it is also called link-nested logit model), the PCL model assumed a nest was a route pair. Because the number of route pairs (nests) in the PCL is very big in a large-scale traffic network, it decreased the applicability of this model. In recent years, some network MEV-based route choice models relaxing explicit route enumeration were proposed (Papola and Marzano[34]; Hara and Akamatsu[35]). Mai et al.[36] also proposed a nested recursive logit model for route choice analysis that using a system of non-linear equations to overcome the IIA property and based on a recursive logit model without sampling any choice sets of routes (Fosgerau et al.[37]). However, it very difficult to apply to the traffic equilibrium problem because it assumed the existence of the cyclic route and required the calculation of the inverse matrix. All of the above, the CNL model with a flexible correlation structure keeping the closed-form structure of the MNL model has been still valuable in representing the SUE traffic assignment problem. The details of the CNL model will be considered as follows:

In the CNL for route choice context, each link on the traffic network is considered as a nest and each route could belong to more than one nest. Let we denote $\alpha_{ij,k}^{rs}$ the inclusion coefficient of route k of each OD pair rs in the nest (link) ij. The similarity between routes of each OD pair rs can be measured only

on the network topology. It is assumed that the inclusion coefficient $\alpha_{ij,k}^{rs}$ expresses a ratio of each link physical length (or free-flow time) on each route in which this coefficient is considered in each OD pair as follows

$$\alpha_{ij,k}^{rs} = \frac{L_{ij}}{L_k^{rs}} \delta_{ij,k}^{rs} \tag{2.21}$$

where L_{ij} is the link physical length (or free-flow time) for link ij, L_k^{rs} is the route physical length (or free-flow time) for route k of OD pair rs, $\delta_{ij,k}^{rs}$ is the link-route incidence variable. $\delta_{ij,k}^{rs} = 1$ if link ij in on route k of OD pair rs and 0 otherwise.

If the MEV theory is applied and the utility of route k of OD pair rs is assumed as $-\theta c_k^{rs}$, the probability of choosing route k of OD pair rs is as the following equation:

$$\begin{aligned} p_k^{rs} &= \sum_{ij \in A} p_{(k/ij)}^{rs} pm_{ij}^{rs} \\ &= \frac{\sum_{ij \in A} \left(\left\{ \alpha_{ij,k}^{rs} \exp(-\theta c_k^{rs}) \right\}^{\frac{1}{\mu}} \left[\sum_{k \in K_{ij}^{rs}} \left\{ \alpha_{ij,k}^{rs} \exp(-\theta c_k^{rs}) \right\}^{\frac{1}{\mu}} \right]^{\mu-1} \right)}{\sum_{ij \in A} \left[\sum_{k \in K_{ij}^{rs}} \left\{ \alpha_{ij,k}^{rs} \exp(-\theta c_k^{rs}) \right\}^{\frac{1}{\mu}} \right]^{\mu}} \end{aligned} \tag{2.22}$$

where pm_{ij}^{rs} is the marginal probability of choosing nest ij of OD pair rs, $p_{(k/ij)}^{rs}$ is the conditional probability of choosing route k of OD pair rs if nest ij is selected, μ is the degree of nesting, $0 < \mu \leq 1$, c_k^{rs} is the travel time on route k of OD pair rs, K_{ij}^{rs} is the set of the routes included in the nest (link) ij of OD pair rs.

pm_{ij}^{rs} and $p_{(k/ij)}^{rs}$ are given by the following equations:

$$pm_{ij}^{rs} = \frac{\left[\sum_{k \in K_{ij}^{rs}} \left\{ \alpha_{ij,k}^{rs} \exp(-\theta c_k^{rs}) \right\}^{\frac{1}{\mu}} \right]^{\mu}}{\sum_{ij \in A} \left[\sum_{k \in K_{ij}^{rs}} \left\{ \alpha_{ij,k}^{rs} \exp(-\theta c_k^{rs}) \right\}^{\frac{1}{\mu}} \right]^{\mu}} \tag{2.23}$$

$$p_{(k/ij)}^{rs} = \frac{\left\{\alpha_{ij,k}^{rs} \exp(-\theta c_k^{rs})\right\}^{\frac{1}{\mu}}}{\left[\sum_{k \in K_{ij}^{rs}} \left\{\alpha_{ij,k}^{rs} \exp(-\theta c_k^{rs})\right\}^{\frac{1}{\mu}}\right]^{\mu}} \tag{2.24}$$

In the logit-based traffic assignment model with fixed travel demand,

$$f_k^{rs} = Q_0^{rs} p_k^{rs}. \tag{2.25}$$

It is clear that flow conservation is automatically satisfied:

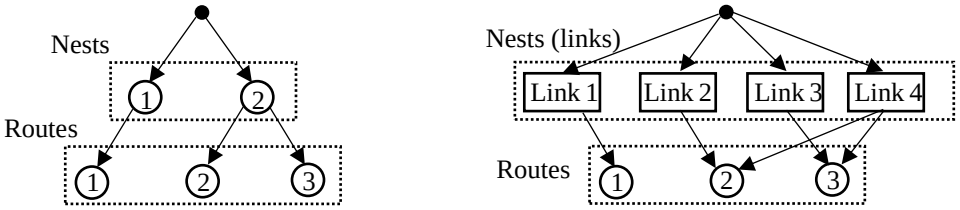
$$Q_0^{rs} = \sum_{k \in K^{rs}} f_k^{rs} \ \forall rs \in W. \tag{2.26}$$

And, the relationship between the link flow and route flow; the route cost and link cost are represented similarly in the logit-based model (**Equations (2.4) - (2.6)**). From there, a fixed-point problem with route flow of the SUE CNL model traffic assignment is considered as follows:

$$f_k^{rs} = Q_0^{rs} \frac{\sum_{ij \in A} \left(\left\{\alpha_{ij,k}^{rs} \exp(-\theta c_k^{rs}(\mathbf{f}))\right\}^{\frac{1}{\mu}} \left[\sum_{k \in K_{ij}^{rs}} \left\{\alpha_{ij,k}^{rs} \exp(-\theta c_k^{rs}(\mathbf{f}))\right\}^{\frac{1}{\mu}}\right]^{\mu-1} \right)}{\sum_{ij \in A} \left[\sum_{k \in K_{ij}^{rs}} \left\{\alpha_{ij,k}^{rs} \exp(-\theta c_k^{rs}(\mathbf{f}))\right\}^{\frac{1}{\mu}}\right]^{\mu}}. \tag{2.27}$$

The well-known MSA method could be utilized to achieve equilibrium[28].

To illustrate the CNL model in the route choice context and show how this model can solve the overlapping problem, we consider the small network shown in **Figure 2.1**. Denoting link 1 is from node 1 to node 2, link 4 is from node 2 to node 3. Because there are 2 links from node 1 to node 2, to distinguish these links we will use the symbol link 2 for the link above and link 3 for the link below. There are 3 possible routes from origin node 1 to destination node 3: route 1 includes link 1, route 2 includes link 2 and link 4, route 3 includes link 3 and link 4. If we applied the nested logit model for the route choice problem, this network can be divided into two nests. Route 1 will be placed in a nest because it does not have common links with the other two routes. The remaining 2 routes will join the remaining nest because they have the common link. However, the link-nested structure of the CNL model is different with 4 nests in concordance with 4 links.



a) b)

Figure 2.3 The difference in the nest structure of the nested logit model and CNL model in the small network. (a) nested logit model, (b) CNL model

For further comparison, the route choice probabilities will be calculated in the case shown in **Figure 2.1b**. Some parameters are set for the computation reason in which $\mu = 0.1$ and $\theta = 1$. The results are shown in the following table:

Table 2.1 Route probabilities comparison

Route	**Probability of choosing the route in different models**		
	MNL	**Nested logit**	**CNL**
1	0.333	0.483	0.472
2	0.333	0.259	0.264
3	0.333	0.259	0.264

As can be seen, while the MNL cannot consider the overlapping problem, the nested-logit model and CNL can treat this problem. In detail, the probability of choosing route 1 in the nested-logit model and CNL is greater than that in the MNL model because of the overlap between routes 2 and 3. To show that the CNL model can treat differential degrees of overlap, the travel time of link 4 (overlap link) is changed from 1 min to 19 min but the total travel time on routes 2 and 3 are unchanged (equals to 20 min). The result of the probability of choosing route 1 is represented as follows:

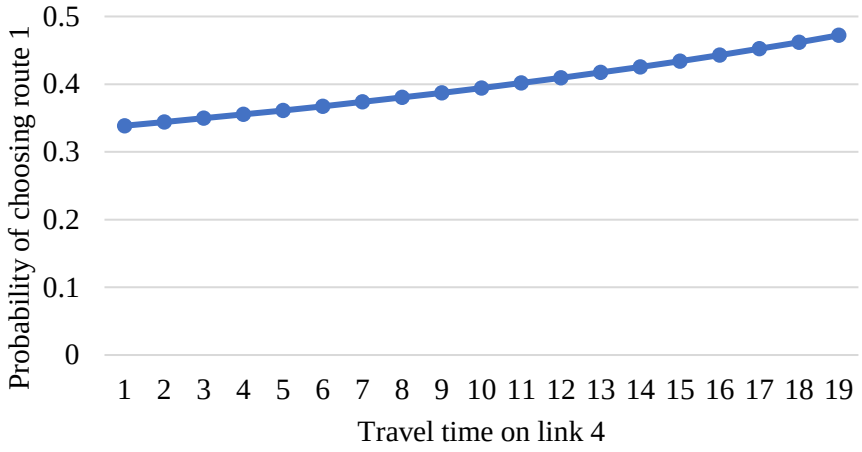


Figure 2.4 Choice probability of route 1

Figure 2.4 performs the influence of overlap route length on the CNL model. The probability of choosing route 1 increases when the length of overlap between route 2 and route 3 increases. For example, in case the nest of link 4 contains only 5 percent of routes 2 and 3 (travel time on link 4 equals to 1 min), the CNL model approaches the MNL. In contrast, in case the nest of link 4 contains 95 percent of routes 2 and 3 (travel time on link 4 equals to 19 min), these routes may be perceived as a single route and the probability between route 1 (0.47) and the two overlapping routes 2 and 3 taken together (0.53) are almost the same. From this example, it can be seen that the CNL model considering a link is a nest that can solve the overlapping problem and can be easily applied.

When applying to a large-scale network, the case of each route belong to exactly one nest (perfect nest structure) in the nested-logit model is impossible. The GNL model, meanwhile, requires a large number of parameters in which μ is different for each nest. For these reasons, CNL will be noticed to solve the problem of overlapping. Of course, there are some other models such as network MEV-based route choice models [33), 34)] that can effectively solve the overlapping problem. However, the CNL algorithm will still be chosen for its simplicity, proximity to the MNL model, and its applicability to the semi-DTA model used in this research.

The next problem is how to relax the identical-variance assumption. Castillo et al.[38)] proposed weibit model in which a closed-form model based on the Weibull distribution was used. And applying weibit model to route-choice contexts was examined by Kitthamkesorn and Chen[39), 40)]. However, the weibit model pre-

determined the perception variance. Because of varies according to the experience of users, the perception variance needs to be estimated from data[41]. To incorporate the logit and weibit models and make the travel-behavior analysis more flexible, a single closed-form expression derived from the MEV distribution was proposed by Nakayama[42] and Nakayama and Chikaraishi[43]; the so-called *q*-generalized MEV model. From there, the application of this model to large-scale networks with a link-based approach has been a great challenge. Details of the *q*-generalized logit model will be mentioned in Chapter 5.

The use of the MNL model brings the stable results even when we change the network structure such as merging some parts of links without changing the total travel times that causes the route overlapping problem or shortening the link travel times without changing the travel different between two links that cause the route length problem. The CNL model can help to solve the route overlapping problem and the q-generalized logit model could help to solve the route length problem. Hence, the use of CNL and q-generalized logit models will help to consider the effectiveness of the projects where the MNL model cannot take into account different aspects.

2.2.4 The existing literature on the sensitivity analysis method

In general, traffic assignment for large-scale road networks requires a huge amount of computational cost. Besides, the mathematical problem with equilibrium constraints (MPEC)[44), 45), 46)] call for repeated traffic assignment. Thus, it is very important to reduce the computational cost while ensuring the accuracy of traffic assignment results. Besides, since there are numerous uncertainty parameters in functions (e.g. link performance functions) providing a mathematical model for traffic network equilibrium analysis, it is also important to justify the strength of the model to such uncertainties and to recognize those parameters to which the equilibrium solution is most sensitive. To solve two problems, sensitivity analysis for traffic network equilibrium problems is taken into consideration.

The research to date on computational methods and applications of sensitivity analysis has been mainly for the deterministic UE and SUE traffic assignment problems. The well-known sensitivity analysis method for the UE traffic assignment was proposed by Tobin and Friesz[47]. Kyparisis[48] and Qui and

Magnanti[49], among many others, developed sensitivity analysis for general variational inequalities. Based on the method of Tobin and Friesz, Yang[50] introduced sensitivity analysis for queuing equilibrium network flow and the derivatives of equilibrium link flows and equilibrium queuing times for traffic control parameters were presented. Sensitivity analysis for others expansion of the UE traffic assignment has also been proposed and applied for solving network design (Kim and Suh[51]), bi-level traffic control (Yang et al.[52]; Yang and Yagar,[53]), congestion pricing problems (Yang and Lam[54]; Yang[55]; Yang and Bell[56]). Since Daganzo and Sheffi[5] proposed the principle of SUE, sensitivity analysis for the SUE model is also an important problem. While Clark and Watling[57] proposed the mathematical programming method for sensitivity analysis of the Probit-based SUE model, Ying and Miyagi[58] presented the research on sensitivity analysis of the logit-based SUE model with a dual approach. In the paper of Ying and Miyagi[58], the MNL SUE model with fixed travel demand was considered and the derivatives of link costs and flows with respect to some parameters including free-flow travel time and travel demand parameters were calculated based on Dial's algorithm. Extension of this method for the traffic network with elastic travel demand was proposed by Ying and Miyagi[59]. Also, the sensitivity analysis method with elastic demand for optimal road network pricing was presented in the paper of Ying[60]. Another paper of sensitivity analysis of SUE flows in a combined transport system was proposed by Ying and Yang[61]. Recently, the sensitivity analysis of the link-capacitated SUE model was conducted by Ji et al. [62]. Since some extensions of logit-based traffic assignment have been proposed, finding a sensitivity-analysis method for these models remains an outstanding challenge for traffic-network modeling.

2.2.5 Dynamic traffic assignment model

In DTA models, traffic flow is dynamically handled and the changes in traffic conditions from time to time are taken into account, so it can be said that a reasonable result can be obtained. Given the huge number of DTA-related papers, various review studies have been conducted to summarize the literature and to identify future research directions of DTA models. There are at least nine conducted DTA reviews. While the DTA models and solution method developed before 1991 was summarized by Cascetta, Cantarella[63], a helpful review of the

numerous part studies on DTA and examined DTA papers published before 2000 was conducted by Peeta and Ziliaskopoulos[64]. In another approach, analytical DTA formulations focusing on the variational inequality approach were reviewed by Boyce et al.[65]. In the paper of Szeto and Lo[66], comparing the properties of DTA with different forms of traffic flow models, discussing their implications, and suggesting future research directions were presented. Besides, this paper discussed the properties of DTA problems with and without considering the effects of spatial queues, and their implications. Next, Mun[67] focused on the traffic flow component of DTA and Jeihani[65] addressed the DTA models used in some well-known computer programs such as CONTRAM, TRANSIMS, PARAMICS, DYNASMART, DynaMIT, VISSIM, etc. Furthermore, Szeto et al.[68] reviewed one type of DTA model, namely cell-based dynamic equilibrium models. To obtain realistic solutions for practical applications, one of the important components of DTA models is the travel choice principle. Szeto, Wong[69] addressed this travel choice principle should be behaviorally sound and the route/departure time choice behavior of users needs to be reflected. Advances in the principles of travel choice and the various DTA classifications were also outlined in the paper of Szeto and Wong[70]. Moreover, the implications of the travel choice principles for the solution existence and uniqueness of the DTA models and the integration of the travel choice and traffic flow components were discussed in this paper. The authors also briefly considered the limitations of existing models and mentioned some trends of DTA models. Then, the recent methodological advances in environmentally sustainable road transportation applications of DTA models were summarized and examined by Wang et al.[71].

According to the approach method, DTA can be broadly divided into the following two types. The first approach is reproducing relatively micro traffic conditions by computer simulation; also named “traffic flow simulation” or “microsimulation” and the second one is expanding mathematical models such as UE traffic assignment models to handle dynamic phenomena.

With the recent improvement of computational capabilities, microsimulation is becoming increasingly popular and considered a practical tool. However, the following issues have appeared:

- If too much reproducibility of the current situation is performed, it will

become complicated with many rules and parameters, and become a “black box” that is hard to see the contents other than the modeler.

- Since different results may be output depending on the simulator, it is difficult to know which result the best choice is.

- Even if a reasonable calculation result is obtained in each case, there is a possibility that the characteristics such as output stability and sensitivity are not comprehensively grasped.

In another approach, the merits of mathematical models are used, and the properties of the solution can be grasped analytically. For example, dynamic user equilibrium is expressed as an extension of Wardrop's first principle, system optimal DTA is an extension of Wardrop’s second principle and dynamic stochastic equilibrium may be similarly defined in the SUE model (Peeta and Ziliaskopoulos[64]).

In recent years, although modelers increasingly prefer to use dynamic assignment models, practitioners may be disturbed by the variety of concepts and applications. Besides, the requirements of computational capacity and input data for applying the DTA model need to be taken into consideration. The willingness to apply DTA technology into a large-scale network is hampered by these hinders.

2.2.6 Overview of existing literature on semi-dynamic traffic assignment

The daily STA widely used in practice defined on a relatively long time-of-day period and the traffic condition has been described by the average or steady-state travel time on a link as a function of the volume of traffic on the link. However, some problems with this approach could be shown as follows:

- It is not always appropriate to assume daily steadiness in traffic flow.

- It is cannot be used to evaluate measures by period, such as road fees by period, traffic restrictions by period, etc.

- It is difficult to rationally set the daily traffic capacity of the link.

- It is could not adequately represent traffic flow, especially time-varying congestion phenomena because traffic conditions change significantly with time of day. Although time-slided static traffic assignment divides the day into several periods and static traffic assignment is made in each period, the flow propagation

is not considered. Especially at peak times, because of the congestion, it is unreasonable to assume that all travel demand can arrive at its destination within a single assignment.

Methods that can deal with these problems include traffic flow simulation and the DTA model. However, these can be microscopically analyzed in part of the network, but it is difficult to apply to a large-scale network in the metropolitan area. Moreover, it is difficult to say that a general calculation theory has been established. DTA models require a large calculation load and a detailed OD matrix.

As a combination of STA and DTA, the time-of-day traffic assignment model, also known as the semi-DTA model has been studied. The position of semi-DTA models was presented by Nakayama and Connors[72]. Accordingly, the semi-DTA models have been very similar to discrete-time DTA models. This approach takes advantages of STA models while incorporating the dynamic aspects of traffic phenomena. To the best of our knowledge, some semi-DTA models have been proposed by Fujita et al. [73), 74)], Miyagi et al. [75)], and Akamatsu et al.[76)], also known as dynamic UE with a large discrete period. Recently, Nakayama and Connors[72)] presented the existence and uniqueness of the equilibrium solution in a semi-DTA of UE. In another approach, Bliemer et al.[77)]; Brederode et al.[78)] proposed quasi-DTA models in which the time dimension was excluded by assuming stationary travel demand (as in static models) and capacity constraints were included (as in dynamic models). However, there was only a single period other than the multiple periods and the flow propagation was not considered in this approach. Indeed, such quasi-dynamic models are STA models with capacity constraints. The semi-DTA approach is different from these. In semi-DTA models, the day is divided into several periods, the state of the driver and the network are made steady in each period, and the propagation phenomenon between periods can be considered. Many of the semi-DTA models assume that the amount of traffic that could not reach the destination within the period is the residual traffic volume, which affects the next period. Residual flow is the flow on a link that cannot exit the link in a given period. This flow is propagated to the subsequent period; the so-called flow propagation. The model can be roughly divided into three types depending on how the residual traffic is handled and what kind of propagation phenomenon is considered between periods. Three categories are as follows:

- The OD demand modification approach (Fujita et al.[73] and Miyagi and Makimura [75]): The residual flow remains on each link at the end of the period, and this is added to the OD demand for the next period for convenience. Because traffic equilibrium in each period is formulated as an optimization problem with elastic demand, the computational cost in each period is approximately equal to the ordinary STA model with elastic demand. Thus, it is relatively easy to apply the demand modification approach to large-scale networks.

- The link flow approach (Fujita et al.[74]): Vehicles that could not reach their destination within the period remain on each link at the end of this period, and they will affect the link performance in the subsequent period. The formulation consists of the variational problem that requires a huge amount of computational cost.

- The queue approach (Akamatsu et al.[76]): It is assumed that the bottleneck on the network is affected by the size of the bottleneck capacity and the inflow traffic. The queue starting from the bottleneck is generated. The link performance in the following period is affected. The vertical queue is assumed, and the model is formulated as an optimization problem.

The semi-DTA models maintain the characteristics of STA models, such as the small computational load and uniqueness of the solution, while handling the traffic flow and the propagation of the traffic flow between the periods. The need for developing semi-DTA for applying to various situations such as computational cost, data accuracy, and others is very necessary. Once the result of the SUE is more reasonable, realistic, and general than UE (Sheffi [3]), developing semi-DTA models with SUE is crucial to contribute to advanced traffic assignment models. Ha et al.[79] and Itagaki et al.[80] presented a semi-DTA with SUE based on sensitivity analysis. At each period, they computed a static SUE and the residual flow for flow propagation. When the residual flow is eliminated from the current period, it affects the link travel time and link flow at an equilibrium state. They used a sensitivity analysis method to solve this problem. However, because they used the route-based approach that consumes computational time and computer memory, this approach is very difficult to apply in practice. Besides, because the residual flows were assumed and calculated at the static SUE, this approach was essentially the STA model that takes into account the effect of the residual flow. The equilibrium solution of semi-DTA with SUE was not presented.

The semi-DTA model is very close to the discrete-time DTA model. Differences between the semi-DTA model and the DTA model are shown in the following table

Table 2.2 The differences between the semi-DTA model and the DTA model

Semi-DTA	**DTA**
➢ Multiple periods (e.g. from 15min to 90min length in each period). ➢ Applying a sequence of STA in which route choice proportions are assumed stationary during each period. ➢ Need a coarse OD matrix. ➢ Residual flow is propagated to the next period. However, some DTA requirements such as physical queues, First-In-First-Out principle, vehicle propagation speeds, etc. are not described explicitly.	➢ Many smaller periods (e.g. smaller 15min in each period). ➢ Representing time-varying travel demand and network loading is much more complicated. ➢ Need a detailed OD matrix. ➢ Flow propagation between small time steps is considered in more sophisticated aspects.

2.3 The position of this research

Sensitivity analysis plays an important role in the traffic assignment equilibrium problem. Many sensitivity analysis methods have been proposed for reducing the computational cost. However, most of them for traffic assignment with UE problem. Traffic assignment with the SUE problem with more reasonable results than the UE problem has been received numerous attention from researchers. The well-known sensitivity analysis for the SUE traffic assignment was proposed by Ying and Miyagi[58]. This approach derived from the minimization problem presented by Daganzo[10] and based on Dial's algorithm to calculate gradient matrices. Nevertheless, when applying this method to the actual Kanazawa road network, several problems occurred. First, the weakness of the Dial's algorithm with the instability of the route-set, as indicated by Leurent[15], made the convergence of SUE traffic assignment impossible. Because it is impossible to have the SUE solution, it is also not feasible to apply the Dial's algorithm in the sensitivity analysis method. We have replaced the use of Dial's algorithm with the STOCH3 algorithm and applying Ying and Miyagi's calculation procedure[58], but the results of calculating the gradient matrices of link traffic flows with respect to some parameters do not guarantee the accuracy. Secondly, Ying and Miyagi[58] proposed two concepts consisting of apparent derivative and true partial derivative. While the apparent derivative of link traffic flow with respect to parameter denoted the derivative of link traffic flow as an explicit function in the parameter, the true partial derivative was the ratio of change of link traffic flow with respect to a small change of the parameter. These definitions may confuse practice users. In the new approach, we derive from the nature of the fixed-point problem in the SUE traffic assignment and from applying the properties of the derivative of the implicit function and the chain rule of the derivative to obtain the formula for calculating the gradient matrices. Besides, the calculation process will be different from the process of Ying and Miyagi[58]. Based on the STOCH3 algorithm and Depth-first search[81] (DFS) algorithm, the calculation of gradient matrices will be proposed with high efficiency of computational accuracy and computational cost. In the process of researching and applying the STOCH3 algorithm, a tree search algorithm called the DFS algorithm can be regarded as an effective integration tool with the STOCH3 algorithm.

Moreover, the semi-DTA model with advantages of computation time, applicability, and utilization of existing OD data has been seen as an effective solution to combine STA and DTA models. In this study, we propose semi-DTA with the SUE model that takes into account the congested spatial-temporal movement. Consequently, we will show a double-looped fixed-point problem in semi-DTA with the SUE model. Because the computational cost for the equilibrium solution of this model is very large, we propose the approximation method using the sensitivity analysis method. Besides, we will relax the route-based approach, which has been raised as a problem in the studies of Ha et al.[79] and Itagaki et al.[80]. The sensitivity analysis method using the link-based STOCH3 algorithm will be used to reduce the computational time and computer memory. Only link-based and node-based variables are used in the calculation process to achieve computational efficiency while ensuring the accuracy of the method. The computational efficiency will be shown in the application to small example and Kanazawa road network.

Besides, some extensions of logit-based models such as the CNL model and the q-generalized logit model were proposed for solving the weaknesses of the MNL based model. Thus, we also propose the sensitivity analysis for these approaches with an integrated algorithm based on the STOCH3 efficient route definition and DFS algorithm. Finally, we expand the semi-DTA with some extensions of the logit-based model that handle the weaknesses of the MNL model. The DFS algorithm will be scrutinized to figure out good results on computation time and memory savings when studying these expansion models.

References

1. J. G. Wardrop, “Some Theoretical Aspects of Road Traffic Research,” *Proc. Inst. Civ. Eng.*, vol. 1, no. 3, pp. 325–362, 1952.
2. J. de D. Ortúzar and L. G. Willumsen, *Modelling Transport*. 2011.
3. Y. Sheffi, *Urban Transportation Networks: Equilibrium Analysis with Mathematical Programming Methods*. 1985.
4. E.W. Dijkstra. 1959, “A Note on Two Problems in Connexion with Graphs,” *Numerische Mathematik*, vol. 1, no. 1, pp. 269–71, 1959.
5. C. F. Daganzo and Y. Sheffi, “On Stochastic Models of Traffic Assignment,” *Transp. Sci.*, vol. 11, no. 3, pp. 253–274, 1977.
6. R. B. Dial, “A Probabilistic Multipath Traffic Assignment Model which Obviates Path Enumeration,” *Transp. Res.*, vol. 5, pp. 83–111, 1971.
7. M. G. H. Bell, “Alternatives to Dial’s Logit Assignment Algorithm,” *Transp. Res. Part B*, vol. 29B, no. 4, pp. 287–295, 1995.
8. J. E. Burrell, “Multiple Route Assignment and Its Application to Capacity Restraint,” *Proc. Fourth Int. Symp. Theory Traffic Flow*, 1968.
9. C. Fisk, “Some Developments in Equilibrium Traffic Assignment,” *Transp. Res. Part B*, vol. 14B, pp. 243–255, 1980.
10. C. F. Daganzo, “Unconstrained Extremal Formulation of Some Transportation Equilibrium Problems,” *Transp. Sci.*, vol. 16, no. 3, pp. 332–360, 1982.
11. W. B. Powell and Y. Sheffi, “The convergence of equilibrium algorithms with predetermined step sizes,” *Transp. Sci.*, vol. 16, no. 1, pp. 45–55, 1982.
12. T. Akamatsu, “Decomposition of Path Choice Entropy in General Transport Networks,” *Transp. Sci.*, vol. 31, no. 4, pp. 349–362, 1997.
13. Guidebook on Traffic Demand Forecasting, Volume II: User Equilibrium Assignment Models and Their Developments, Japan Society of Civil Engineers (In Japanese), 2006.
14. Damberg, J. T. Lundgren, and M. Patriksson, “An algorithm for the

stochastic user equilibrium problem," *Transp. Res. Part B Methodol.*, vol. 30, no. 2, pp. 115–131, 1996.

15. F. M. Leurent, "Curbing the Computational Difficulty of the Logit Equilibrium Assignment Model," *Transp. Res. Part B*, vol. 31, no. 4, pp. 315–326, 1997.
16. F. Tagliacozzo and F. Pirzio, "Assignment models and urban path selection criteria: Results of a survey of the behavior of road users," *Transp. Res.*, vol. 7, pp. 313–329, 1973.
17. M. Chikaraishi and H. Yaginuma, "Recent development of discrete choice models," *15th Summer course Behav. Model. Transp. Networks, Univ. Tokyo*, 2016.
18. E. Cascetta, A. Nuzzolo, F. Russo, and A. Vitetta, "A Modified Logit Route Choice Model Overcoming Path Overlapping Problems. Specification and Some Calibration Results for Interurban Networks," *Transportation and Traffic Theory*. pp. 697–711, 1996.
19. F. Russo and A. Vitetta, "An Assignment Model with Modified Logit, which Obviates Enumeration and Overlapping Problems," *Transportation (Amst).*, vol. 30, no. 2, pp. 177–201, 2003.
20. M. Ben-Akiva and M. Bierlaire, "Discrete Choice Methods and their Applications to Short Term Travel Decisions," pp. 5–33, 1999.
21. M. S. Ramming, "Network Knowledge and Route Choice," *Ph.D. Diss. Dep. Civ. Environ. Eng. Massachusetts Inst. Technol.*, 2002.
22. J. N. Prashker and S. Bekhor, "Route Choice Models Used in the Stochastic User Equilibrium Problem: A Review," *Transp. Rev.*, vol. 24, no. 4, pp. 437–463, 2004.
23. S. Bekhor, M. E. Ben-Akiva, and M. S. Ramming, "Adaptation of logit kernel to route choice situation," *Transp. Res. Rec.*, no. 1805, pp. 78–85, 2002.
24. S. Pravinvongvuth and A. Chen, "Adaptation of the Paired Combinatorial Logit Model to the Route Choice Problem," *Transportmetrica*, vol. 1, no. 3, pp. 223–240, 2005.

25. McFadden, *Modelling the Choice of Residential Location*. 1978.

26. J. N. Prashker and S. Bekhor, “Investigation of Stochastic Network Loading Procedures,” *Transp. Res. Rec. J. Transp. Res. Board*, vol. 1645, pp. 94–102, 1998.

27. P. Vovsha and S. Bekhor, “Link-Nested Logit Model of Route Choice: Overcoming Route Overlapping Problem,” *Transp. Res. Rec. J. Transp. Res. Board*, vol. 1645, no. 1, pp. 133–142, Jan. 1998.

28. S. Bekhor, L. Reznikova, and T. Toledo, “Application of Cross-Nested Logit Route Choice Model in Stochastic User Equilibrium Traffic Assignment,” *Transp. Res. Rec. J. Transp. Res. Board*, vol. 2003, no. 1, pp. 41–49, 2007.

29. Papola, “Some Developments on the Cross-nested Logit Model,” *Transp. Res. Part B Methodol.*, vol. 38, pp. 833–851, 2004.

30. S. Bekhor and J. N. Prashker, “Stochastic User Equilibrium Formulation for Generalized Nested Logit Model,” *Transp. Res. Rec. J. Transp. Res. Board*, vol. 1752, pp. 84–90, 2001.

31. J. N. Prashker and S. Bekhor, “Stochastic User-Equilibrium Formulations for Extended-Logit Assignment Models,” *Transp. Res. Rec. J. Transp. Res. Board*, vol. 1676, pp. 145–152, 1999.

32. J. N. Prashker and S. Bekhor, “Congestion, Stochastic, and Similarity Effects in Stochastic: User-Equilibrium Models,” *Transp. Res. Rec. J. Transp. Res. Board*, vol. 1733, pp. 80–87, 2000.

33. J. P. Gliebe, F. S. Koppleman, and A. Ziliaskopoulos, “Route Choice Using a Paired Combinatorial Logit Model,” *Present. 78th Annu. Meet. Transp. Res. Board*, 1999.

34. Papola and V. Marzano, “A network generalized extreme value model for route choice allowing implicit route enumeration,” *Comput. Civ. Infrastruct. Eng.*, vol. 28, no. 8, pp. 560–580, 2013.

35. Y. Hara and T. Akamastu, “Stochastic User Equilibrium Traffic Assignment with a Network GEV Based Route Choice Model,” *J. Japan Soc. Civ. Eng. D3 (Civil Eng. Stud.*, vol. 70, no. 5, pp. 611–620, 2014 (In Japanese).

36. T. Mai, M. Fosgerau, and E. Frejinger, “A Nested Recursive Logit Model for Route Choice Analysis,” *Transp. Res. Part B Methodol.*, vol. 75, no. August, pp. 100–112, 2015.

37. M. Fosgerau, E. Frejinger, and A. Karlstrom, “A link based network route choice model with unrestricted choice set,” *Transp. Res. Part B Methodol.*, vol. 56, pp. 70–80, 2013.

38. E. Castillo, J. M. Menéndez, P. Jiménez, and A. Rivas, “Closed Form Expressions for Choice Probabilities in the Weibull Case,” *Transp. Res. Part B*, vol. 42, pp. 373–380, 2008.

39. S. Kitthamkesorn and A. Chen, “A Path-size Weibit Stochastic User Equilibrium Model,” *Transp. Res. Part B*, vol. 57, pp. 378–397, 2013.

40. S. Kitthamkesorn and A. Chen, “Unconstrained Weibit Stochastic User Equilibrium Model with Extensions,” *Transp. Res. Part B*, vol. 59, pp. 1–21, 2014.

41. M. Chikaraishi and S. Nakayama, “Discrete Choice Models with q-product Random Utilities,” *Transp. Res. Part B*, vol. 93, pp. 576–595, 2016.

42. S. Nakayama, “*q*-Generalized Logit Route Choice and Network Equilibrium Model,” *Proc. 20th Int. Symp. Transp. Traffic Theory. Proc. Soc. Behav. Sci.*, vol. 80, pp. 753–763, 2013.

43. S. Nakayama and M. Chikaraishi, “Unified Closed-form Expression of Logit and Weibit and its Application to a Transportation Network Equilibrium Assignment,” *Transp. Res. Part B*, vol. 81, pp. 672–685, 2015.

44. T. L. Friesz, R. L. Tobin, H.-J. Cho, and N. J. Mehta, “Sensitivity Analysis based Heuristic Algorithms for Mathematical Programs with Variational Inequality Constraints,” *Math. Program.*, vol. 48, pp. 265–284, 1990.

45. Y. Okamoto, S. Nakayama, and J. Takayama, “Network Route Aggregation with Sensitivity Analysis for Expressway Pricing,” *Presented at 91st Annual Meeting of the Transportation Research Board, Washington, D.C.*, 2012.

46. G. M. Wang, Z. Y. Gao, and M. Xu, “An MPEC Formulation and Its Cutting Constraint Algorithm for Continuous Network Design Problem with Multi-

user Classes," *Appl. Math. Model.*, vol. 38, pp. 1846–1858, 2014.

47. R. L. Tobin and T. L. Friesz, "Sensitivity Analysis for Equilibrium Network Flow," *Transp. Sci.*, vol. 22, no. 4, pp. 242–250, 1988.
48. J. Kyparisis, "Sensitivity Analysis for Variational Inequalities and Nonlinear Complementarity Problems," *Ann. Oper. Res.*, vol. 27, pp. 143–173, 1990.
49. Y. Qiu and T. L. Magnanti, "Sensitivity Analysis for Variational Inequalities Defined on Polyhedral sets," *Math. Oper. Res.*, vol. 14, no. 3, pp. 410–432, 1989.
50. H. Yang, "Sensitivity Analysis for Queuing Equilibrium Network Flow and Its Application to Traffic Control," *Math. Comput. Model.*, vol. 22, no. 4–7, pp. 247–258, 1995.
51. T. J. Kim and S. Suh, *Advanced Transport and Spatial Systems Models: Applications to Korea*. Springer-Verlag, NewYork, 1990.
52. H. Yang, S. Yagar, Y. Iida, and Y. Asakura, "An Algorithm for the Inflow Control Problem on Urban Freeway Networks with User-optimal Flows," *Transp. Res. Part B*, vol. 28B, no. 2, pp. 123–139, 1994.
53. H. Yang and S. Yagar, "Traffic Assignment and Traffic Control in General Freeway-arterial Corridor Systems," *Transp. Res. Part B*, vol. 28B, no. 6, pp. 463–486, 1994.
54. H. Yang and W. H. K. Lam, "Optimal Road Tolls Under Conditions of Queueing and Congestion," *Transp. Res. Part A*, vol. 30, no. 5, pp. 319–332, 1996.
55. H. Yang, "Sensitivity Analysis for the Elastic-Demand Network Equilibrium Problem with Applications," *Transp. Res. Part B*, vol. 31, no. 1, pp. 55–70, 1997.
56. H. Yang and M. G. H. Bell, "Traffic Restraint, Road Pricing and Network Equilibrium," *Transp. Res. Part B*, vol. 31, no. 4, pp. 303–314, 1997.
57. S. Clark and D. Watling, "Probit-Based Sensitivity Analysis for General Traffic Networks," *Transp. Res. Rec.*, vol. 1733, no. 1, pp. 88–95, 2000.
58. J. Q. Ying and T. Miyagi, "Sensitivity Analysis for Stochastic User

Equilibrium Network Flows — A Dual Approach," *Transp. Sci.*, vol. 35, no. 2, pp. 124–133, 2001.

59. J. Q. Ying and T. Miyagi, "Sensitivity Analysis for Stochastic User Equilibrium with Elastic Demand Assignment Model," *Int. Conf. Traffic Transp. Stud. 2002 July 23-25, 2002 | Guilin, China*, pp. 857–864, 2002.
60. J. Q. Ying, "Sensitivity Analysis based Method for Optimal Road Network Pricing," *Ann. Oper. Res.*, vol. 133, pp. 303–317, 2005.
61. J. Q. Ying and H. Yang, "Sensitivity Analysis of Stochastic User Equilibrium Flows in a Bi-modal Network with Application to Optimal Pricing," *Transp. Res. Part B*, vol. 39, pp. 769–795, 2005.
62. K. Ji, J. Ma, and W. Tang, "Sensitivity Analysis for Stochastic User Equilibrium Traffic assignment with Constraints," *Adv. Mech. Eng.*, vol. 9, no. 5, pp. 1–7, 2017.
63. E. Cascetta and G. E. Cantarella, "Modelling dynamics in transportation networks: State of the art and future developments," *Simul. Pract. Theory*, vol. 1, pp. 65–91, 1993.
64. S. Peeta and A. Ziliaskopoulos, "Foundations of Dynamic Traffic Assignment: The Past, the Present and the Future," *Networks Spat. Econ.*, vol. 1, no. 3/4, pp. 233–265, 2001.
65. D. Boyce, D.-H. Lee, and B. Ran, "Analytical Models of the Dynamic Traffic Assignment Problem," *Networks Spat. Econ.*, vol. 1, no. 3/4, pp. 377–390, 2001.
66. W. Y. Szeto and H. K. Lo, "Dynamic Traffic Assignment: Review and Future Research Directions," *J. Transp. Syst. Eng. Inf. Technol.*, vol. 5, no. 2005, p. 2019, 2005.
67. J. S. Mun, "Traffic performance models for dynamic traffic assignment: An assessment of existing models," *Transp. Rev.*, vol. 27, no. 2, pp. 231–249, 2007.
68. M. Jeihani, "A Review of Dynamic Traffic Assignment Computer Packages," *J. Transp. Res. Forum*, vol. 46, no. 2, pp. 35–46, 2010.

69. W. Y. Szeto, Y. Jiang, and A. Sumalee, “A cell-based model for multi-class doubly stochastic dynamic traffic assignment,” *Comput. Civ. Infrastruct. Eng.*, vol. 26, no. 8, pp. 595–611, 2011.

70. W. Y. Szeto and S. C. Wong, “Dynamic traffic assignment: Model classifications and recent advances in travel choice principles,” *Cent. Eur. J. Eng.*, vol. 2, no. 1, pp. 1–18, 2012.

71. Y. Wang, W. Y. Szeto, K. Han, and T. L. Friesz, “Dynamic traffic assignment: A review of the methodological advances for environmentally sustainable road transportation applications,” *Transp. Res. Part B Methodol.*, vol. 111, pp. 370–394, 2018.

72. S. Nakayama and R. Connors, “A Quasi-dynamic Assignment Model that Guarantees Unique Network Equilibrium,” *Transp. A Transp. Sci.*, vol. 10, no. 7, pp. 669–692, 2014.

73. M. Fujita, H. Matsui, and M. Shoshi, “Modelling of the time-of-day traffic assignment over a traffic network.,” *J. Infrastruct. Plan. Manag.*, no. 389, pp. 111–119, 1988 (in Japanese).

74. M. Fujita, K. Yamamoto, and H. Matsui, “Modelling of the time-of-day traffic assignment over a congested network.,” *Proc. Japan Soc. Civ. Eng.*, no. 407, pp. 129–138, 1989 (in Japanese).

75. T. Miyagi and K. Makimura, “A study on semi-dynamic traffic assignment method,” *Traffic Eng.*, vol. 26, pp. 17–28, 1991 (in Japanese).

76. T. Akamatsu, M. Yukio, and T. Eikou, “Semi-dynamic Traffic Assignment Models with Queue Evolution and Elastic OD Demand,” *Infrastruct. Plan. Rev.*, vol. 15, no. 4, pp. 535–545, 1998 (in Japanese).

77. M. C. J. Bliemer, M. P. H. Raadsen, E. S. Smits, B. Zhou, and M. G. H. Bell, “Quasi-dynamic traffic assignment with residual point queues incorporating a first order node model,” *Transp. Res. Part B Methodol.*, vol. 68, pp. 363–384, 2014.

78. L. Brederode, A. Pel, L. Wismans, E. de Romph, and S. Hoogendoorn, “Static Traffic Assignment with Queuing: Model Properties and Applications,” *Transp. A Transp. Sci.*, vol. 15, no. 2, pp. 179–214, 2019.

79. P. T. T. Ha, S. Nakayama, and J. Takayama, “A semi-dynamic traffic assignment model and its application,” *Civ. Eng. Stud. Res. Lect. Collect. vol. 45*, 2012.

80. Y. Itagaki, S. Nakayama, and J. Takayama, “A semi-dynamic traffic assignment model with endogenous traffic congestion using the sensitivity analysis,” *J. Japan Soc. Civ. Eng. D3 (Civil Eng. Stud.)*, vol. 70, pp. 569–577, 2014 (in Japanese).

81. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Second Edition*. MIT Press and McGraw-Hill, 2001.

Chapter 3

Sensitivity analysis method for logit-based stochastic user equilibrium traffic assignment

3.1 Overview

Logit-based SUE traffic assignment plays a vital role in the range of traffic assignment models. However, repeated logit-based SUE traffic assignment for large-scale traffic networks requires a huge computational cost. Reducing calculation costs while ensuring the accuracy of traffic assignment results are principal requirements, here this chapter will present a sensitivity analysis method for SUE traffic assignment with the link-based approach. Our method is derived from the fixed-point problem of link traffic flow in the SUE traffic assignment problem. Besides, the properties of the derivative of the implicit function and the chain rule of the derivative will be used. This approach creates unity in the use of derivative concepts and does not have to distinguish between concepts apparent derivative and true partial derivative as in Ying and Miyagi's paper[1)]. Instead of originating from the optimization problem[2)], the fixed-point approach will make this method generalizable and applicable to all fixed-point problems of logit-based SUE traffic assignment models. Moreover, the calculation process will be different from the calculation process of Ying and Miyagi[1)] with the application of STOCH3 and DFS algorithms to ensure the accuracy of the results of calculating gradient matrices. In this study, the derivative of the link flow variable regarding some parameters has been calculated effectively both in terms of computational cost and accuracy. Then, we verify the effectiveness of the calculation method with an illustrative example and case study in the Kanazawa City road network.

3.2 Notations and assumptions

The sensitivity analysis method for the logit-based SUE traffic assignment problem has been taken into consideration to assess how the equilibrium state changes according to the change of model parameters. Assume that two parameters related to free-flow travel time and travel demand will be changed and the degree of their influence on the equilibrium solution will be considered by the sensitivity analysis method. With other parameters, the identical calculation method could be put into use.

Accordingly, notations and assumptions used in this chapter are as follows:

rs: An origin-destination (OD) pair.

$W = \{12, \dots, rs, \dots, uv\}$: The set of OD pairs.

$K^{rs} = \{k, p, \dots\}$: The set of routes connecting an origin and destination.

Q^{rs}: The travel demand of an OD pair rs.

p_k^{rs}: The probability of choosing the route k of OD pair rs (see **Equation (2.16)**).

f_k^{rs}: The flow on the route k of OD pair rs.

θ: The positive parameter of the logit-based route choice model.

c_k^{rs}: The travel time on the route k of OD pair rs.

$N = \{1,2, \dots, i, j, \dots, m, n\}$: The set of nodes.

$A = \{12, \dots, ij, gh, \dots nm\}$: The set of links.

$A_k^{rs} = \{ij, gh, \dots\}$: The set of links that belong to route k of an OD pair rs.

$\delta_{ij,k}^{rs}$: The link-route incidence variable. If the route k includes link ij then $\delta_{ij,k}^{rs} = 1$, otherwise $\delta_{ij,k}^{rs} = 0$.

t_{ij}: The travel time of link ij. It is assumed that t_{ij} is a continuous and non-decreasing function of x_{ij} and parameter ζ_{ij}. By using a conventional traffic model based on BPR (Bureau of Public Roads) curves, we have:

$$t_{ij} = \left(t_{ij}^0 + \zeta_{ij}\right)\left[1 + \alpha\left(\frac{x_{ij}}{Cap_{ij}}\right)^{\beta}\right]. \quad (3.1)$$

where t_{ij}^0 is the free-flow travel time of link ij; α, β are the BPR function parameters; x_{ij} is the travel flow of link ij and Cap_{ij} is the traffic capacity of link ij.

$\mathbf{x} = (x_{12}, \dots, x_{ij}, \dots, x_{mn})^T$: The vectors of all link flows.

$\mathbf{t} = (t_{12}, \dots, t_{ij}, \dots, t_{mn})^T$: The vectors of all link travel times.

$\boldsymbol{\zeta} = (\zeta_{12}, \dots, \zeta_{ij}, \dots, \zeta_{mn})^T$: The vectors of all free-flow travel time parameters.

$\boldsymbol{\nabla_x t}$ and $\boldsymbol{\nabla_\zeta t}$: The diagonal matrix of partial derivatives of t_{ij} as an explicit function with respect to x_{ij} and ζ_{ij}, respectively. We have:

$$\boldsymbol{\nabla_x t} = \begin{pmatrix} \frac{\partial t_{12}}{\partial x_{12}} & & 0 \\ & \ddots & \\ 0 & & \frac{\partial t_{mn}}{\partial x_{mn}} \end{pmatrix}, \tag{3.2}$$

$$\boldsymbol{\nabla_\zeta t} = \begin{pmatrix} \frac{\partial t_{12}}{\partial \zeta_{12}} & & 0 \\ & \ddots & \\ 0 & & \frac{\partial t_{mn}}{\partial \zeta_{mn}} \end{pmatrix}. \tag{3.3}$$

$\boldsymbol{\nabla_\zeta x}$: The matrix of partial derivatives of x_{ij} with respect to ζ_{ij}

$$\boldsymbol{\nabla_\zeta x} = \begin{pmatrix} \frac{\partial x_{12}}{\partial \zeta_{12}} & \dots & \frac{\partial x_{12}}{\partial \zeta_{ij}} & \dots & \frac{\partial x_{12}}{\partial \zeta_{mn}} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \frac{\partial x_{ij}}{\partial \zeta_{12}} & \dots & \frac{\partial x_{ij}}{\partial \zeta_{ij}} & \vdots & \frac{\partial x_{ij}}{\partial \zeta_{mn}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial x_{mn}}{\partial \zeta_{12}} & \dots & \frac{\partial x_{mn}}{\partial \zeta_{ij}} & \dots & \frac{\partial x_{mn}}{\partial \zeta_{mn}} \end{pmatrix}. \tag{3.4}$$

p_{ij}^{rs}: The probability of choosing the link ij of OD pair rs

$$p_{ij}^{rs} = \sum_{k \epsilon K^{rs}} p_k^{rs} \delta_{ij,k}^{rs} = \frac{\sum_{k \epsilon K^{rs}} \delta_{ij,k}^{rs} \exp(-\theta c_k^{rs})}{\sum_{k \epsilon K^{rs}} \exp(-\theta c_k^{rs})}. \tag{3.5}$$

x_{ij}^{rs}: The number of users from node r to node s who choose some routes containing link ij.

$$x_{ij}^{rs} = \sum_{k\epsilon K^{rs}} f_k^{rs}\delta_{ij,k}^{rs} = \sum_{k\epsilon K^{rs}} Q^{rs}p_k^{rs}\delta_{ij,k}^{rs} = Q^{rs}\frac{\sum_{k\epsilon K^{rs}} \delta_{ij,k}^{rs}\exp(-\theta c_k^{rs})}{\sum_{k\epsilon K^{rs}} \exp(-\theta c_k^{rs})}. \quad (3.6)$$

$x_{ij\to gh}^{rs}$: The number of users from node r to node s who choose some routes which contain both links ij and gh in such a way that link ij is used before link gh.

$x_{ij,gh}^{rs}$: The number of users from node r to node s who choose some routes which contain both links ij and gh ($x_{ij,gh}^{rs}(= x_{gh,ij}^{rs})$). We have:

$$x_{ij,gh}^{rs} = \sum_{k\epsilon K^{rs}} f_k^{rs}\delta_{ij,k}^{rs}\delta_{gh,k}^{rs} = \sum_{k\epsilon K^{rs}} Q^{rs}p_k^{rs}\delta_{ij,k}^{rs}\,\delta_{gh,k}^{rs} = Q^{rs}\frac{\sum_{k\epsilon K^{rs}} \delta_{ij,k}^{rs}\delta_{gh,k}^{rs}\exp(-\theta c_k^{rs})}{\sum_{k\epsilon K^{rs}} \exp(-\theta c_k^{rs})}, \quad (3.7)$$

or,

$$x_{ij,gh}^{rs} = x_{gh,ij}^{rs} = \begin{cases} x_{ij\to gh}^{rs} + x_{gh\to ij}^{rs} & \text{if } ij \neq gh, \\ x_{ij}^{rs} & \text{otherwise.} \end{cases} \quad (3.8)$$

C_{rn}^{0}: The reference shortest generalized travel time from origin node r to node n, based on the link travel time $\{t_{ij}^{0}\}$.

ξ^{rs}: The parameter of travel demand of OD pair rs.

$Q^{rs}(\xi^{rs})$: The travel demand of an OD pair that depends on a perturbation ξ^{rs}. Assuming that:

$$Q^{rs}(\xi^{rs}) = Q_0^{rs} + \xi^{rs}. \quad (3.9)$$

$\mathbf{Q} = (Q^{12}, \dots, Q^{rs}, \dots, Q^{uv})^T$: The travel demand vector.

$\boldsymbol{\xi} = (\xi^{12}, \dots, \xi^{rs}, \dots, \xi^{uv})^T$: The vector of the parameter of travel demand.

I: The unit matrix.

$\boldsymbol{\nabla_\xi \mathbf{Q}}$: The matrix of partial derivatives of Q^{rs} as an explicit function with respect to ξ^{rs}

$$\boldsymbol{\nabla_\xi \mathbf{Q}} = \begin{pmatrix} \frac{\partial Q^{12}}{\partial \xi^{12}} & \cdots & \frac{\partial Q^{12}}{\partial \xi^{rs}} & \cdots & \frac{\partial Q^{12}}{\partial \xi^{uv}} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \frac{\partial Q^{rs}}{\partial \xi^{12}} & \cdots & \frac{\partial Q^{rs}}{\partial \xi^{rs}} & \cdots & \frac{\partial Q^{rs}}{\partial \xi^{uv}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial Q^{uv}}{\partial \xi^{12}} & \cdots & \frac{\partial Q^{uv}}{\partial \xi^{rs}} & \cdots & \frac{\partial Q^{uv}}{\partial \xi^{uv}} \end{pmatrix}. \quad (3.10)$$

From **Equation (3.9)**, $\boldsymbol{\nabla_\xi \mathbf{Q}} = \mathbf{I}$.

$\boldsymbol{\nabla_\xi \mathbf{x}}$: The matrix of partial derivatives of x_{ij} with respect to ξ^{rs}

$$\boldsymbol{\nabla_\xi \mathbf{x}} = \begin{pmatrix} \frac{\partial x_{12}}{\partial \xi^{12}} & \cdots & \frac{\partial x_{12}}{\partial \xi^{rs}} & \cdots & \frac{\partial x_{12}}{\partial \xi^{uv}} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \frac{\partial x_{ij}}{\partial \xi^{12}} & \cdots & \frac{\partial x_{ij}}{\partial \xi^{rs}} & \cdots & \frac{\partial x_{ij}}{\partial \xi^{uv}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial x_{mn}}{\partial \xi^{12}} & \cdots & \frac{\partial x_{mn}}{\partial \xi^{rs}} & \cdots & \frac{\partial x_{mn}}{\partial \xi^{uv}} \end{pmatrix}. \quad (3.11)$$

h_{ij}^r: STOCH3 parameter of link ij. Leurent[3)] used this parameter to define efficient links from origin node r (see Chapter 2).

N_i^{in} : The set of start nodes of the links that connect to node i.

N_i^{out}: The set of end nodes of the links that are connected from node i.

Ω_{ij}^r: Indicator variable ($\Omega_{ij}^r \coloneqq 1$ if link ij is efficient from origin node r and $\Omega_{ij}^r \coloneqq 0$ otherwise).

a_{ij}: Likelihood of link ij

$$a_{ij} = \exp\left(-\theta t_{ij}\right) \forall ij \in A. \tag{3.12}$$

wl_{ij}^{rs}: Link weight that presents the importance of link ij in contributing to efficient routes from origin node r to destination node s.

$$wl_{ij}^{rs} = \begin{cases} a_{ij} & if\ i = r\ and\ \Omega_{ij}^{r} = 1 \\ a_{ij} \sum_{m \epsilon N_i^{in}} wl_{mi}^{rs} & if\ i \neq r\ and\ \Omega_{ij}^{r} = 1 \\ 0 & otherwise \end{cases} \tag{3.13}$$

xl_{ij}^{rs}: The flow of link ij from origin node r to destination node s.

$$xl_{ij}^{rs} = \begin{cases} Q^{rs} \dfrac{wl_{ij}^{rs}}{\sum_{m \epsilon N_j^{in}} wl_{mj}^{rs}} & if\ j = s \\ \left(\sum_{n \epsilon N_j^{out}} xl_{jn}^{rs}\right) \dfrac{wl_{ij}^{rs}}{\sum_{m \epsilon N_j^{in}} wl_{mj}^{rs}} & if\ j \neq s \end{cases} \tag{3.14}$$

3.3 Depth-first search algorithm for solving the logit-based stochastic user equilibrium problem

As mentioned in Chapter 2, the logit-based SUE problem could be resolved by using the well-known Dial-based algorithm[4] in which the STOCH3 algorithm with stable route identification performed the more efficiency. This algorithm has a few disadvantages. Firstly, it is possible to skip some important routes. This disadvantage is also a common weakness of Dial-based algorithm shown by Akamatsu[5]. Secondly, there is the possibility of an inconsistency between the definition of efficient links[6]. The set of efficient links used free-flow travel time pattern may be different from these used link trave time in the final SUE solution. Thirdly, it is used to solve the MNL model assuming independence among alternatives. The limitations of the MNL model are shown in Chapter 2. Although there are weaknesses, it cannot be denied the ability to apply this algorithm with the advantage of a link-based approach avoiding route enumeration and mathematical convenience.

Based on Dial's algorithm, Leurent[3] proposed the STOCH3-efficient route definition and performed the effectiveness of the STOCH3 logit model using the MSA. STOCH3 algorithm performing for each origin node r (see Chapter 2) exactly represented the logit-based model in route choice[3]. It is also can run for each OD pair rs (see **Appendix B**) and the convergence of the algorithm is unchanged because the logit-based model is still shown. In the process of applying the STOCH3 algorithm, we tried to list all implicit efficient routes to compare the STOCH3 algorithm with the route-based approach. The DFS algorithm[7] was utilized to list all implicit efficient routes. DFS is a well-known algorithm for traversing tree data structures. The DFS algorithm could be used to search "deeper" in the graph whenever possible. In finding the implicit STOCH3 efficient route, the idea of the algorithm can be presented as follows: First, the nodes adjacent to the original node r will, of course, come from r. For each node n adjacent to r, of course, the nodes adjacent to n also come from r, etc. From that, we develop the DFS algorithm with a recursive technique for displaying entire routes in implicit STOCH3 route-set by Fortran.90 programming (see **Appendix A.1**). When the traffic network is represented as adjacent links instead of matrices and lists types, the algorithm starts at the origin node, explores along with each link, and ends at

the destination node before backtracking. The algorithm is handled with a recursive technique in which each subroutine for each node will visit the adjacent links of that node, continue recursively for the end node of the adjacent links and the stopping condition is that destination node s is found.

For example, if the following network consists of efficient links in **Figure 3.1** and network is saved as adjacent links instead of matrices shown in **Table 3.1**, the order of links visited by the DFS algorithm is *r*1,12,2*s*,25,56,6*s*,14,45,56,6*s*.

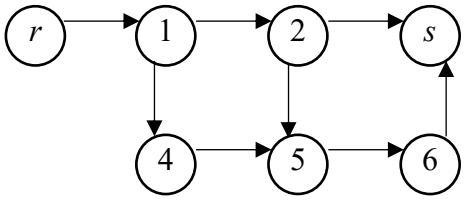


Figure 3.1 A virtual network with efficient links

Table 3.1 Adjacent links of each link in the virtual network

No.	Link	Adjacent links
1	*r*1	{12,14}
2	12	{2*s*,25}
3	14	{45}
4	2*s*	{∅}
5	25	{56}
6	45	{56}
7	56	{6*s*}
8	6*s*	{∅}

The next will show how to find all the efficient routes of the virtual network. The DFS algorithm starts from origin node *r* and visits adjacent link *r*1 of node *r*, adjacent link 12 of node 1, and adjacent link 2*s* of node 2 to arrive *s*. We can use an array, $mark_{ij}$, to save this trail. The number of elements in the $mark_{ij}$ can be set by the number of links on the network. Thus, at the first attempting at the destination s, the DFS algorithm queries the following links: *r*1, 12, 2*s* that form

an efficient route. Because the destination node is reached, the subroutine of node *s* will stop and the DFS algorithm will backtrack link 2*s*, this backtracked link will be removed from mark array. The current node is node 2, because adjacent link 25 is still in the list of links that are waiting for traversing. The DFS algorithm will continue running by traversing link 25, adjacent link 56 of node 5, and adjacent link 6*s* of node 6 to arrive *s*. Hence, the second efficient route is illustrated including links *r*1,12, 25, 56, 6*s*. Because the destination node is reached, the subroutine of node *s* will stop. The DFS algorithm will backtrack link 6*s*, this backtracked link will be dropped from mark array. The current node is node 6. Because there are no adjacent links of node 6 that are waiting for traversing, the DFS algorithm will come back to node 5 from node 6 and backtrack link 56 (this backtracked link will be removed from mark array). Similarly, the DFS algorithm will backtrack links 25, 12, and continue running by traversing link 14. The visiting process is similar, and the final efficient route including links *r*1,14,45,56,6*s* is also found. Because there are no links that need to be visited, the DFS algorithm will stop. This process is illustrated as the following table:

Table 3.2 Detail of the DFS algorithm for the virtual network

No.	Current node	Adjacent links	Link is traversed	Links are waiting for traversing	Next node	Mark array
1	*r*	{*r*1}	*r*1	{∅}	1	{*r*1}
2	1	{12,14}	12	{14}	2	{*r*1,12}
3	2	{2*s*,25}	2*s*	{14, 25}	*s*	**{*r*1,12,2*s*}**
4	2		25	{14}	5	{*r*1,12, 25}
5	5	{56}	56	{14}	6	{*r*1,12, 25, 56}
6	6	{6*s*}	6*s*	{14}	*s*	**{*r*1,12, 25, 56, 6*s*}**
7	1		14	{∅}	4	{*r*1,14}
8	4	{45}	45	{∅}	5	{*r*1,14,45}
9	5	{56}	56	{∅}	6	{*r*1,14,45,56}
10	6	{6*s*}	6*s*	{∅}	*s*	**{*r*1,14,45,56,6*s*}**

If we consider a real traffic network with $|K^{rs}|$ is the total number of implicit STOCH3-efficient-routes, the DFS algorithm operating the recursive technique is capable of representing all $|K^{rs}|$ implicit routes by $|K^{rs}|$ time arriving destination *s*. Although this algorithm takes into account $|K^{rs}|$ time reaching each destination, it is possible to exploit the information of previously accessed links to save the calculation time. If in the set of routes, there are many overlapping routes in the first links, the use of the DFS algorithm will save calculation time. For example, in the second arriving *s*, the algorithm does not have to go back to the origin *r* and it only needs to visit three links 25, 56, 6*s*.

STOCH3 algorithm with the link-based approach running for each origin node r remarkably effective in calculating the MNL model. However, in some other logit models such as CNL, *q*-generalized logit, the use of the STOCH3 algorithm is impossible. We will think about using the DFS algorithm for the extensions of the logit model in later chapters. STOCH3 algorithm based on Dial's algorithm avoids cyclic routes in implicit route-set, but this can exclude some routes used by many users from the implicit route-set [5)]. For example, **Figure 3.2** depicts a small network with the free-flow travel time of each link.

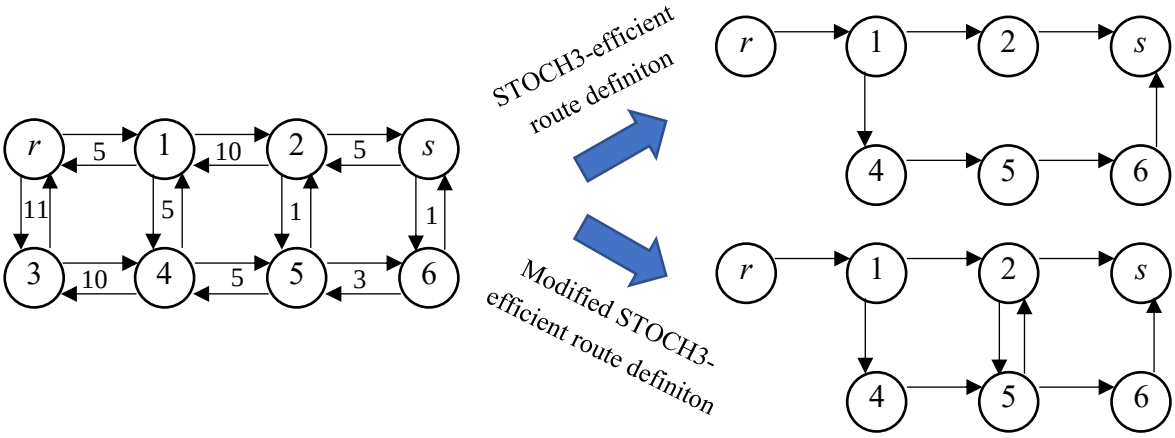


Figure 3.2 Small example shown the weakness of STOCH3 algorithm

If the STOCH3 algorithm is used, the route-set will include only one route. Because the shortest route from node r to node 2 equals to the shortest route from node r to node 5 ($C_{r2}^0 = C_{r5}^0 = 15$), the links 25, 52 are inefficient links. In reality, many users will use these links. If the concept of the efficient routes is modified to mention more about the number of routes in which efficient route does not include the same node more than once and consists of links have their initial node no

further to the origin than their final node ($C_{rj}^0 \geq C_{ri}^0$), the STOCH3 algorithm will not be able to handle this case. In this case, the DFS algorithm is still capable of searching all these efficient routes. Because each link will have a unique start node and end node, the DFS algorithm can search all these efficient routes and avoid infinite loops by using an array to mark the start nodes of the links that have been visited and only searching for links that contain the end nodes have not been visited yet. **Table 3.3** shows this procedure, the coding of this will be detailed in **Appendix A.2**. This algorithm can be used to find all simple routes (the routes do not include the same node more than once) for each OD pair. However, it is a very time-consuming task and not suitable for application purposes. The main purpose of considering the DFS algorithm is to serve the calculation of the sensitivity analysis formulas and compare it with the STOCH3 algorithm. So, its applications into the concept of a modified STOCH3-efficient route and the simple route will not be focused. Here, this algorithm is only considered the applicability with the STOCH3-efficient route definition.

Table 3.3 Detail of the DFS algorithm for searching the modified STOCH3-efficient routes in the small network

No.	Current node	Qualified adjacent links	Link is traversed	Links are waiting for traversing	Next node	Mark array
1	*r*	{*r*1}	*r*1	{∅}	1	{*r*1}
2	1	{12,14}	12	{14}	2	{*r*1,12}
3	2	{2*s*,25}	2*s*	{14, 25}	*s*	**{*r*1,12,2*s*}**
4	2		25	{14}	5	{*r*1,12, 25}
5	5	{56}	56	{14}	6	{*r*1,12, 25, 56}
6	6	{6*s*}	6*s*	{14}	*s*	**{*r*1,12, 25, 56, 6*s*}**
7	1		14	{∅}	4	{*r*1,14}
8	4	{45}	45	{∅}	5	{*r*1,14,45}
9	5	{52,56}	52	{56}	2	{*r*1,14,45,52}
10	2	{2*s*}	2*s*	{56}	*s*	**{*r*1,14,45,52,2*s*}**
11	5		56	{∅}	6	{*r*1,14,56}
12	6	{6*s*}	6*s*	{∅}	*s*	**{*r*1,14,45,56,6*s*}**

Consequently, we will propose a new algorithm to present stochastic network loadings based on the DFS algorithm with the STOCH3-efficient route definition. Instead of relying on the STOCH3 loading procedure (see Chapter 2), we will adopt the DFS algorithm. We will call it DFS loading procedure[8]. In calculations, we only use link-based and node-based variables by using the information each time the algorithm reaches the destination (information about 1 efficient route). The goal of this calculation is to use the DFS algorithm to calculate the numerator and denominator of **Equation (3.5)**. In this case, we use the STOCH3-efficient route definition and calculating the link traffic flow for each OD pair rs. Indicator variable Ω_{ij}^{r} will be used to represent the efficiency of link ij (see Chapter 2). Denoting num_{ij}^{rs} and dnm^{rs} are the numerator and the denominator of **Equation (3.5)**, respectively. We also use $mark_{ij}$ array and use another node-based variable, vn_n^{rs}, for the calculation reason. The process of calculating num_{ij}^{rs} and dnm^{rs} as follows:

For each OD pair, set all num_{ij}^{rs}, $mark_{ij}$ and vn_n^{rs} variables to 0. Set vn_r^{rs}:=1 and $dnm^{rs} = 0$. Performing the DFS algorithm from origin node r to destination node s with recursive technique. After each link with the nonzero value of Ω_{ij}^{r} is visited, we save link ij to the mark array and update $vn_j^{rs} = a_{ij} vn_i^{rs}$. When reaching destination node s, we add vn_s^{rs} to num_{ij}^{rs} ($num_{ij}^{rs} := num_{ij}^{rs} + wn_s^{rs}$) if link ij is included in the mark array and add vn_s^{rs} to dnm^{rs}($dnm^{rs} := dnm^{rs} + wn_s^{rs}$). We have:

$$vn_s^{rs} = \prod_{ij \in A_k^{rs}} a_{ij} = \prod_{ij \in A_k^{rs}} \exp(-\theta\, t_{ij}) = \exp(-\theta c_k^{rs}). \tag{3.15}$$

With the recursive technique, we can reach the destination node s by $|K^{rs}|$ times. dnm^{rs} and num_{ij}^{rs} variables perform the denominator and numerator of **Equation (3.5)** follows:

$$dnm^{rs} = \sum_{k=1}^{|K^{rs}|} vn_s^{rs(k)} = \sum_{k=1}^{|K^{rs}|} \exp(-\theta c_k^{rs}), \tag{3.16}$$

$$num_{ij}^{rs} = \sum_{k=1}^{|K^{rs}|} \delta_{ij,k}^{rs} vn_s^{rs(k)} = \sum_{k=1}^{|K^{rs}|} \delta_{ij,k}^{rs} \exp(-\theta c_k^{rs}). \tag{3.17}$$

Thus, we can compute **Equation (3.5)** through dnm^{rs} and num_{ij}^{rs} variables. Next, link traffic flow of each OD pair in **Equation (3.6)** is also calculated without using route-based $\delta_{ij,k}^{rs}$, c_k^{rs} variables. To show the proposed algorithm that can calculate the link traffic flow without route-based variables, the application to the virtual road network shown in **Figure 3.1** is conducted. Assuming the free-flow travel time of the links is shown in the following table.

Table 3.4 Free-flow travel time of links in the virtual network

No.	Link	Start node	End node	Free-flow travel time (min)
1	*r*1	*r*	1	10
2	12	1	2	10
3	14	1	4	5
4	2*s*	2	*s*	10
5	25	2	5	5
6	45	4	5	8
7	56	5	6	2
8	6*s*	6	*s*	5

If the fixed travel demand and the logit parameter are assumed as $Q_0^{rs} = 100$ and $\theta = 0.1$, applying the route-based approach bring the results of efficient route flows as follows:

Table 3.5 The results of route flows in the virtual network

No.	Detailed links	Route travel time (min)	Probability of choosing route	Route flow (pcu)
1	*r*1-12-2*s*	10+10+10=30	0.355	35.477
2	*r*1-12-25-56-6*s*	10+10+5+2+5=32	0.290	29.046
3	*r*1-14-45-56-6*s*	10+5+8+2+5=30	0.355	35.477

From the link-route incidence matrix, the efficient link traffic flows are calculated as the following table:

Table 3.6 The results of link flows in the virtual network

No.	**Link**	**Link flow (pcu)**
1	*r*1	100.000
2	12	64.523
3	14	35.477
4	2*s*	35.477
5	25	29.046
6	45	35.477
7	56	64.523
8	6*s*	64.523

If we do not apply the route-based approach, the DFS loading procedure will be depicted through the following table:

Table 3.7 The DFS loading procedure in the application to the virtual network

No.	Link is traversed	Mark array	vn_i^{rs}	vn_j^{rs}	Note
1	*r*1	{*r*1}	1.000	0.368	
2	12	{*r*1,12}	0.368	0.135	
3	2*s*	**{*r*1,12,2*s*}**	0.135	0.050	dnm^{rs} =0.050; num_{r1}^{rs} =0.050; num_{12}^{rs} =0.050; num_{2s}^{rs} =0.050
4	25	{*r*1,12, 25}	0.135	0.082	
5	56	{*r*1,12, 25, 56}	0.082	0.067	
6	6*s*	**{*r*1,12, 25, 56, 6*s*}**	0.067	0.041	dnm^{rs} =0.050 + 0.041= 0.091; num_{r1}^{rs} =0.091; num_{12}^{rs} =0.091; num_{2s}^{rs} =0.050; num_{25}^{rs} =0.041; num_{56}^{rs} =0.041; num_{6s}^{rs} =0.041
7	14	{*r*1,14}	0.368	0.223	
8	45	{*r*1,14,45}	0.223	0.100	
9	56	{*r*1,14,45,56}	0.100	0.082	
10	6*s*	**{*r*1,14,45,56,6*s*}**	0.082	0.050	dnm^{rs} =0.091+ 0.050= 0.140; num_{r1}^{rs} =0.140; num_{12}^{rs} =0.091; num_{2s}^{rs} =0.050; num_{25}^{rs} =0.041; num_{56}^{rs} =0.091; num_{6s}^{rs} =0.091; num_{14}^{rs} =0.050; num_{45}^{rs} =0.050

After the DFS algorithm stops, the probability of choosing the links and link flows are calculated by the following table:

Table 3.8 The results of link flows brought from the DFS loading procedure

No.	Link	num_{ij}^{rs}	Probability of choosing link	Link flow (pcu)
1	$r1$	0.140	1.000	100.000
2	12	0.091	0.645	64.523
3	14	0.050	0.355	35.477
4	2s	0.050	0.355	35.477
5	25	0.041	0.290	29.046
6	45	0.050	0.355	35.477
7	56	0.091	0.645	64.523
8	6s	0.091	0.645	64.523

The results of link traffic flow brought from the DFS loading procedure are equal to these brought from the route-based approach. Either the use of the STOCH3 algorithm also for each origin or for each OD pair rs (see **Appendix B**) also brings the same link flows results. In the proposed algorithm, we do not need to sort $\{C_{rn}^0\}$ in order of increasing access time from r and all variables are link-based and node-based variables.

In the proposed new algorithm, instead of using well-known "forward" and "backward" techniques with the requirement of sorting $\{C_{rn}^0\}$ in order of increasing access time from r (see Chapter 2), we only exploit the DFS algorithm with link-based and node-based variables. Comparing to the STOCH3 algorithm, on one hand, the new algorithm could keep the advantage of preventing the route variables that reduces computation storage, on the other hand, the new algorithm is easy to practice and will also reduce the computation time in some cases of testing. MSA will also be used to achieve the SUE solution with this algorithm.

The DFS loading procedure for solving SUE traffic assignment with fixed travel demand problem using the MSA and STOCH3-efficient route definition is showed as follows:

Step 0: Preliminaries

(a) Set iteration counter $l := 0$ and maximum value of the change (σ) in link flow from the previous iteration $l-1$ to the current one l.

(b) Based on the free-flow travel time $\{t_{ij}^0\}$, for each origin r, calculate the minimum travel time to all other nodes.

(c) For each link ij, set $a_{ij} := \exp(-\theta\, t_{ij}^0)$ and $\Omega_{ij}^r := 1$ if $(1 + h_{ij}^r)(\mathrm{C}_{rj}^0 - \mathrm{C}_{ri}^0) \geq t_{ij}^0$, otherwise $\Omega_{ij}^r := 0$ and set $x_{ij}^{(l)} := 0$, $t_{ij} = t_{ij}^0$.

(d) For each OD pair rs: Set all num_{ij}^{rs}, $mark_{ij}$ and vn_n^{rs} to 0. Set vn^{rs}:=1 and $dnm^{rs} = 0$. Performing the DFS algorithm from origin node r to destination node s with recursive technique. After each link with the nonzero value of Ω_{ij}^r is visited, link ij is stored in mark array and we update $vn_j^{rs} = a_{ij} vn_i^{rs}$. When arriving at the destination node s, vn_s^{rs} is added to num_{ij}^{rs} ($num_{ij}^{rs} := num_{ij}^{rs} + wn_s^{rs}$) if link ij is stored in $mark_{ij}$ and vn_s^{rs} is also added to dnm^{rs}($dnm^{rs} := dnm^{rs} + vn_s^{rs}$).

After the DFS algorithm stops, the contribution to total link-flows is calculated by:

$$x_{ij}^{(l)} := x_{ij}^{(l)} + Q_0^{rs} \frac{num_{ij}^{rs}}{dnm^{rs}}. \tag{3.18}$$

When all OD pairs are counted, link traffic flows are fully calculated.

Step 1: Link travel time and link likelihood update

(a) Set $l := l + 1$.

(b) Set $t_{ij}^{(l)} := t_{ij}(x_{ij}^{(l)})$, $x_{ij}^{(l+1)} := 0$, and $a_{ij} := \exp(-\theta\, t_{ij}^{(l)})$.

Step 2: Direction finding

Set all num_{ij}^{rs}, $mark_{ij}$ and vn_n^{rs} to 0. Set vn_r^{rs}:=1 and $dnm^{rs} = 0$. Based on link travel time $\{t_{ij}^{(l)}\}$ and a_{ij}, adopting the DFS algorithm with the same procedure as substep (d) of step 0. When the DFS algorithm stops, the auxiliary link flow is yielded as the following equation:

$$yx_{ij}^{(l)} := yx_{ij}^{(l)} + Q_0^{rs}\frac{num_{ij}^{rs}}{dnm^{rs}}. \quad (3.19)$$

Step 3: Link flow update

(a) Choose a sequence of real numbers such that $(0 < \lambda^{(l)} < 1)$.

(b) Let $rg_{ij}^{(l)} = yx_{ij}^{(l)} - x_{ij}^{(l)}$

(c) Set $x_{ij}^{(l+1)} = x_{ij}^{(l)} + \lambda^{(l)} rg_{ij}^{(l)}$

Step 4: Stopping the test

If $max_{ij}\{|rg_{ij}^{(l)}|\} \leq \sigma$, stop. The solution is $\{x_{ij}^{(l)}\}$. Otherwise, go to step 2.

3.4 Sensitivity analysis method for stochastic user equilibrium traffic assignment based on the Depth-first search algorithm and STOCH3 algorithm

A sensitivity analysis method for logit-based SUE traffic assignment was proposed by Ying and Miyagi[1]. Based on the minimization formulation (Daganzo[2]), Ying and Miyagi[1] used the different notations of true partial derivative and "apparent" derivative of link travel time and link flow concerning parameters and calculated them. The computation procedure with Dial's traffic assignment and MSA algorithm was also presented. Nevertheless, the weakness of Dial's algorithm with the instability of the efficient route definition was shown by Leurent[2]. The set of efficient routes are different depending on the number of iterations and the convergence of the iterative process cannot be assured. In fact, when applying the Dial's algorithm to the Kanazawa road network, the algorithm did not converge to the equilibrium solution. There is no problem applying the Dial's algorithm and the sensitivity analysis method based on Dial's algorithm to the small road network. However, it is difficult to keep the stability of this approach when applying it to the complex network. Instead of applying the Dial's algorithm, we have tried to apply the method of Ying and Miyagi[1] with the STOCH3 algorithm. Because the STOCH3 algorithm is a Dial-based algorithm, it can be applied to the method of Ying and Miyagi[1]. The results are still good with the application to a small network. But, the accuracy of the sensitivity analysis results was not high in the application to Kanazawa road network and it was also impossible to apply this method in some cases. The reasons for these will be analyzed below.

If the method of Ying and Miyagi[1] is used, the "true" gradient matrix $\boldsymbol{\nabla_{\zeta}\mathbf{x}}$ and $\boldsymbol{\nabla_{\xi}\mathbf{x}}$ is given by:

$$\boldsymbol{\nabla_{\zeta}\mathbf{x}} = (\mathbf{x})_{\boldsymbol{\zeta}} + (\mathbf{x})_{\mathbf{t}}\left[-(\boldsymbol{\nabla_{\mathbf{t}}^{2}}Z)^{-1}(\mathbf{x})_{\boldsymbol{\zeta}}\right], \tag{3.20}$$

$$\boldsymbol{\nabla_{\xi}\mathbf{x}} = (\mathbf{x})_{\mathbf{t}}\left[-(\boldsymbol{\nabla_{\mathbf{t}}^{2}}Z)^{-1}(\boldsymbol{\nabla_{\mathbf{t}}}Z)_{\mathbf{Q}}\boldsymbol{\nabla_{\xi}}\mathbf{Q}\right], \tag{3.21}$$

where, $(\mathbf{x})_{\boldsymbol{\zeta}}$ and $(\mathbf{x})_{\mathbf{t}}$ are the diagonal matrix of "apparent" derivative of x_{ij} with respect to ζ_{ij} and t_{ij}, respectively. It is confusing to distinguish between the two concepts "true" derivative and "apparent" derivative. According to Ying and Miyagi[1], "apparent" derivative of x_{ij} with respect to ζ_{ij} and t_{ij} (denoted by

$(x_{ij})_{\zeta_{ij}}$ and $(x_{ij})_{t_{ij}}$) could be computed by using the explicit function of link travel time such as BPR function. The elements of $(\mathbf{x})_{\boldsymbol{\zeta}}$ and $(\mathbf{x})_{\mathbf{t}}$ are calculated as follows:

$$(\mathbf{x})_{\boldsymbol{\zeta}} = \begin{pmatrix} (x_{12})_{\zeta_{12}} & & 0 \\ & \ddots & \\ 0 & & (x_{mn})_{\zeta_{mn}} \end{pmatrix}, \tag{3.22}$$

$$(\mathbf{x})_{\mathbf{t}} = \begin{pmatrix} (x_{12})_{t_{12}} & & 0 \\ & \ddots & \\ 0 & & (x_{mn})_{t_{mn}} \end{pmatrix}, \tag{3.23}$$

$$\begin{aligned} (x_{ij})_{t_{ij}}\Big|_{\zeta_{ij}=0} &= \frac{1}{(t_{ij})_{x_{ij}}}\Bigg|_{\zeta_{ij}=0} = \frac{1}{\left((t_{ij}^0+\zeta_{ij})\left(1+\alpha\left(\frac{x_{ij}}{cap_{ij}}\right)^{\beta}\right)\right)_{x_{ij}}}\Bigg|_{\zeta_{ij}=0} \\ &= \frac{(cap_{ij})^{\beta}}{(t_{ij}^0+\zeta_{ij})\alpha\beta {x_{ij}}^{\beta-1}}\Bigg|_{\zeta_{ij}=0} = \frac{(cap_{ij})^{\beta}}{t_{ij}^0\alpha\beta {x_{ij}}^{\beta-1}}, \end{aligned} \tag{3.24}$$

$$\begin{aligned} (x_{ij})_{\zeta_{ij}}\Big|_{\zeta_{ij}=0} &= -\frac{(t_{ij})_{\zeta_{ij}}}{(t_{ij})_{x_{ij}}}\Bigg|_{\zeta_{ij}=0} = -\frac{\left((t_{ij}^0+\zeta_{ij})\left(1+\alpha\left(\frac{x_{ij}}{cap_{ij}}\right)^{\beta}\right)\right)_{\zeta_{ij}}}{\left((t_{ij}^0+\zeta_{ij})\left(1+\alpha\left(\frac{x_{ij}}{cap_{ij}}\right)^{\beta}\right)\right)_{x_{ij}}}\Bigg|_{\zeta_{ij}=0} \\ &= -\frac{\left(1+\alpha\left(\frac{x_{ij}}{cap_{ij}}\right)^{\beta}\right)(cap_{ij})^{\beta}}{(t_{ij}^0+\zeta_{ij})\alpha\beta {x_{ij}}^{\beta-1}}\Bigg|_{\zeta_{ij}=0} = -\frac{(cap_{ij})^{\beta}+\alpha {x_{ij}}^{\beta}}{t_{ij}^0\alpha\beta {x_{ij}}^{\beta-1}}; \end{aligned} \tag{3.25}$$

$(\boldsymbol{\nabla}_{\mathbf{t}} Z)_{\mathbf{Q}}$ is the matrix of mixed second-order partial derivatives of the minimization function Z of the SUE problem with respect to link travel times and travel demands. According to Ying and Miyagi[1)], $(\boldsymbol{\nabla}_{\mathbf{t}} Z)_{\mathbf{Q}}$ is shown as follows:

$$(\boldsymbol{\nabla}_{\mathbf{t}} Z)_{\mathbf{Q}} = -\begin{pmatrix} \frac{x_{12}^{12}}{Q_0^{12}} & \cdots & \frac{x_{ij}^{12}}{Q_0^{12}} & \cdots & \frac{x_{mn}^{12}}{Q_0^{12}} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \frac{x_{12}^{rs}}{Q_0^{rs}} & \cdots & \frac{x_{ij}^{rs}}{Q_0^{rs}} & \cdots & \frac{x_{mn}^{rs}}{Q_0^{rs}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{x_{12}^{uv}}{Q_0^{uv}} & \cdots & \frac{x_{ij}^{uv}}{Q_0^{uv}} & \cdots & \frac{x_{mn}^{uv}}{Q_0^{uv}} \end{pmatrix}; \tag{3.26}$$

$^{-1}$ is the inverse of the matrix and $\boldsymbol{\nabla}_{\mathbf{t}}^2 Z$ is the matrix of Hessian of the minimization function Z of the SUE problem with respect to link travel times (see Ying and Miyagi[1)] for details). According to Ying and Miyagi[1)], $\boldsymbol{\nabla}_{\mathbf{t}}^2 Z$ is calculated as follows:

$$\boldsymbol{\nabla}_{\mathbf{t}}^2 Z = (\mathbf{x})_{\mathbf{t}} - \mathbf{H} \tag{3.27}$$

where **H** is $|A| \times |A|$ matrix represented as follows:

$$\mathbf{H} = \begin{pmatrix} h_{12,12} & \cdots & h_{12,ij} & \cdots & h_{12,mn} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ h_{ij,12} & \cdots & h_{ij,ij} & \cdots & h_{ij,mn} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ h_{mn,12} & \cdots & h_{mn,ij} & \cdots & h_{mn,mn} \end{pmatrix}$$

each element of matrix **H** is calculated in the general form as follows:

$$h_{ij,gh} = \sum_{rs \in W} Q_0^{rs} \theta \left[\frac{x_{ij,gh}^{rs}}{Q_0^{rs}} + \frac{x_{ij}^{rs}}{Q_0^{rs}} \frac{x_{gh}^{rs}}{Q_0^{rs}} \right]$$

At SUE state, when the travel time of all links are fixed, x_{ij}^{rs} can be calculated by using the STOCH3 once because in the STOCH3 algorithm, x_{ij} is computed for each OD pair and $x_{ij} = \sum_{rs \in W} x_{ij}^{rs}$. After x_{ij}^{rs} is easily computed with the link-based approach, we need to compute $x_{ij,gh}^{rs}$. According to Ying and Miyagi[1)], either $x_{ij \to gh}^{rs}$ or $x_{gh \to ij}^{rs}$ is zero implies that $x_{ij,gh}^{rs} = \max\{x_{ij \to gh}^{rs}, x_{gh \to ij}^{rs}\}$. Ying and Miyagi[1)] computed $x_{ij \to gh}^{rs}$ as follows: Denoting p_{gh}^{js} is the fraction of the number of users from node j to node s who use link gh once. Ying and Miyagi[1)] stated that

$$\frac{x_{vj \to gh}^{us}}{x_{vj}^{rs}} = p_{gh}^{js} = \frac{x_{ij \to gh}^{rs}}{x_{ij}^{rs}} \tag{3.28}$$

for any $us \in W$ and $vj \in A$; hence, both $x_{ij\to gh}^{rs}$ and $x_{vj\to gh}^{us}$ could be computed through p_{gh}^{js} in which p_{gh}^{js} can be computed by assigning $Q_0^{js} = 1$ and running STOCH3 algorithm from j to s for all links gh.

The computation procedure, called algorithm 1, is summarized as follows:

Step 1: Compute the SUE by the MSA, which repeatedly uses the STOCH3 algorithm until the link flows converge. The results of the link flow x_{ij} and the link travel time t_{ij} are attained.

Step 2: Based on the link travel time in ***Step 1***, setting $a_{ij} \coloneqq \exp(-\theta\, t_{ij})$ for each link ij and running STOCH3 algorithm for each OD pair once to calculate x_{ij}^{rs} as follows:

- Each link ij is considered in the order of increasing reference access time from r and wl_{ij}^{rs} by using **Equation (3.13)**.

- For each link ij regarded in the order of decreasing reference access time from r, compute xl_{ij}^{rs} by using **Equation (3.14)**.

After all, x_{ij}^{rs} is equal to xl_{ij}^{rs}.

Step 3: For each node pair js, if there is some $x_{ij}^{rs} \neq 0$, then compute p_{gh}^{js} for all links gh by setting $a_{gh} \coloneqq \exp(-\theta\, t_{gh})$ running STOCH3 algorithm once from j to s with $Q_0^{js} = 1$ as follows:

- For each link gh considered in the order of increasing reference access time from j, wl_{gh}^{js} is calculated

$$wl_{gh}^{js} = \begin{cases} a_{gh} & if\ g = j\ and\ \Omega_{gh}^{j} = 1 \\ a_{gh} \sum_{m \epsilon N_g^{in}} wl_{mg}^{rs} & if\ g \neq j\ and\ \Omega_{gh}^{j} = 1 \\ 0 & otherwise \end{cases}$$

- For each link gh taken in the order of decreasing reference access time from j, compute p_{gh}^{js}

$$p_{gh}^{js} = \begin{cases} \dfrac{wl_{gh}^{js}}{\sum_{m \epsilon N_h^{in}} wl_{mh}^{js}} & if\ h = s \\ \left(\sum_{n \epsilon N_h^{out}} xl_{hn}^{js} \right) \dfrac{wl_{gh}^{js}}{\sum_{m \epsilon N_h^{in}} wl_{mh}^{js}} & if\ h \neq s \end{cases}$$

Step 4: Compute

$$x_{ij \to gh}^{rs} = x_{ij}^{rs} p_{gh}^{js},\ x_{gh \to ij}^{rs} = x_{gh}^{rs} p_{ij}^{hs}$$

and

$$x_{ij,gh}^{rs} = x_{gh,ij}^{rs} = \begin{cases} x_{ij \to gh}^{rs} + x_{gh \to ij}^{rs}, & \text{if } ij \neq gh, \\ x_{ij}^{rs}, & \text{otherwise.} \end{cases}$$

Step 5: Compute $(\boldsymbol{\nabla_t} Z)_{\mathbf{Q}}$ and $\boldsymbol{\nabla_t^2} Z$ from **Equations (3.26) and (3.27)**

Step 6: Compute $\boldsymbol{\nabla_\zeta \mathbf{x}}$ and $\boldsymbol{\nabla_\xi \mathbf{x}}$ from **Equations (3.20) and (3.21)**

From the above formulas and calculations, some problems decrease the accuracy of the proposed sensitivity analysis method when it is applied to a large road network.

Firstly, the mathematical formulations of the proposed sensitivity analysis method confused practice users with the definitions of "true" derivative and "apparent" derivative.

Secondly, the matrix $\boldsymbol{\nabla_t^2} Z$ is not always calculable. If there is a link have no flow (x_{ij} =0), $\left(x_{ij}\right)_{t_{ij}}$ is infinite from using **Equation (3.24)**. Therefore, $\boldsymbol{\nabla_t^2} Z$ can not be computed in this case.

Thirdly, the problem lies in ***Step 3*** and ***Step 4*** with the use of p_{gh}^{js} variable. Ignoring the need for memory to save this variable, formula and calculational process do not guarantee consistency. Since the definition of the effective routes in Dial-based algorithms depends on the origin node r of the OD pair rs (Dial[4]),

Leuren[3]), the set of effective routes is different depending on the origin node r and p_{gh}^{js} will also be different in each OD pair rs. Therefore, it is impossible to use **Equation (3.28**) to express p_{gh}^{js} and use the Dial's algorithm to calculate this variable. Accordingly, p_{gh}^{js} cannot be used to calculate $x_{ij\to gh}^{rs}$ in ***Step 4***.

For example, the following small virtual network consisting of 6 nodes, 8 links and two OD pair from node 1 to node 6 (OD1) and node 2 to node 6 (OD2) is used to show the problem of utilizing p_{gh}^{js} variable. The link capacity and free-flow travel time are written in the form of [free-flow travel time (min), capacity (pcu)] in **Figure 3.3**. The parameters of the BPR-type performance function are $\alpha = 1.0$ and $\beta = 2.0$. The parameter in the logit model, θ, is 1.0. The OD demands are $Q_0^{16} = 30$ and $Q_0^{26} = 30$.

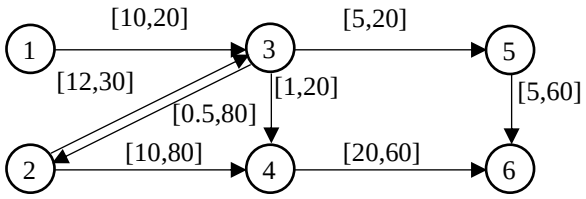


Figure 3.3 A small network with 6 nodes and 8 links

After using the STOCH3 algorithm with the MSA method, the results of link travel time and link travel flow at the SUE state are shown in **Table 3.9**.

Table 3.9 The results of SUE state of the small network

No.	Link	Link travel time (min)	Link travel flow (pcu)		
			OD1	OD2	Total
1.	13	32.500	30.00	0	30.00
2.	23	12.608	0	6.75	6.75
3.	24	10.845	0.00	23.25	23.25
4.	32	0.500	0.00	0	0.00
5.	34	1.064	5.08	0	5.08
6.	35	17.539	24.92	6.75	31.67
7.	46	24.458	5.08	23.25	28.33
8.	56	6.393	24.92	6.75	31.67

Efficient routes for each OD pair rs are shown in **Table 3.10**

Table 3.10 Efficient routes for each OD pair rs of the small network

OD pair	Efficient routes
OD1 (1→6)	(1: 1→3→5→6; 2: 1→3→4→6; 3: 1→3→2→4→6)
OD2 (2→6)	(1: 2→3→5→6; 2: 2→4→6)

At SUE state, we compute p_{gh}^{js} for all links by running STOCH3 algorithm for assigning a virtual unit OD demand ($Q_0^{js} = 1$) on all links gh and compute $x_{ij\to gh}^{rs} = x_{ij}^{rs} p_{gh}^{js}$. The results of p_{gh}^{js} and $x_{ij\to gh}^{rs}$ are shown in **Table 3.11** and **Table 3.12**.

Table 3.11 The results of p_{gh}^{js} of the small network

js	gh	p_{gh}^{js}
36	24	0.00001
	32	0.00001
	34	0.16928
	35	0.83071
	46	0.16929
	56	0.83071
46	46	1.00000
26	23	0.22501
	24	0.77499
	35	0.22501
	46	0.77499
	56	0.22501
56	56	1.00000

Table 3.12 The results of $x_{ij\to gh}^{rs}$ of the small network

rs	ij	gh	$x_{ij\to gh}^{rs}$
16	13	24	0.00017
	13	32	0.00017
	13	34	5.07838
	13	35	24.92144
	13	46	5.07856
	13	56	24.92144
	24	46	0.00017
	32	23	0.00004
	32	24	0.00014
	32	35	0.00004
	32	46	0.00014
	32	56	0.00004
	34	46	5.07838
	35	56	24.92144
26	23	24	0.00004
	23	32	0.00004
	23	34	1.14270
	23	35	5.60763
	23	46	1.14274
	23	56	5.60763
	24	46	23.24963
	35	56	6.75037

If we use p_{gh}^{js} to calculate $x_{ij\to gh}^{rs}$, there will be some unreasonable results such as $x_{23\to 34}^{26} = 1.14274$, $x_{23\to 32}^{26} = x_{32\to 23}^{16} = 0.00004$. These results are unreasonable compared to the equilibrium result in **Table 3.9**. In fact, $x_{23\to 34}^{26} = 0$ and $x_{23\to 32}^{26} = 0$ because links 32, 34 are not efficient links of OD pair 26; and $x_{32\to 23}^{16} = 0$ because link 23 is not an efficient link of OD pair 16 (see **Table 3.10**). Efficient routes are different between OD1 and OD2, but the calculation of p_{gh}^{js} do not consider these differences. This error causes the results of the gradient matrix calculated in ***Steps 5 and 6*** to be affected.

From the above discoveries, we think of another approach to solve the sensitivity analysis for the SUE problem with higher efficiency and wider application than the proposed method. The new approach can solve the above problems.

The starting point of our approach is from the fixed-point problem of the logit-based traffic assignment shown in Chapter 2. The link flow is calculated as:

$$x_{ij} = \sum_{rs\in W} Q^{rs}(\boldsymbol{\xi}) p_{ij}^{rs}(\mathbf{c}(\mathbf{t}(\mathbf{x},\boldsymbol{\zeta}))), \tag{3.29}$$

where p_{ij}^{rs} is the probability of choosing link ij from r to s

$$p_{ij}^{rs} = \frac{\sum_{k\epsilon K^{rs}} \delta_{ij,k}^{rs} \exp(-\theta c_k^{rs}(\mathbf{t}(\mathbf{x},\boldsymbol{\zeta})))}{\sum_{k\epsilon K^{rs}} \exp(-\theta c_k^{rs}(\mathbf{t}(\mathbf{x},\boldsymbol{\zeta})))} \tag{3.30}$$

The right-hand side of **Equation (3.29)** is a function of three variables **ξ, x**, and **ζ**. At static SUE state, the left-hand side equals to the right-hand side. Let b_{ij} denotes the right-hand side of **Equation (3.29)** (b_{ij} is a function of three variables **ξ , x**, and **ζ**), $\mathbf{b} = (b_{12}, \dots, b_{ij}, b_{gh} \dots, b_{mn})^T$ is the vector form of b_{ij} . The following equation, $\mathbf{d} = (d_{12}, \dots, d_{ij}, d_{gh} \dots, d_{mn})^T$ is also defined as a function with three variables **ξ, x**, and **ζ**

$$\mathbf{d}(\boldsymbol{\xi},\mathbf{x},\boldsymbol{\zeta}) = \mathbf{x} - \mathbf{b}\big(\mathbf{Q}(\boldsymbol{\xi}),\mathbf{t}(\mathbf{x},\boldsymbol{\zeta})\big). \tag{3.31}$$

The gap between **x** and **b** should be zero; that is, $\mathbf{d}(\boldsymbol{\xi},\mathbf{x},\boldsymbol{\zeta}) = \mathbf{0}$, under network equilibrium state.

Denoting $\boldsymbol{\nabla_{\xi}\mathbf{b}}$, $\boldsymbol{\nabla_{\mathbf{Q}}\mathbf{b}}$, $\boldsymbol{\nabla_{\mathbf{x}}\mathbf{b}}$, $\boldsymbol{\nabla_{\mathbf{t}}\mathbf{b}}$ and $\boldsymbol{\nabla_{\zeta}\mathbf{b}}$ are the matrices of partial derivatives of b_{ij} with respect to ξ^{rs}, Q^{rs}, x_{ij}, t_{ij} and ζ_{ij} in which:

$$\boldsymbol{\nabla_{\xi}\mathbf{b}} = \begin{pmatrix} \frac{\partial b_{12}}{\partial \xi^{12}} & \cdots & \frac{\partial b_{12}}{\partial \xi^{rs}} & \cdots & \frac{\partial b_{12}}{\partial \xi^{uv}} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \frac{\partial b_{ij}}{\partial \xi^{12}} & \cdots & \frac{\partial b_{ij}}{\partial \xi^{rs}} & \cdots & \frac{\partial b_{ij}}{\partial \xi^{uv}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial b_{mn}}{\partial \xi^{12}} & \cdots & \frac{\partial g_{mn}}{\partial \xi^{rs}} & \cdots & \frac{\partial b_{mn}}{\partial \xi^{uv}} \end{pmatrix} \tag{3.32}$$

$$\boldsymbol{\nabla_{\mathrm{Q}}\mathbf{b}} = \begin{pmatrix} \frac{\partial b_{12}}{\partial Q^{12}} & \cdots & \frac{\partial b_{12}}{\partial Q^{rs}} & \cdots & \frac{\partial b_{12}}{\partial Q^{uv}} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \frac{\partial b_{ij}}{\partial Q^{12}} & \cdots & \frac{\partial b_{ij}}{\partial Q^{rs}} & \cdots & \frac{\partial b_{ij}}{\partial Q^{uv}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial b_{mn}}{\partial Q^{12}} & \cdots & \frac{\partial b_{mn}}{\partial Q^{rs}} & \cdots & \frac{\partial b_{mn}}{\partial Q^{uv}} \end{pmatrix} \tag{3.33}$$

$$\boldsymbol{\nabla_{\mathbf{x}}\mathbf{b}} = \begin{pmatrix} \frac{\partial b_{12}}{\partial x_{12}} & \cdots & \frac{\partial b_{12}}{\partial x_{ij}} & \cdots & \frac{\partial b_{12}}{\partial x_{mn}} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \frac{\partial b_{ij}}{\partial x_{12}} & \cdots & \frac{\partial b_{ij}}{\partial x_{ij}} & \cdots & \frac{\partial b_{ij}}{\partial x_{mn}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial b_{mn}}{\partial x_{12}} & \cdots & \frac{\partial b_{mn}}{\partial x_{ij}} & \cdots & \frac{\partial b_{mn}}{\partial x_{mn}} \end{pmatrix} \tag{3.34}$$

$$\boldsymbol{\nabla_{\mathbf{t}}\mathbf{b}} = \begin{pmatrix} \frac{\partial b_{12}}{\partial t_{12}} & \cdots & \frac{\partial b_{12}}{\partial t_{ij}} & \cdots & \frac{\partial b_{12}}{\partial t_{mn}} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \frac{\partial b_{ij}}{\partial t_{12}} & \cdots & \frac{\partial b_{ij}}{\partial t_{ij}} & \cdots & \frac{\partial b_{ij}}{\partial t_{mn}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial b_{mn}}{\partial t_{12}} & \cdots & \frac{\partial b_{mn}}{\partial t_{ij}} & \cdots & \frac{\partial b_{mn}}{\partial t_{mn}} \end{pmatrix} \tag{3.35}$$

$$\boldsymbol{\nabla_{\zeta}\mathbf{b}} = \begin{pmatrix} \frac{\partial b_{12}}{\partial \zeta_{12}} & \cdots & \frac{\partial b_{12}}{\partial \zeta_{ij}} & \cdots & \frac{\partial b_{12}}{\partial \zeta_{mn}} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \frac{\partial b_{ij}}{\partial \zeta_{12}} & \cdots & \frac{\partial b_{ij}}{\partial \zeta_{ij}} & \cdots & \frac{\partial b_{ij}}{\partial \zeta_{mn}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial b_{mn}}{\partial \zeta_{12}} & \cdots & \frac{\partial b_{mn}}{\partial \zeta_{ij}} & \cdots & \frac{\partial b_{mn}}{\partial \zeta_{mn}} \end{pmatrix} \tag{3.36}$$

Denoting $\boldsymbol{\nabla_{\xi}\mathbf{d}}$, $\boldsymbol{\nabla_{\mathbf{Q}}\mathbf{d}}$, $\boldsymbol{\nabla_{\mathbf{x}}\mathbf{d}}$, $\boldsymbol{\nabla_{\mathbf{t}}\mathbf{d}}$ and $\boldsymbol{\nabla_{\zeta}\mathbf{d}}$ are the matrices of partial derivatives of d_{ij} with respect to ξ^{rs}, Q^{rs}, x_{ij}, t_{ij} and ζ_{ij}. The form of these matrices is also expressed similarly to the partial derivative matrices of function $\mathbf{b}$ mentioned above.

By the chain rule of differentiation, we have $\boldsymbol{\nabla_{\xi}\mathbf{b}} = \boldsymbol{\nabla_{\mathbf{Q}}\mathbf{b}\nabla_{\xi}\mathbf{Q}}$, $\boldsymbol{\nabla_{\mathbf{x}}\mathbf{b}} = \boldsymbol{\nabla_{\mathbf{t}}\mathbf{b}\nabla_{\mathbf{x}}\mathbf{t}}$ and $\boldsymbol{\nabla_{\zeta}\mathbf{b}} = \boldsymbol{\nabla_{\mathbf{t}}\mathbf{b}\nabla_{\zeta}\mathbf{t}}$ in which $\boldsymbol{\nabla_{\xi}\mathbf{Q}} = \mathbf{I}$ and $\boldsymbol{\nabla_{\mathbf{x}}\mathbf{t}}$ and $\boldsymbol{\nabla_{\zeta}\mathbf{t}}$ are stated in Section 3.2. Also,

$$\boldsymbol{\nabla_{\xi}\mathbf{d}} = \boldsymbol{\nabla_{\xi}}\left(\mathbf{x} - \mathbf{b}\big(\mathbf{Q}(\boldsymbol{\xi}), \mathbf{t}(\mathbf{x}, \boldsymbol{\zeta})\big)\right) = -\boldsymbol{\nabla_{\xi}\mathbf{b}} = -\boldsymbol{\nabla_{\mathbf{Q}}\mathbf{b}\nabla_{\xi}\mathbf{Q}} = -\boldsymbol{\nabla_{\mathbf{Q}}\mathbf{b}}. \tag{3.37}$$

$$\boldsymbol{\nabla_{\mathbf{x}}\mathbf{d}} = \boldsymbol{\nabla_{\mathbf{x}}}\left(\mathbf{x} - \mathbf{b}\big(\mathbf{Q}(\boldsymbol{\xi}), \mathbf{t}(\mathbf{x}, \boldsymbol{\zeta})\big)\right) = \mathbf{I} - \boldsymbol{\nabla_{\mathbf{x}}\mathbf{b}} = \mathbf{I} - \boldsymbol{\nabla_{\mathbf{t}}\mathbf{b}\nabla_{\mathbf{x}}\mathbf{t}}. \tag{3.38}$$

$$\boldsymbol{\nabla_{\zeta}\mathbf{d}} = \boldsymbol{\nabla_{\zeta}}\left(\mathbf{x} - \mathbf{b}\big(\mathbf{Q}(\boldsymbol{\xi}), \mathbf{t}(\mathbf{x}, \boldsymbol{\zeta})\big)\right) = -\boldsymbol{\nabla_{\zeta}\mathbf{b}} = -\boldsymbol{\nabla_{\mathbf{t}}\mathbf{b}\nabla_{\zeta}\mathbf{t}}. \tag{3.39}$$

When parameters are equal to zero ($\boldsymbol{\xi} = \boldsymbol{\zeta} = \mathbf{0}$), solving the fixed-point problem of **Equation (3.29)** brings a result of the SUE solution. Either the link-based STOCH3 algorithm with the MSA or the DFS loading procedure with the MSA and STOCH3-efficient route definition in Section 3.3 could be used to have the SUE solution. Since the SUE solution is attained, the problem of sensitivity analysis is the calculation of the changes of link flow x_{ij} caused by small changes in ξ^{rs} and ζ_{ij}, i.e. computing the partial derivatives. Since we compute the partial derivative of link flow with respect to one uncertainty parameter, another is assumed equal 0.

The first problem is the calculation of $\boldsymbol{\nabla_{\zeta}\mathbf{x}}$ with $\boldsymbol{\xi} = \mathbf{0}$ and fixed travel demand $\{Q_0^{rs}\}$. From the general formula for derivative of the implicit function,

$$\boldsymbol{\nabla_{\zeta}\mathbf{x}} = -\boldsymbol{\nabla_{\mathbf{x}}\mathbf{d}^{-1}\nabla_{\zeta}\mathbf{d}}. \tag{3.40}$$

Substituting $\boldsymbol{\nabla_x d}$ and $\boldsymbol{\nabla_\zeta d}$ by using **Equations (3.38) and (3.39)**, we have:

$$\boldsymbol{\nabla_\zeta x} = (\mathbf{I} - \boldsymbol{\nabla_t b \nabla_x t})^{-1}\left(\boldsymbol{\nabla_t b \nabla_\zeta t}\right). \tag{3.41}$$

With the same approach, the second problem of calculating $\boldsymbol{\nabla_\xi x}$ with $\boldsymbol{\zeta = 0}$ is resolved by:

$$\boldsymbol{\nabla_\xi x} = -\boldsymbol{\nabla_x d}^{-1}\boldsymbol{\nabla_\xi d}. \tag{3.42}$$

According to **Equations (3.37) and (3.38)**:

$$\boldsymbol{\nabla_\xi x} = (\mathbf{I} - \boldsymbol{\nabla_t b \nabla_x t})^{-1}\boldsymbol{\nabla_Q b}. \tag{3.43}$$

From **Equation (3.1)**, each element of $\boldsymbol{\nabla_x t}$ and $\boldsymbol{\nabla_\zeta t}$ is given by:

$$\left.\frac{\partial t_{ij}}{\partial x_{ij}}\right|_{\zeta_{ij}=0} = \left.\frac{\partial\left(\left[t_{ij}^0 + \zeta_{ij}\right]\left[1 + \alpha\left(\frac{x_{ij}}{Cap_{ij}}\right)^\beta\right]\right)}{\partial x_{ij}}\right|_{\zeta_{ij}=0} = \frac{t_{ij}^0 \alpha\beta {x_{ij}}^{\beta-1}}{{Cap_{ij}}^\beta}, \tag{3.44}$$

$$\left.\frac{\partial t_{ij}}{\partial \zeta_{ij}}\right|_{\zeta_{ij}=0} = \left.\frac{\partial\left(\left[t_{ij}^0 + \zeta_{ij}\right]\left[1 + \alpha\left(\frac{x_{ij}}{Cap_{ij}}\right)^\beta\right]\right)}{\partial \zeta_{ij}}\right|_{\zeta_{ij}=0} = 1 + \alpha\left(\frac{x_{ij}}{Cap_{ij}}\right)^\beta. \tag{3.45}$$

Since $\boldsymbol{\nabla_x t}$ and $\boldsymbol{\nabla_\zeta t}$ are easily computed, the difficulties in here are the calculations of $\boldsymbol{\nabla_t b}$ and $\boldsymbol{\nabla_Q b}$. Each element of $\boldsymbol{\nabla_t b}$ and $\boldsymbol{\nabla_Q b}$ is calculated as follows:

$\frac{\partial b_{ij}}{\partial t_{gh}}$ is calculated as

$$\begin{aligned}\frac{\partial b_{ij}}{\partial t_{gh}} &= \frac{\partial \sum_{rs\in W}\left(Q_0^{rs}\left[\frac{\sum_{k\epsilon K^{rs}}\delta_{ij,k}^{rs}\exp(-\theta c_k^{rs})}{\sum_{k\epsilon K^{rs}}\exp(-\theta c_k^{rs})}\right]\right)}{\partial t_{gh}} \\ &= \sum_{rs\in W}\left(Q_0^{rs}\left[\frac{\partial\left(\frac{\sum_{k\epsilon K^{rs}}\delta_{ij,k}^{rs}\exp(-\theta c_k^{rs})}{\sum_{k\epsilon K^{rs}}\exp(-\theta c_k^{rs})}\right)}{\partial t_{gh}}\right]\right)\end{aligned} \tag{3.46}$$

Applying the derivative rule of the quotient (see **Appendix C**), it is clear that

$$\frac{\partial\left(\dfrac{\sum_{k\epsilon K^{rs}}\delta_{ij,k}^{rs}\exp(-\theta c_k^{rs})}{\sum_{k\epsilon K^{rs}}\exp(-\theta c_k^{rs})}\right)}{\partial t_{gh}}$$

$$=\frac{\dfrac{\partial\sum_{k\epsilon K^{rs}}\delta_{ij,k}^{rs}\exp(-\theta c_k^{rs})}{\partial t_{gh}}}{\sum_{k\epsilon K^{rs}}\exp(-\theta c_k^{rs})}-\frac{\left[\sum_{k\epsilon K^{rs}}\delta_{ij,k}^{rs}\exp(-\theta c_k^{rs})\right]}{\sum_{k\epsilon K^{rs}}\exp(-\theta c_k^{rs})}\frac{\dfrac{\partial\sum_{k\epsilon K^{rs}}\delta_{ij,k}^{rs}\exp(-\theta c_k^{rs})}{\partial t_{gh}}}{\sum_{k\epsilon K^{rs}}\exp(-\theta c_k^{rs})}$$

we first need to find

$$\begin{aligned}\frac{\partial\sum_{k\epsilon K^{rs}}\delta_{ij,k}^{rs}\exp(-\theta c_k^{rs})}{\partial t_{gh}}&=\sum_{k\epsilon K^{rs}}\delta_{ij,k}^{rs}\frac{\partial\exp(-\theta c_k^{rs})}{\partial t_{gh}}\\&=\sum_{k\epsilon K^{rs}}\delta_{ij,k}^{rs}\exp(-\theta c_k^{rs})\frac{\partial(-\theta c_k^{rs})}{\partial t_{gh}}\\&=\sum_{k\epsilon K^{rs}}\delta_{ij,k}^{rs}\exp(-\theta c_k^{rs})\left(-\theta\delta_{gh,k}^{rs}\right)\\&=-\theta\sum_{k\epsilon K^{rs}}\delta_{ij,k}^{rs}\delta_{gh,k}^{rs}\exp(-\theta c_k^{rs}),\end{aligned}\qquad(3.47)$$

and

$$\frac{\partial \sum_{k\epsilon K^{rs}} \exp(-\theta c_k^{rs})}{\partial t_{gh}} = \sum_{k\epsilon K^{rs}} \frac{\partial \exp(-\theta c_k^{rs})}{\partial t_{gh}}$$
$$= \sum_{k\epsilon K^{rs}} \exp(-\theta c_k^{rs}) \frac{\partial(-\theta c_k^{rs})}{\partial t_{gh}} \quad (3.48)$$
$$= \sum_{k\epsilon K^{rs}} \exp(-\theta c_k^{rs}) \left(-\theta \delta_{gh,k}^{rs}\right)$$
$$= -\theta \sum_{k\epsilon K^{rs}} \delta_{gh,k}^{rs} \exp(-\theta c_k^{rs}),$$

Using **Equations (3.47) and (3.48)**, we have

$$\frac{\partial \left(\frac{\sum_{k\epsilon K^{rs}} \delta_{ij,k}^{rs} \exp(-\theta c_k^{rs})}{\sum_{k\epsilon K^{rs}} \exp(-\theta c_k^{rs})} \right)}{\partial t_{gh}}$$

$$= \frac{-\theta \sum_{k\epsilon K^{rs}} \delta_{ij,k}^{rs} \delta_{gh,k}^{rs} \exp(-\theta c_k^{rs})}{\sum_{k\epsilon K^{rs}} \exp(-\theta c_k^{rs})} - \frac{\left[\sum_{k\epsilon K^{rs}} \delta_{ij,k}^{rs} \exp(-\theta c_k^{rs})\right]\left[-\theta \sum_{k\epsilon K^{rs}} \delta_{gh,k}^{rs} \exp(-\theta c_k^{rs})\right]}{\left(\sum_{k\epsilon K^{rs}} \exp(-\theta c_k^{rs})\right)^2}$$

from the definition of $x_{ij,gh}^{rs}$ and x_{ij}^{rs} in Section 3.2, it is clear that:

$$\frac{x_{ij,gh}^{rs}}{Q_0^{rs}} = \frac{\sum_{k\epsilon K^{rs}} \delta_{ij,k}^{rs} \delta_{gh,k}^{rs} \exp(-\theta c_k^{rs})}{\sum_{k\epsilon K^{rs}} \exp(-\theta c_k^{rs})}, \quad (3.49)$$

$$\frac{x_{ij}^{rs}}{Q_0^{rs}} = \frac{\sum_{k\epsilon K^{rs}} \delta_{ij,k}^{rs} \exp(-\theta c_k^{rs})}{\sum_{k\epsilon K^{rs}} \exp(-\theta c_k^{rs})}, \quad (3.50)$$

thus,

$$\frac{\partial\left(\dfrac{\sum_{k\epsilon K^{rs}}\delta_{ij,k}^{rs}\exp(-\theta c_k^{rs})}{\sum_{k\epsilon K^{rs}}\exp(-\theta c_k^{rs})}\right)}{\partial t_{gh}}=\frac{-\theta x_{ij,gh}^{rs}}{Q_0^{rs}}-\frac{x_{ij}^{rs}}{Q_0^{rs}}\frac{\left[-\theta x_{gh}^{rs}\right]}{Q_0^{rs}}=\frac{-\theta x_{ij,gh}^{rs}}{Q_0^{rs}}+\frac{\theta x_{ij}^{rs}}{Q_0^{rs}}\frac{x_{gh}^{rs}}{Q_0^{rs}}$$

and, **Equation (3.46)** becomes:

$$\begin{aligned}\frac{\partial b_{ij}}{\partial t_{gh}}&=\sum_{rs\in W}\left(Q_0^{rs}\left[-\frac{\theta x_{ij,gh}^{rs}}{Q_0^{rs}}+\frac{\theta x_{ij}^{rs}}{Q_0^{rs}}\frac{x_{gh}^{rs}}{Q_0^{rs}}\right]\right)\\&=\theta\sum_{rs\in W}\left(\left[-x_{ij,gh}^{rs}+\frac{x_{ij}^{rs}x_{gh}^{rs}}{Q_0^{rs}}\right]\right),\end{aligned}\tag{3.51}$$

also, $\frac{\partial b_{ij}}{\partial Q^{rs}}$ is calculated as follows:

$$\begin{aligned}\frac{\partial b_{ij}}{\partial Q^{rs}}&=\frac{\partial\sum_{rs\in W}\left(Q^{rs}\left[\dfrac{\sum_{k\epsilon K^{rs}}\delta_{ij,k}^{rs}\exp(-\theta c_k^{rs})}{\sum_{k\epsilon K^{rs}}\exp(-\theta c_k^{rs})}\right]\right)}{\partial Q^{rs}}\\&=\frac{\sum_{k\epsilon K^{rs}}\delta_{ij,k}^{rs}\exp(-\theta c_k^{rs})}{\sum_{k\epsilon K^{rs}}\exp(-\theta c_k^{rs})}=\frac{x_{ij}^{rs}}{Q_0^{rs}}.\end{aligned}\tag{3.52}$$

From **Equations (3.51) and (3.52)**, the difficulties become the calculations of x_{ij}^{rs} and $x_{ij,gh}^{rs}$. The difficulty of calculating $x_{ij,gh}^{rs}$ is similar to the difficulty of the sensitivity analysis method of Ying and Miyagi[1)]. Using p_{gh}^{js} variable and running STOCH3 algorithm to calculate this variable was proposed by Ying and Miyagi[1)]. However, the use of this technique did not achieve the effect as analyzed above (from page 71 to page 75). In our approach, the DFS loading procedure using the STOCH3-efficient route definition introduced in Section 3.3 is used to solve these difficulties. **Equations (3.51) and (3.52)** are repeatedly computed for

each OD pair, so we only need variables including link-based variables and node-based variables that reduce storage cost. For example, we just use a variable with $|A| \times |A|$ memory to store the $x_{ij,gh}^{rs}$ variable.

At SUE state, when the travel time of all links are fixed, x_{ij}^{rs} can be calculated by using the DFS algorithm once because x_{ij} is computed for each OD pair and $x_{ij} = \sum_{rs \in W} x_{ij}^{rs}$ in the DFS algorithm in Section 3.3. After x_{ij}^{rs} is easily computed with the DFS algorithm approach, we need to compute $x_{ij,gh}^{rs}$. This variable is computed through x_{ij}^{rs} and $x_{ij \to gh}^{rs}$ (see **Equation (3.8**)). The next problem of calculating $x_{ij \to gh}^{rs}$ is solved by wielding the DFS method with the STOCH3-efficient route definition.

We consider that $K_{ij \to gh}^{rs}$ denotes the set of efficient routes of OD pair rs in which both links ij and gh are used and link ij is used before link gh. From the definition of $x_{ij \to gh}^{rs}$, we have:

$$x_{ij \to gh}^{rs} = Q_0^{rs} \frac{\sum_{k \in K_{ij \to gh}^{rs}} \exp(-\theta c_k^{rs})}{\sum_{k \in K^{rs}} \exp(-\theta c_k^{rs})}. \quad (3.53)$$

Denoting the numerator of **Equation (3.53)** by a variable $d_{ij,gh}^{rs}$ with $|A| \times |A|$ memory. Because $x_{ij \to gh}^{rs}$ will be repeatedly computed for each OD pair, we do not need to store this variable and reuse it after the calculation for each OD pair has ended. As highlighted in Section 3.3, by running the DFS algorithm from origin node r to destination node s with recursive technique, each time we reach the destination s, we will have the information of all the efficient links that are visited, i.e. all links belong to an efficient route k. We can only spend a one-dimensional array, $mark_{ij}$, to mark the route from origin r to destination s that can be reused for saving memory. The maximum number of elements for this array is only the number of links on the network. We also use the same variables and procedure shown in Section 3.3. At each time destination node s is reach, we have $vn_s^{rs} = \exp(-\theta c_k^{rs})$ (see **Equation (3.15)**) and if link ij is before link gh in the mark array, we add vn_s^{rs} to $d_{ij,gh}^{rs}$. With the recursive technique, we can reach the destination node s by $|K^{rs}|$ times and the numerator of **Equation (3.53)** is totally

computed because all efficient routes K^{rs} including $K_{ij\to gh}^{rs}$ are considered. The denominator of **Equation (3.53)**, denoted by dnm^{rs} variable (see **Equation (3.16)**), could be computed by the DFS loading procedure. The calculation of x_{ij}^{rs} is also conducted through dnm^{rs}, wl_{ij}^{rs} variables by the DFS loading procedure (see Section 3.3). After the DFS loading procedure completes the calculation for each OD pair, we have:

$$x_{ij\to gh}^{rs} = Q_0^{rs} \frac{d_{ij,gh}^{rs}}{dnm^{rs}}, \tag{3.54}$$

$$x_{ij}^{rs} = Q_0^{rs} \frac{num_{ij}^{rs}}{dnm^{rs}}. \tag{3.55}$$

For example, if we use this procedure to calculate num_{ij}^{rs}, $d_{ij,gh}^{rs}$ and dnm^{rs} variables in the application to the network shown in **Figure 3.1 and Table 3.4**, the results of these variables at each time reaching destination node s are illustrated through the following Table.

Table 3.13 The example of using DFS loading procedure to calculate num_{ij}^{rs}, $d_{ij,gh}^{rs}$, dnm^{rs}, x_{ij}^{rs}, $x_{ij\to gh}^{rs}$ and variables

No.	Link is traversed	Mark array	vn_i^{rs}	vn_j^{rs}	Note
1	*r*1	{*r*1}	1.000	0.368	
2	12	{*r*1,12}	0.368	0.135	
3	2*s*	**{*r*1,12,2*s*}**	0.135	0.050	dnm^{rs} =0.050;

ij	*r*1	12	14	2*s*	25	45	56	6*s*
num_{ij}^{rs}	0.050	0.050	0	0.050	0	0	0	0

$d_{ij,gh}^{rs}$:

ij \ *gh*	*r*1	12	14	2*s*	25	45	56	6*s*
*r*1		0.050		0.050				
12				0.050				
⋮								

4	25	{*r*1,12, 25}	0.135	0.082	
5	56	{*r*1,12, 25, 56}	0.082	0.067	
6	6*s*	**{*r*1,12, 25, 56, 6*s*}**	0.067	0.041	dnm^{rs} =0.050 + 0.041= 0.091;

ij	*r*1	12	14	2*s*	25	45	56	6*s*
num_{ij}^{rs}	0.091	0.091		0.050	0.041		0.041	0.041

$d_{ij,gh}^{rs}$:

ij \ *gh*	*r*1	12	14	2*s*	25	45	56	6*s*
*r*1		0.091		0.050	0.041		0.041	0.041
12				0.050	0.041		0.041	0.041
14								
2*s*								
25							0.041	0.041
45								
56								0.041
6*s*								

7	14	{*r*1,14}	0.368	0.223	
8	45	{*r*1,14,45}	0.223	0.100	
9	56	{*r*1,14,45,56}	0.100	0.082	
10	6*s*	**{*r*1,14,45,56,6*s*}**	0.082	0.050	dnm^{rs} =0.091+ 0.050= 0.140;

ij	*r*1	12	14	2*s*	25	45	56	6*s*
num_{ij}^{rs}	0.140	0.091	0.050	0.050	0.041	0.050	0.091	0.091

$d_{ij,gh}^{rs}$:

ij \ *gh*	*r*1	12	14	2*s*	25	45	56	6*s*
*r*1		0.091	0.050	0.050	0.041	0.050	0.091	0.091
12				0.050	0.041		0.041	0.041
14						0.050	0.050	0.050
2*s*								
25							0.041	0.041
45							0.050	0.050
56								0.091
6*s*								

ij	*r*1	12	14	2*s*	25	45	56	6*s*
x_{ij}^{rs}	100.00	64.52	35.48	35.48	29.05	35.48	64.52	64.52

$x_{ij\to gh}^{rs}$:

ij \ *gh*	*r*1	12	14	2*s*	25	45	56	6*s*
*r*1		64.52	35.48	35.48	29.05	35.48	64.52	64.52
12				35.48	29.05		29.05	29.05
14						35.48	35.48	35.48
2*s*								
25							29.05	29.05
45							35.48	35.48
56								64.52
6*s*								

After calculating x_{ij}^{rs} and $x_{ij\to gh}^{rs}$, we easily calculate $x_{ij,gh}^{rs}$. When all difficulties of calculation are solved, the gradient matrices $\boldsymbol{\nabla_{\zeta}\mathbf{x}}$ and $\boldsymbol{\nabla_{\xi}\mathbf{x}}$ are computed from **Equations (3.41) and (3.43)**. As can be seen, the formulas for calculating these matrices are different from the approach of Ying and Miyagi[1)]. The issues of the approach of Ying and Miyagi[1)] mentioned above have been resolved. Firstly, the mathematical formulations do not use the definitions of "true" derivative and "apparent" derivative. Secondly, if there is a link have no flow (x_{ij} =0), the calculations of **Equations (3.41) and (3.43)** are not be affected. Thirdly, our approach solves the weakness of using p_{gh}^{js} variable of Ying and Miyagi[1)] because we do not need to use this technique to calculate $x_{ij\to gh}^{rs}$. Besides, we compute for each OD pair, the variables are reused after each calculation for each pair of OD pairs. So, the number of OD pairs does not affect the memory and computer memory is saved when using only link-based and node-based variables.

The calculation process of sensitivity analysis for the logit-based static SUE traffic assignment based on the DFS loading procedure, called algorithm 2, is as follows:

Step 1: Compute the SUE by the MSA, which repeatedly uses the DFS loading procedure with the STOCH3-efficient route definition until the link flows converge. The results of link flow x_{ij} and link travel time t_{ij} are attained.

Step 2: At SUE state, based on the link travel time $\{t_{ij}\}$, set $a_{ij} \coloneqq \exp(-\theta\, t_{ij})$ for each link ij and calculate $\boldsymbol{\nabla_{\mathbf{x}}\mathbf{t}}$, $\boldsymbol{\nabla_{\zeta}\mathbf{t}}$ by using **Equations (3.44), (3.45).**

Step 3: This step is run for each OD pair

(a) Set all num_{ij}^{rs}, $mark_{ij}, d_{ij,gh}^{rs}$ and vn_n^{rs} to 0. Set vn_r^{rs}:=1 and $dnm^{rs} = 0$. Carrying the DFS algorithm from origin node r to destination node s with recursive technique and using the array $mark_{ij}$ to mark the efficient route. After each efficient link is visited, we update $vn_j^{rs} = a_{ij} vn_i^{rs}$. When reaching destination node s, we add vn_s^{rs} to num_{ij}^{rs} ($num_{ij}^{rs} \coloneqq num_{ij}^{rs} + vn_s^{rs}$) if link ij is included in the mark array, add vn_s^{rs} to $d_{ij,gh}^{rs}$ if link ij is

used before link gh in the mark array and add vn_s^{rs} to $dnm^{rs}(dnm^{rs} \coloneqq dnm^{rs} + vn_s^{rs})$.

(b) When the DFS loading procedure completes the calculation for each OD pair, num_{ij}^{rs}, $d_{ij,gh}^{rs}$ and dnm^{rs} variables are completely computed and used to calculate:

$$x_{ij\to gh}^{rs} = Q_0^{rs}\frac{d_{ij,gh}^{rs}}{dnm^{rs}},$$

$$x_{ij}^{rs} = Q_0^{rs}\frac{num_{ij}^{rs}}{dnm^{rs}},$$

$$x_{ij,gh}^{rs} = x_{gh,ij}^{rs} = \begin{cases} x_{ij\to gh}^{rs} + x_{gh\to ij}^{rs}, & \text{if } ij \neq gh, \\ x_{ij}^{rs}, & \text{otherwise.} \end{cases}$$

Because num_{ij}^{rs}, $d_{ij,gh}^{rs}$ and dnm^{rs} variables are only used to calculate x_{ij}^{rs}, $x_{ij\to gh}^{rs}$, they are reused in the next calculation for another OD pair.

(c) Calculate $\boldsymbol{\nabla_t \mathbf{b}}$ and $\boldsymbol{\nabla_Q \mathbf{b}}$: Each element of these matrices is calculated by using **Equations 3.51 and 3.52**. Because $x_{ij,gh}^{rs}$, x_{ij}^{rs} are only used to calculate $\frac{\partial b_{ij}}{\partial t_{gh}}$, $\frac{\partial b_{ij}}{\partial Q^{rs}}$, they are reused in the next calculation for another OD pair. After all OD pairs are considered, $\boldsymbol{\nabla_t \mathbf{b}}$ and $\boldsymbol{\nabla_Q \mathbf{b}}$ are completely computed.

Step 4: Calculate $\boldsymbol{\nabla_\zeta \mathbf{x}}$, $\boldsymbol{\nabla_\xi \mathbf{x}}$ according to **Equations (3.41) and (3.43)**.

Using the DFS algorithm has the advantage of not having to arrange the links in descending order of access time from origin node r and can combine $x_{ij,gh}^{rs}$ and x_{ij}^{rs} calculations in the same calculation for each OD pair. However, it needs to take into account the $|K^{rs}|$ times reaching the destination node s in the calculation algorithm. That is, it depends on the number of efficient routes defined in the route-set. This approach is independent of the route variable and the calculation time is great in the application to Kanazawa road network that will be shown later. However, to ensure the ability to apply to the more complex road network with a huge amount of routes in the implicit route-set, an alternative approach is using only the link-based STOCH3 algorithm will be given.

Because x_{ij}^{rs} could be computed by running the STOCH3 algorithm for each

OD pair rs (see **Appendix B**). The difficulty lies in the calculation of $x_{ij\to gh}^{rs}$, the calculation of the algorithm 1 (see pages 70, 71) has proved ineffective. Redefining the effective route from j to s regardless of the initial origin node r has resulted in a change in the implicit efficient route-set and resulted in a false result. We have proved that $x_{ij\to gh}^{rs}$ could be computed by running STOCH3 algorithm once from j to s for all links gh by keeping the order of reference shortest generalized travel time from origin r and setting all node weight variables, wn_n^{rs}, to 0, wn_j^{rs}:=1 in the forward pass and xn_s^{rs}:= x_{ij}^{rs} in the backward pass (see **Appendix B**). As a result, the procedure of calculating sensitivity analysis formulations of the logit-based static SUE traffic assignment based on the STOCH3, called algorithm 3, is as follows:

Step 1: Compute the logit-based SUE by the MSA, which repeatedly uses the STOCH3 algorithm until the link flows converge. The results of link flows x_{ij} and link travel times t_{ij} are worked out.

Step 2: Based on the link travel time $\{t_{ij}\}$ that is calculated in ***Step 1***, compute $a_{ij} \coloneqq \exp(-\theta\, t_{ij})$ for each link ij and $\mathbf{\nabla_x t}$, $\mathbf{\nabla_\zeta t}$ by using **Equations (3.44), (3.45).**

Step 3: This step is run for each OD pair

(a) Calculate x_{ij}^{rs}:

- For each link ij considered in the order of increasing reference access time from r, using **Equation (3.13)** to compute wl_{ij}^{rs}.
- For each link ij considered in the order of decreasing reference access time from r, using **Equation (3.14)** to compute xl_{ij}^{rs}.
- Setting $x_{ij}^{rs} \coloneqq xl_{ij}^{rs}$.

(b) Calculate $x_{ij\to gh}^{rs}$: For each link $x_{ij}^{rs} \neq 0$, running STOCH3 algorithm from node j to node s to calculate $x_{ij\to gh}^{rs}$ as follows:

- For each link gh considered in the order of increasing reference access time from r, wl_{gh}^{js} is calculated

$$wl_{gh}^{js} = \begin{cases} a_{gh} & if\ g = j\ and\ \Omega_{gh}^{r} = 1 \\ a_{gh} \sum_{m \epsilon N_g^{in}} wl_{mg}^{rs} & if\ g \neq j\ and\ \Omega_{gh}^{r} = 1 \\ 0 & otherwise \end{cases}$$

- For each link gh taken in the order of decreasing reference access time from r, compute $x_{ij\to gh}^{rs}$

$$x_{ij\to gh}^{rs} = \begin{cases} x_{ij}^{rs} \dfrac{wl_{gh}^{js}}{\sum_{m \epsilon N_h^{in}} wl_{mh}^{js}} & if\ h = s \\ \left(\sum_{n \epsilon N_h^{out}} xl_{hn}^{js} \right) \dfrac{wl_{gh}^{js}}{\sum_{m \epsilon N_h^{in}} wl_{mh}^{js}} & if\ h \neq s \end{cases}$$

(c) Calculate $x_{ij,gh}^{rs}$:

$$x_{ij,gh}^{rs} = x_{gh,ij}^{rs} = \begin{cases} x_{ij\to gh}^{rs} + x_{gh\to ij}^{rs}, & \text{if } ij \neq gh, \\ x_{ij}^{rs}, & \text{otherwise.} \end{cases}$$

(d) Calculate $\mathbf{\nabla_t b}$ and $\mathbf{\nabla_Q b}$: Using **Equations (3.51) and (3.52)** to calculate $\frac{\partial b_{ij}}{\partial t_{gh}}$, $\frac{\partial b_{ij}}{\partial Q^{rs}}$.

x_{ij}^{rs}, $x_{ij\to gh}^{rs}$ and $x_{ij,gh}^{rs}$ are reused for the next calculation for the next OD pair. After all OD pairs are regarded, $\mathbf{\nabla_t b}$ and $\mathbf{\nabla_Q b}$ are completely computed.

Step 4: Compute $\mathbf{\nabla_\zeta x}$, $\mathbf{\nabla_\xi x}$ according to **Equations (3.41) and (3.43)**.

When these gradient matrices ($\mathbf{\nabla_\zeta x}$, $\mathbf{\nabla_\xi x}$) can be computed, the approximate link flows under any changes of $\boldsymbol{\zeta}$ and $\boldsymbol{\xi}$ are calculated by using the first-order Taylor expansion as follows:

$$\tilde{\mathbf{x}}_{\boldsymbol{\zeta}} = \mathbf{x} + \mathbf{\nabla_\zeta x}\boldsymbol{\zeta} \tag{3.56}$$

$$\tilde{\mathbf{x}}_{\boldsymbol{\xi}} = \mathbf{x} + \mathbf{\nabla_\xi x}\boldsymbol{\xi} \tag{3.57}$$

where $\tilde{\mathbf{x}}_{\boldsymbol{\zeta}} = (\tilde{x}_{12,\zeta}, \dots, \tilde{x}_{ij,\zeta}, \dots, \tilde{x}_{mn,\zeta})$ and $\tilde{\mathbf{x}}_{\boldsymbol{\xi}} = (\tilde{x}_{12,\xi}, \dots, \tilde{x}_{ij,\xi}, \dots, \tilde{x}_{mn,\xi})$ represent the vector of approximate link flows according to the change of $\boldsymbol{\zeta}$ and $\boldsymbol{\xi}$, respectively.

3.5 Computational evidence

In this section, a simple application and Kanazawa city urban application are adopted to demonstrate the accuracy and effectiveness of the proposed method.

Assuming perturbation link flows are the link flows corresponding to the specific parameter values. Here, $\tilde{\mathbf{x}}$ denotes the estimated perturbation link flows calculated by using the sensitivity analysis method, and $\mathbf{x}$ denotes the actual perturbation link flows at SUE state calculated by the STOCH3 algorithm with the MSA or the DFS loading procedure using STOCH3-efficient route definition with the MSA. To assess the accuracy of the proposed method, performance indicators are defined. In these indicators, the $RMSE$ and $\%RMS$ indicate the error of comparing between approximated flows and actual calculation flows:

$$RMSE = \sqrt{\frac{1}{|A|}\sum_{ij\in A}\left(\tilde{x}_{ij} - x_{ij}\right)^2}, \tag{3.58}$$

$$\%RMS = \frac{RMSE}{\frac{1}{|A|}\sum_{ij\in A} x_{ij}}100(\%). \tag{3.59}$$

We will implement five sensitivity analysis methods as follows:

(I) The method proposed by Ying and Miyagi[1)] based on Dial's algorithm.

(II) The method of Ying and Miyagi[1)] is modified based on the STOCH3 algorithm shown on pages 70, 71 (algorithm 1).

(III) The formulations of $\boldsymbol{\nabla_{\zeta}\mathbf{x}}$, $\boldsymbol{\nabla_{\xi}\mathbf{x}}$ are the new approach according to **Equations (3.41) and (3.43)**. However, the method used to calculate SUE solution and the calculations of x_{ij}^{rs}, $x_{ij,gh}^{rs}$ are the procedure of Ying and Miyagi[1)] based on the STOCH3 algorithm (***steps 1, 2, 3, 4*** on pages 70, 71).

(IV) The new formulations and calculation process proposed in Section 3.4 based on the DFS loading procedure (algorithm 2).

(V) The new formulations and calculation process proposed in Section 3.4 based on the STOCH3 algorithm (algorithm 3).

The same formulations and calculation process are used in (I) and (II) based

on the method of Ying and Migagi. However, (II) uses the STOCH3 algorithm instead of Dial's algorithm because Dial's algorithm does not converge to the equilibrium solution in the application to Kanazawa road network. Also, (III) and (IV) use the same new formulations proposed in Section 3.4. The differences between (III) and (IV) are in the calculation method used to achieve SUE solution and the calculating of x_{ij}^{rs}, $x_{ij,gh}^{rs}$. For achieving the SUE solution, while (III) uses the STOCH3 algorithm solution, (IV) uses the DFS loading procedure proposed in Section 3.3. This difference is used to compare the calculational time between using the DFS loading procedure and using the STOCH3 algorithm. Because the new approach also needs to calculate x_{ij}^{rs}, $x_{ij,gh}^{rs}$, the technique of Ying and Miyagi[1)] could be used to calculate these variables (see ***Steps 2, 3, and 4*** on pages 70, 71). The differences in the calculating $x_{ij}^{rs}, x_{ij,gh}^{rs}$ of (III) and (IV) are used to compare the efficient usage between the technique of Ying and Miyagi[1)] and the new approach. Besides, the difference between (IV) and (V) lies in the algorithm used to calculate the new sensitivity analysis formulations.

The programs were coded by Fortran.90 programming language and ran on a personal computer with an Intel Core i7-8700 CPU of 3.20 GHz, 16GB RAM, and Windows 10 Home.

3.5.1 Simple example

Five approaches (I), (II), (III), (IV), and (V) use the same definition of efficient routes in which the route is efficient if it includes links that have their start node closer to the origin than their end node. While (I) uses this definition based on the travel time of each iteration, (II), (III), (IV) and (V) use this based on the fixed free-flow travel time. These approaches are applied to the small network stated in **Figure 3.4** with the same input data. Assuming that the parameter $\boldsymbol{\zeta}$ is changed from 0 to 1 in all links and we need to compute the change of SUE link traffic flows. When $\boldsymbol{\zeta = 0}$, unperturbed link flows are calculated based on either Dial's algorithm, STOCH3 algorithm, or DFS loading procedure with the MSA method. Perturbed link flows with $\boldsymbol{\zeta = 1}$ could be also calculated by using these algorithms. The calculation time of all algorithms is very small, and the difference is not significant. Besides, the results brought from these three algorithms are the same (see **Table 3.14**). From there, it can be seen that when applied to a small road network, Dial's algorithm still ensures the convergence of traffic flow. If we do not

want to run these algorithms again when the parameter **ζ** changes, we can use the sensitivity analysis method to calculate the approximate link flow results. All of the results are expressed in **Table 3.14**.

Table 3.14 The comparison between the five sensitivity analysis approaches

Link	Unperturbed flows (pcu)	Perturbed link flows with ζ = 1 (pcu)					
		Not approximate	Actual change	(I), (II), (III)		(IV), (V)	
				Approximate	Error	Approximate	Error
13	30.00	30.00	0.00	30.00	0.00	30.00	0.00
23	6.75	4.67	-2.08	4.35	0.32	4.50	-0.17
24	23.25	25.33	2.08	25.72	-0.40	25.50	0.17
32	0.0002	0.0001	-0.0001	-0.0001	-0.0001	0.0000	-0.0001
34	5.08	4.87	-0.21	4.65	0.22	4.95	0.08
35	31.67	29.80	-1.87	29.63	0.17	29.55	-0.25
46	28.33	30.20	1.87	30.37	-0.17	30.45	0.25
56	31.67	29.80	-1.87	29.63	0.17	29.55	-0.25
RMSE				**0.223**		**0.175**	
%RMS				**1.16%**		**0.90%**	

We can see that the three first approaches work out the same results in the approximated results. Because Dial's algorithm can converge to the equilibrium state in this case, the use of Ying and Miyagi's method[1)] based on the Dial's (I) and STOCH3 (II) algorithms are the same. Thus, (II) can be used to replace (I) when Dial's algorithm cannot converge to equilibrium. (III) using the new approach and the new sensitivity analysis formulas but still applying the calculation procedure of Ying and Miyagi[1)] gives very similar results to (II). Therefore, the new algorithm can replace the old method of Ying and Miyagi[1)]. Finally, (IV) and (V) give the same results. Besides, the two last approaches bring a higher accuracy with a smaller error than the three first approaches. The $RMSE$ and $\%RMS$ indicators of (IV) and (V) are all lower than (I), (II) and (III). This shows that the proposed new algorithm is capable of calculating better than the old algorithm. Although the difference is quite small because the applied model is simple with only 2 OD pairs, 7 nodes, and 8 links. The errors caused by using the old method will increase when applied to the actual network with the complexity of the network and large OD pairs, nodes, and links.

3.5.2 Application to Kanazawa road network

As shown in **Figure 3.4**, the range of Kanazawa road network covers almost the entire area of Kanazawa City. This network is based on the network used in the automobile commuting survey conducted as a supplementary survey of the person trip survey in the Kanazawa metropolitan area described later. It is composed of 272 nodes and 964 links, covering almost all major. OD traffic is generated and concentrated with each node as a centroid. Details about the nodes and links of the network are listed in **Appendix D.**

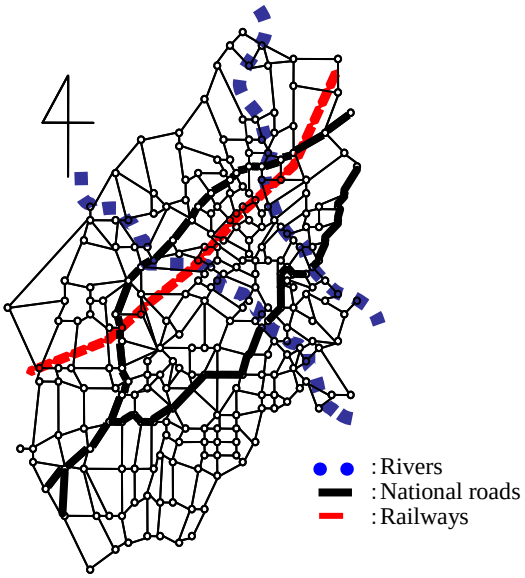


Figure 3.4 Kanazawa road network

In this study, we handle the data of the automobile commuting survey conducted as a supplementary survey of the Person Trip Survey in the Kanazawa Metropolitan Area in 1995. This survey was conducted for people who used a car while commuting. The contents consist of departure time, arrival time, route, and expansion factor. In this study, we apply the model to the morning commuting time when traffic demand peaks. The OD demand data with 383 OD pairs of the morning peak from 6:00 to 7:00 AM is used. The parameters of the BPR function are $\alpha = 1.0$, $\beta = 2.0$ and the logit-based parameter is $\theta = 1$.

Because Dial's algorithm does not give the convergence solution of link traffic flows, (I) could not be used. For (II), (III), (IV) and (V), we use the definition of STOCH3-efficient route (see Chapter 2). According to Leurent, the parameter h_{ij}^{r} could be set from 1.3 to 1.5 in the urban studies. The number of

efficient links as well as the number of efficient routes increases with the increase of h_{ij}^r. In the case of Kanazawa road network with 383 OD pairs, the number of efficient routes is 3819 routes if $h_{ij}^r = 1.5$ and it is 4995 routes if $h_{ij}^r = 1.5$. For comparison, we set h_{ij}^r to 1.5 for every link of the network and assume 4 cases: Case 1 assumes all parameters are equal to 0, case 2 sets the **ζ** parameter to **0.1** at all links and the **ξ** parameter is equal to **0**, case 3 assumes that the **ζ** parameter is **0** and the **ξ** parameter is **5** in all OD pairs, case 4 assumes that the **ζ** and **ξ** parameters change simultaneously with **0.1** and **5**, respectively.

The first comparison is the comparison between the DFS-based algorithm and the STOCH3 algorithm in the calculation time for achieving the SUE solution without the sensitivity analysis method. **Figure 3.5** shows this comparison. The algorithms are coded for each OD pair and the MSA method is run for the iterative procedure. The same results of link traffic flow are output at equilibrium state and the calculation time of the DFS-based algorithm is reduced when compared to the STOCH3 algorithm. Although the DFS-based algorithm depends on the number of efficient routes in implicit route-set, the calculation time is still lower than the STOCH3 algorithm at least when applied to the common case $h_{ij}^r \leq 1.5$ of STOCH3-efficient route definition. The reasons for this have been analyzed in Section 3.3. The exploration of the applicability of the DFS-based algorithm will be considered in the future, the focus here is on its application for the sensitivity analysis method.

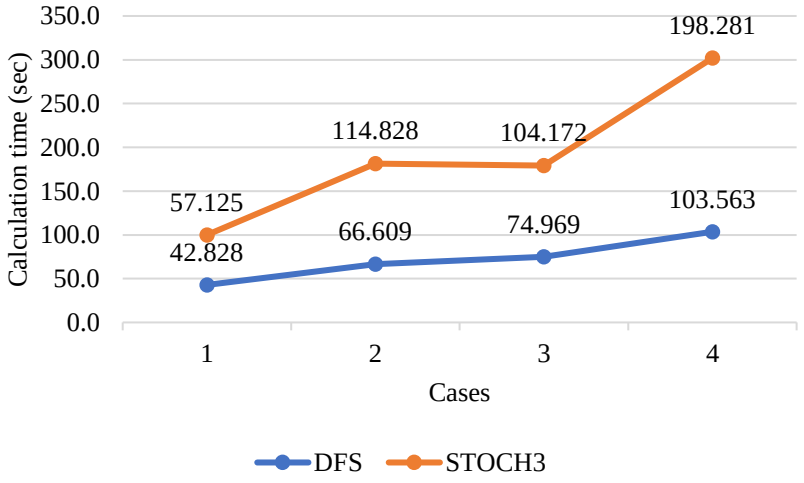


Figure 3.5 The comparison between DFS and STOCH3 algorithms in calculation time without sensitivity analysis

Secondly, if we do not want to run these algorithms again to find the SUE solutions in cases 2, 3, and 4, we can use the sensitivity analysis method when the results of the SUE solution in case 1 are obtained. (II) could not run due to some links on the Kanazawa road network that had no flows in the SUE solution. This has shown the weaknesses of the Ying and Miyagi algorithm[1)] when it cannot be applied to the case $x_{ij} = 0$ in some links. The results of (III), (IV), and (V) in calculation time are shown in **Tables 3.15**.

Table 3.15 The calculation time of the sensitivity analysis methods

Calculation time	**Sensitivity analysis methods**		
	(III)	**(IV)**	**(V)**
The first SUE solution (sec)	57.13	42.83	57.13
Sensitivity analysis formulations (sec)	65.84	28.31	35.75
Equations (3.51) and (3.52) (sec)	37.69	3.19	10.97

Table 3.15 shows the calculation time of the three sensitivity analysis methods. In calculating the sensitivity analysis formulas, the calculation time of (IV) and (V) is only half of the calculation time of (III). If not mention the inverse matrix calculation time, the difference is greater when only mentioning the time for calculating **Equations (3.51) and (3.52)**. The proposed methods will save at least 70% of the time calculated these equations. This shows the efficiency of calculating the x_{ij}^{rs} and $x_{ij,gh}^{rs}$ variables in the calculation procedure proposed in this chapter. Comparing the calculation time of (IV) and (V) in this case, it can be seen that the calculation time of (IV) is better than that of (V). However, both methods have very good computation time, and there will be many options for applying either (IV) or (V) in different cases depending on the advantages of both mentioned in Section 3.4. Further analysis of these two methods for various cases will be focused in future studies. In this case, the total calculation time of using the sensitivity method (IV) based on the initial SUE solution is nearly 1 minute 10 seconds. It saves the calculation by about 75% compared to having to run the algorithms again to get the SUE solution. Here, only 3 cases of parameter change are considered. If there are more cases, the calculation time will be more efficient because the difficulty in the sensitivity analysis method lies only in calculating the

gradient matrices. From these matrices, approximate equilibrium solutions can be calculated for various cases of changing parameters.

Consequently, the difference in accuracy is indicated in **Table 3.16**. While the error of comparing between approximated flows and actual calculation flows of (III) is quite large, it of the proposed method shows high accuracy. The percentage error ($\%RMS$) of (IV) and (V) are the same and do not exceed 0.5% if only one parameter is changed, either $\boldsymbol{\zeta}$ or $\boldsymbol{\xi}$. When changing both parameters at the same time, the approximated results of (IV) and (V) are also quite exact. Thus, the accurate calculation of (IV) and (V) is equal.

Table 3.16 Accuracy comparison between some approaches

Parameters	**Methods**	**Indicators**	
		RMSE	*%RMS*
$\boldsymbol{\zeta} = \mathbf{0.1}$	(IV), (V)	0.423	0.38%
	(III)	4.984	4.47%
$\boldsymbol{\xi} = \mathbf{5}$	(IV), (V)	0.464	0.35%
	(III)	2.652	1.99%
$\boldsymbol{\zeta} = \mathbf{0.1}$ and $\boldsymbol{\xi} = \mathbf{5}$	(IV), (V)	0.871	0.66%
	(III)	6.972	5.25%

The above results show the computational efficiency of the new sensitivity formulas and the new calculation processes both in computation time and accuracy. The results suggest that the proposed method is an accurate method to estimate the SUE model, robust to different parameter settings, and highly applicable to complex networks. In this chapter, the sensitivity analysis method is considered in the static multinomial logit-based SUE model with fixed travel demand. Extension of this method would be considered in future researches.

References

1. J. Q. Ying and T. Miyagi, “Sensitivity Analysis for Stochastic User Equilibrium Network Flows — A Dual Approach,” *Transp. Sci.*, vol. 35, no. 2, pp. 124–133, 2001.
2. C. F. Daganzo, “Unconstrained Extremal Formulation of Some Transportation Equilibrium Problems,” *Transp. Sci.*, vol. 16, no. 3, pp. 332–360, 1982.
3. F. M. Leurent, “Curbing the Computational Difficulty of the Logit Equilibrium Assignment Model,” *Transp. Res. Part B*, vol. 31, no. 4, pp. 315–326, 1997.
4. R. B. Dial, “A Probabilistic Multipath Traffic Assignment Model which Obviates Path Enumeration,” *Transp. Res.*, vol. 5, pp. 83–111, 1971.
5. T. Akamatsu, “Cyclic flows, Markov process and stochastic traffic assignment,” *Transp. Res. Part B Methodol.*, vol. 30, no. 5 PART B, pp. 369–386, 1996.
6. M. Maher, “Algorithms for logit-based stochastic user equilibrium assignment,” *Transp. Res. Part B*, vol. 32, no. 8, pp. 539–549, Nov. 1998.
7. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Second Edition*. MIT Press and McGraw-Hill, 2001.
8. T. T. Bui, S. Nakayama, and H. Yamaguchi, “Improving the Applicability of Sensitivity Analysis for Stochastic User Equilibrium Traffic Assignment based on Depth First Search,” *Paper presented at the 59th Civil Engineering Planning Research Presentation Spring Meeting in Nagoya*, 2018.

Chapter 4

Semi-dynamic stochastic user equilibrium traffic assignment with flow propagation based on the sensitivity analysis method

4.1 Overview

This chapter describes the formulation and calculation algorithm for the semi-dynamic SUE model considering the spatial-temporal movement of congestion. We first describe the assumptions and preconditions necessary for formulating the model, then summarize the formulation, and describe the computational algorithms required for implementation. To save the computation time of the model, the sensitivity analysis method will be used. The application to the simple road network and Kanazawa road network will show the model details and the applicability of the model.

4.2 Assumptions and preconditions

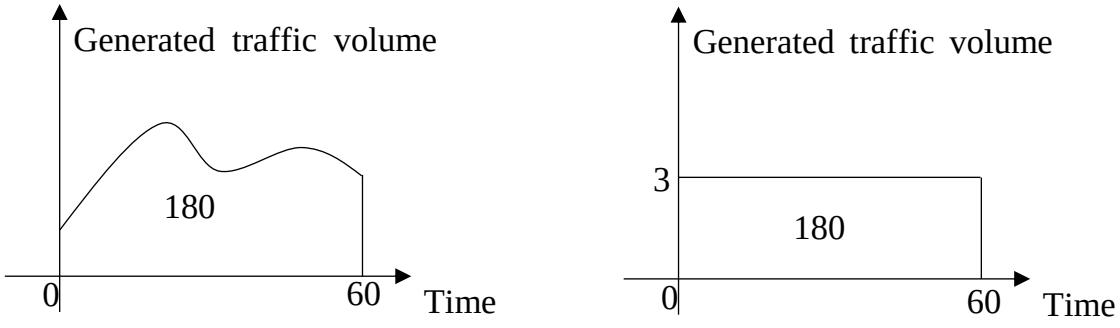
Semi-DTA model (traffic assignment model by periods) divides a day into multiple periods, STA is performed in each period, and the flow propagation is also considered between periods. At a given period, the traffic that cannot arrive at the destination at the end of the period is carried over to the next period as the residual flow. The basic idea of the model used in this study is consistent with this. Including the above, the following five assumptions are made in the formulation of the model.

Assumption 1: A day is divided into several periods with a certain length.

By dividing the day into multiple periods, it is easy to set the OD traffic volume that will be the input data for the calculation. Therefore, the model can be used in practice. Time is divided into the segment of length L. The accuracy of OD data decides the length of a segment period. The period length may be from 15 min to 90 min in many cases because of practical applications. The more accurate and dynamically detailed the OD data are, the much shorter period the length of a period is.

Assumption 2: The travel demand departs from the origin continuously at the same rate.

In the case of the coarseness of OD traffic data, travel demand is assumed to be steady-state as the following figure:



a. Actual traffic flow b. Traffic flow in the model

Figure 4.1 Conceptual diagram of traffic flow handled in the model

Since it is in a steady-state within the period, naturally the traffic volume generated is a constant value per unit time. As a concept, as shown in **Figure 4.1**, the traffic volume that is fluctuating from time to time is generated by taking the

sum of the generated traffic volume in the period and dividing it by the period length. For example, assuming that the period length is 60 minutes and 180 OD traffic volumes are generated in a certain period, the generated traffic volume per unit time (here 1 minute) is naturally 3 pcu/min. Therefore, in this case, 3 pcu/min constantly flows.

Assumption 3: Travel time on a link is the continuous, strictly increasing, and positive function of its inflow. Even if users do not travel the whole of the link, they experience the link travel time at the period of their entry into the link.

In this model, there are three types of traffic on a link in a certain period: inflow traffic, outflow traffic, and residual traffic. Link travel time is affected only by inflow traffic. Even if residual traffic occurs, the residual traffic does not affect the link. It is also a mathematical assumption that can be applied to practice. The model used in this study is intended for practical use and assumes the use of statistical and macro functions used in the static assignment. These functions are assumed to be continuous, strictly increasing, and positive. Therefore, any function that satisfies this condition can be used.

Residual flow on a link is defined as the traffic that could not pass the link within the inflow period and remains in the next period.

Assumption 4: Residual flow on a link is a continuous, strictly increasing, and non-negative function of the inflow to the link and does not exceed the inflow traffic.

The magnitude of residual link flow depends on the travel time on this link and the inflow rate to this link. Because the travel time on a link is the continuous, strictly increasing and positive function of its inflow, residual traffic for the next period is expressed as a continuous and strictly increasing function of the inflow to the link and is always non-negative and does not exceed the inflow traffic.

Assumption 5: Residual flow is added to the demand between the end node of that link and the original destination in the next period.

In the example shown above, if the traffic volume is 180 pcu in which 130 pcu can flow in an hour, 50 pcu (= 180-130) will traverse in the next period. As described in Chapter 2, it is the OD demand modification approach with the advantages of computational cost and applicability.

In this study, we also consider the traffic network with N nodes, A links where $N = \{1,2,\dots,i,j,\dots m,n\}$ is the set of nodes, $A = \{12,\dots,ij,\dots,gh,\dots,mn\}$ is the set of links. According to assumption 1, time in a day is divided into several periods in which L is the length of period τ.

To detail this model, a small virtual network in **Figure 4.2** with 4 nodes, 3 links, and 1 OD pair is used. Denoting Q_τ^{rs} is the travel demand from origin node r to destination node s in period τ, $IN_{\tau,ij}$ is the inflow to link ij in period τ, $OUT_{\tau,ij}$ is the outflow from link ij in period τ and $y_{\tau,ij}$ is the residual flow on link ij in period τ. The description of the semi-DTA model with flow propagation as follows:

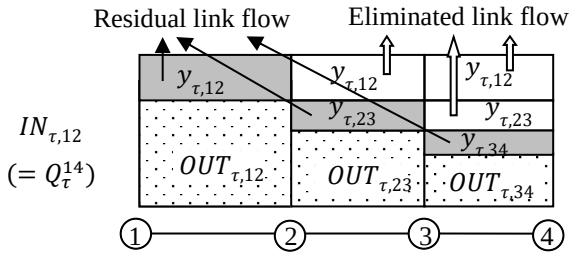


Figure 4.2 A semi-DTA model with flow propagation

- OD demand of the network is from node 1 to node 4 in period τ with period length L, denoted by Q_τ^{14}.
- The inflow to link 12 in period τ is $IN_{\tau,12} = Q_\tau^{14}$. According to assumption 3, the travel time on link 12 in period τ is the function of its inflow $t_{\tau,12} = t_{\tau,12}(IN_{\tau,12})$.
- Some of the inflow to link 12 in period τ cannot exit this link and becomes the residual flow on this link in this period τ, $y_{\tau,12}$, which is eliminated from link 23 and 34 in period τ. Besides, $y_{\tau,12}$ is propagated to the next period as travel demand from node 2 to node 4 in period $\tau + 1$. The value of $y_{\tau,12}$ depends on $t_{\tau,12}$ and this part does not travel on the subsequent links, link 23 and link 34 in period τ. The outflow from link 12 in period τ is $OUT_{\tau,12} = IN_{\tau,12} - y_{\tau,12}$. This outflow exit link 12, reaches link 23 in period τ and become the inflow to link 23 in period τ. This inflow will experience travel time $t_{\tau,23} = t_{\tau,23}(IN_{\tau,12} - y_{\tau,12})$. Some of the inflow to

link 23 in period τ become the residual flow on this link in this period, $y_{\tau,23}$, that is eliminated from link 34 in period τ and considered as travel demand from node 3 to node 4 in period $\tau + 1$. The outflow from link 23 in period τ is $OUT_{\tau,23} = IN_{\tau,12} - y_{\tau,12} - y_{\tau,23}$. This outflow exits link 23, becomes inflow to link 34 in period τ and experiences travel time $t_{\tau,34} = t_{\tau,34}(IN_{\tau,12} - y_{\tau,12} - y_{\tau,23})$. Similarly, the residual flow on link 34 in period τ, $y_{\tau,34}$, is calculated. Because the destination is reached, this residual flow arrives immediately at the destination without consideration in the next period. Besides, the outflow from link 34 in period τ is $OUT_{\tau,34} = IN_{\tau,12} - y_{\tau,12} - y_{\tau,23} - y_{\tau,23}$. This outflow is the flow that can reach the destination in period τ.

The formulation, such as residual traffic volume and equilibrium state, will be described in detail in the next section.

4.3. Modeling

To formulate the model, the logit model (Sheffi[1)]) is used for route choice. Some concepts that are used in the route choice include:

- $f_{\tau,k}^{rs}$: The "reference" route flow on the route k between OD pair rs in period τ. This concept is distinguished from the concept of route flow in the STA model. This reference route flow represents the amount of the flow which departs from the origin node r to the destination node s in period τ choosing route k. While all the route flow in the STA model can arrive at the destination node within period τ, some part of $f_{\tau,k}^{rs}$ cannot reach the destination node s within this period (called residual flows) and residual flow of this reference flow may choose another route in the next period. So, this reference route flow does not necessarily represent the route flow in period τ.
- Q_{τ}^{rs}: The travel demand of OD pair rs in period τ. It reflects the total number of users demanding to go from origin node r to s in period τ. According to the assumption 5, the residual flows in the current period are considered as the travel demand of the next period. So, Q_{τ}^{rs} includes travel demand generated in period τ and residual flows in period $\tau - 1$.
- $p_{\tau,k}^{rs}$: The probability of choosing the route k between OD pair rs in period τ. It is the ratio of the number of users at origin node r choose route k to go to destination node s over the travel demand of OD pair rs in period τ.

Applying logit-based model (see Chapter 2), the reference route flow in period τ is calculated by:

$$f_{\tau,k}^{rs} = Q_{\tau}^{rs} p_{\tau,k}^{rs} = Q_{\tau}^{rs} \frac{\exp(-\theta c_{\tau,k}^{rs})}{\sum_{k \epsilon K^{rs}} \exp(-\theta c_{\tau,k}^{rs})} \tag{4.1}$$

where θ is a positive parameter that depicts the perceived travel time, $c_{\tau,k}^{rs}$ denotes the travel time on route k of OD pair rs in period τ and $K^{rs} = \{ k, l, ... \}$ is the set of routes connecting OD pair rs.

Consequently, the reference traffic flow on link ij in period τ is denoted by $x_{\tau,ij}$. This reference link flow is the sum of all the reference route flow of all OD pairs that are expected to use this link in period τ. In other words, it represents the amount of the flow that is expected to choose link ij in period τ. This reference traffic flow does not reflect the flow on the link ij because some parts of $f_{\tau,k}^{rs}$, residual flows, still travel on the previous links of link ij and cannot reach link ij in period τ (called eliminated flow). Using the link-route incidence variable, the reference traffic flow on link ij is given as the following equation:

$$x_{\tau,ij} = \sum_{rs \in W_\tau} \sum_{k \in K^{rs}} f_{\tau,k}^{rs} \delta_{ij,k}^{rs}, \tag{4.2}$$

where $W_\tau = \{12, \dots, rs, \dots, uv\}$ is the set of OD pairs in period τ, $\delta_{ij,k}^{rs}$ is the link-route incidence variable (if the route k includes link ij then $\delta_{ij,k}^{rs} = 1$, otherwise $\delta_{ij,k}^{rs} = 0$). The reference link flow vector is given by

$$\mathbf{x}_\tau = \Delta \mathbf{f}_\tau, \tag{4.3}$$

where $\mathbf{x}_\tau = (x_{\tau,12}, \dots, x_{\tau,ij}, \dots, x_{\tau,mn})^T$ and $\mathbf{f}_\tau = (f_{\tau,1}^{rs}, \dots, f_{\tau,|K^{rs}|}^{rs}, \dots, f_{\tau,|K^{uv}|}^{uv})^T$ are the vector form of reference link flow and reference route flow in period τ, respectively. Besides, $\Delta\left(= \left\{\delta_{ij,k}^{rs}\right\}\right)$ is the link-route incidence matrix and ^{T} is the transpose.

Also, the route travel time on route k in period τ is denoted by $c_{\tau,k}^{rs}$. It is the sum of the link travel times in this period of the links that are parts of route k in period τ. The route travel time and the vector form of it are given as:

$$c_{\tau,k}^{rs} = \sum_{ij \in A} t_{\tau,ij} \delta_{ij,k}^{rs}, \tag{4.4}$$

$$\mathbf{c}_\tau = \Delta^T \mathbf{t}_\tau, \tag{4.5}$$

where $t_{\tau,ij}$ is the travel time on link ij in period τ. In addition, $\mathbf{c}_\tau = (c_{\tau,1}^{rs}, \dots, c_{\tau,|K^{rs}|}^{rs}, \dots, c_{\tau,|K^{uv}|}^{uv})^T$ and $\mathbf{t}_\tau = (t_{\tau,12}, \dots, t_{\tau,ij}, \dots, t_{\tau,mn})^T$ are the vector form of route and link travel times, respectively.

As mentioned above, flow propagation is considered in the semi-DTA model.

The next is how to compute the residual flows for flow propagation. Let $y_{\tau,k,ij}^{rs}$ denotes the residual flow on link ij of route k of OD pair rs within period τ. It is a part of $f_{\tau,k}^{rs}$ cannot reach the destination node s and still travel on link ij of route k within period τ. According to assumption 2, the travel demand departs from the origin node r to the destination node s continuously at the same rate $\frac{Q_\tau^{rs}}{L}$. Thus, the reference route flow, $f_{\tau,k}^{rs}$, also departs from the origin node r to the destination node s at the same rate $\frac{f_{\tau,k}^{rs}}{L}$ on route k. The parts of $f_{\tau,k}^{rs}$ that departs first may reach the destination within period τ. But some parts of $f_{\tau,k}^{rs}$ that departs later may not be able to reach the destination the period τ and are still moving on the links that are parts of route k at the end of the considering period, called residual link flows. For example, the following figure shows the residual flow on the links belong to route k within period τ

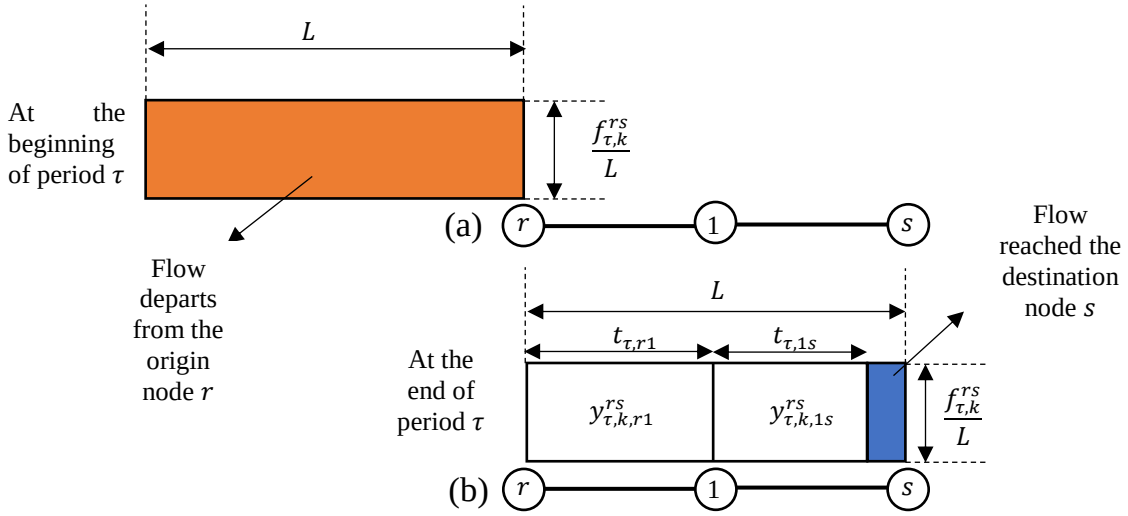


Figure 4.3 Residual flow on the links belong to route k within period τ

Figure 4.3 depicts how to calculate residual flow. At the beginning of period τ, the reference flow $f_{\tau,k}^{rs}$ departs at the rate $\frac{f_{\tau,k}^{rs}}{L}$ from origin node r. This rate is maintained until the end of the period τ with period length L. At the end of this period, some flows reach the destination node s. In addition, the remaining flows of $f_{\tau,k}^{rs}$ are still moving on the links of route k from node r to node s because they depart later and do not have enough time to reach the destination node s.

Accordingly, the residual flow on link ij of route k of OD pair rs in period τ is calculated by the product of $\frac{f_{\tau,k}^{rs}}{L}$ and $t_{\tau,ij}$ as follows:

$$y_{\tau,k,ij}^{rs} = \frac{f_{\tau,k}^{rs}\delta_{ij,k}^{rs}}{L} t_{\tau,ij}. \quad (4.6)$$

Also, the summation of residual flow is given by:

$$y_{\tau,ij}^{rs} = \sum_{k \epsilon K^{rs}} y_{\tau,k,ij}^{rs} = \sum_{k \epsilon K^{rs}} \frac{f_{\tau,k}^{rs}\delta_{ij,k}^{rs}}{L} t_{\tau,ij}, \quad (4.7)$$

where $y_{\tau,ij}^{rs}$ denotes the residual flow on link ij of OD pair rs in period τ. It is the summation of the residual flow on this link of all route k of OD pair rs. This residual flow is added to the travel demand from node j to original destination node s in period $\tau + 1$.

Because residual flow on a link of route k of OD pairs rs can not exit this link in period τ, it can not travel on the subsequent links of this link on the route k of OD pair rs in period τ. The eliminated flow of link ij of each route k of OD pair rs in period τ is denoted by $s_{\tau,k,ij}^{rs}$. The eliminated flow of a link reflects the flow that can not travel on this link. In other words, $s_{\tau,k,ij}^{rs}$ denotes the sum of the residual flows of upstream links of link ij of route k of OD pair rs in period τ. With each route, we assume that $H_{k,ij}^{rs}$ denotes the set of upstream links of link ij of route k of OD pair rs. The eliminated flow of each link ij is calculated by:

$$s_{\tau,k,ij}^{rs} = \sum_{gh \epsilon H_{k,ij}^{rs}} y_{\tau,k,gh}^{rs}, \quad (4.8)$$

where $s_{\tau,k,ij}^{rs}$ is the eliminated flow of link ij of each route k of OD pair rs in period τ. Also, the summation of the eliminated flow of link ij is given as:

$$s_{\tau,ij} = \sum_{rs \epsilon W_\tau} \sum_{k \epsilon K^{rs}} s_{\tau,k,ij}^{rs}, \quad (4.9)$$

where $s_{\tau,ij}$ is the summation of the eliminated flow of link ij for all routes and all OD pairs in period τ. The eliminated flow on a link is the summation of residual flows on previous links of this link and does not include the residual flow of this link.

The link flow after the elimination of residual flow (we will call this "adjusted link flow"), $z_{\tau,ij}$, is given by the following vector form:

$$\mathbf{z}_\tau = \mathbf{x}_\tau - \mathbf{s}_\tau, \tag{4.10}$$

where $\mathbf{z}_\tau = (z_{\tau,12}, \dots, z_{\tau,ij}, \dots, z_{\tau,mn})^T$ and $\mathbf{s}_\tau = (s_{\tau,12}, \dots, s_{\tau,ij}, \dots, s_{\tau,mn})^T$ are the vector of adjusted link flow and eliminated link flow in period τ, respectively. Among flows expected to choose link ij to travel in period τ, $x_{\tau,ij}$, some flows can not arrive this link in this period, $s_{\tau,ij}$. Thus, $z_{\tau,ij} = x_{\tau,ij} - s_{\tau,ij}$, is also the inflow to link ij in period τ. It reflects the total number of traffic flow that can reach link ij in period τ. From **Equations (4.6)-(4.8)**, $\mathbf{s}_\tau$ is the function of $\mathbf{f}_\tau$ and $\mathbf{t}_\tau$, $\mathbf{s}_\tau = \mathbf{s}_\tau(\mathbf{f}_\tau, \mathbf{t}_\tau)$. Thus,

$$\mathbf{z}_\tau = \Delta\mathbf{f}_\tau - \mathbf{s}_\tau(\mathbf{f}_\tau, \mathbf{t}_\tau), \tag{4.11}$$

where $\mathbf{s}_\tau(.)$ is the vector-valued function of eliminated link flows in period τ.

According to assumption 3, the travel time on a link is assumed as a continuous and non-decreasing function of its inflow. Thus, the link travel time on link ij is the function of $z_{\tau,ij}$, $t_{\tau,ij}(z_{\tau,ij})$. The problem is that $z_{\tau,ij}$ is also the function of the link travel time and reference route flow according to **Equation (4.11)**. The link travel time itself is needed to determine the link travel time. Thus, the fixed-point problem of link travel time is given by

$$\mathbf{t}_\tau = \mathbf{t}_\tau(\Delta\mathbf{f}_\tau - \mathbf{s}_\tau[\mathbf{f}_\tau, \mathbf{t}_\tau]), \tag{4.12}$$

where $\mathbf{t}_\tau(.)$ denotes the vector-valued function of link travel times in period τ.

From the above equations, the reference route flow in each period is calculated by the following double-looped fixed-point problem

$$\mathbf{f}_\tau = \mathbf{Q}_\tau\mathbf{p}_\tau(\Delta^T\mathbf{t}_\tau(\Delta\mathbf{f}_\tau - \mathbf{s}_\tau[\mathbf{f}_\tau, \mathbf{t}_\tau])), \tag{4.13}$$

where $\mathbf{Q}_\tau$ is the diagonal matrix of all travel demands in period τ and $\mathbf{p}_\tau = (p_{\tau,1}^{rs}, \dots, p_{\tau,|K^{rs}|}^{rs}, \dots, p_{\tau,|K^{uv}|}^{uv})^T$ is the vector of all route probabilities in period τ. The formulation of $\mathbf{Q}_\tau$ can be expanded as below diagonal matrices:

$$\mathbf{Q}_\tau^{rs} = \begin{pmatrix} Q_\tau^{rs} & & 0 \\ & \ddots & \\ 0 & & Q_\tau^{rs} \end{pmatrix}, \tag{4.14}$$

$$\mathbf{Q}_\tau = \begin{pmatrix} \ddots & & 0 \\ & \mathbf{Q}_\tau^{rs} & \\ 0 & & \ddots \end{pmatrix}. \quad (4.15)$$

Equation (4.15) shows the semi-DTA SUE model in this study. To achieve equilibrium in this model, the double-looped fixed-point problem must to solved. One is the fixed-point problem for link travel times and the other is the fixed-point problem for reference route flows. The link travel times depend on the reference route flows and the link travel times. Also, the reference route flows depend on the reference route flows and the link travel times. As a consequence, a large amount of calculation time will be wasted for equilibrium solution, and applying to the real road network is very difficult.

Robbins and Monro[2] introduced MSA for solving the fixed-point problem $\boldsymbol{F}(\mathbf{X}) = \mathbf{X}$. Based on the current solution $\mathbf{X}^{(l)}$, the general MSA procedure includes two important steps: after finding an auxiliary point by $\mathbf{YX}^{(l)} = \boldsymbol{F}(\mathbf{X}^{(l)})$, the solution is updated by:

$$\mathbf{X}^{(l+1)} = \mathbf{X}^{(l)} + \lambda^{(l)}\left(\mathbf{YX}^{(l)} - \mathbf{X}^{(l)}\right),\ l = 1;\ 2;\ \ldots \quad (4.16)$$

here, l is the current calculation loop and $\lambda^{(l)}$ is the step length taken along the search direction $\left(\mathbf{YX}^{(l)} - \mathbf{X}^{(l)}\right)$. Robbins and Monro[2] and Blum[3] proved that the solution of the fixed-point problem converged under the conditions of $\sum \lambda^{(l)} = \infty$ and $\sum\left(\lambda^{(l)}\right)^2 < \infty$. In a well-known MSA, the predetermined step-size sequence has been used to guarantee the convergence (Maher[4]). We can detail the algorithm in each period τ for solving the double-looped fixed-point problem in **Equation (4.15)** with the MSA as following:

Step 1: Initialization

Set iteration $m = 1$. Set an initial solution of $\left\{f_{\tau,k}^{rs(m)}\right\}$.

Step 2: Solving the fixed-point problem of link travel time

(a) Set iteration $l = 1$. Set link travel time initial solution ${t_{\tau,ij}}^{(l)} = t_{ij}^0$

(b) Based on $\left\{f_{\tau,k}^{rs(m)}\right\}$, $\left\{{t_{\tau,ij}}^{(l)}\right\}$, calculating $\left\{{x_{\tau,ij}}^{(m)}\right\}$, $\left\{{y_{\tau,k,ij}^{rs}}^{(l)}\right\}$, $\left\{{s_{\tau,k,ij}^{rs}}^{(l)}\right\}$, $\left\{{s_{\tau,ij}}^{(l)}\right\}$, $\left\{{z_{\tau,ij}}^{(l)}\right\}$.

(c) Calculating auxiliary link travel time, $\{yt_{\tau,ij}{}^{(l)}\}$.

(d) Update link travel time solution as follows:

$$\mathbf{t}_\tau{}^{(l+1)} = \mathbf{t}_\tau{}^{(l)} + \lambda^{(l)}\left(\mathbf{yt}_\tau{}^{(l)} - \mathbf{t}_\tau{}^{(l)}\right). \quad (4.17)$$

where $\lambda_1{}^{(l)}$ is the step length at iteration l $(0 < \lambda_1{}^{(l)} < 1)$.

(e) Convergence verification: If $\left\|\mathbf{yt}_\tau{}^{(l+1)} - \mathbf{t}_\tau{}^{(l)}\right\| \le \sigma$, go to step 3; otherwise, $l = l + 1$ and come back substep b, where σ represents the convergence level.

Step 3: Based on $\{f_{\tau,k}^{rs(m)}\}$, $\{s_{\tau,k,ij}^{rs}{}^{(l)}\}$, $\{t_{\tau,ij}{}^{(l)}\}$ and $\{z_{\tau,ij}{}^{(l)}\}$, calculating route travel time $\{c_{\tau,k}^{rs}{}^{(m)}\}$ and auxiliary reference route traffic flow $\{\mathrm{y}f_{\tau,k,ij}^{rs}{}^{(m)}\}$ based on $\{c_{\tau,k}^{rs}{}^{(m)}\}$.

Step 4: Calculate auxiliary reference route traffic flow $\left\{\mathrm{y}f_{\tau,k_{\tau,ij}}^{rs}{}^{(m)}\right\}$ based on $\{c_{\tau,k}^{rs}{}^{(m)}\}$ and update reference route traffic flow solution as follows:

$$\mathbf{f}_\tau{}^{(m+1)} = \mathbf{f}_\tau{}^{(m)} + \lambda^{(m)}\left(\mathbf{yf}_\tau{}^{(m)} - \mathbf{f}_\tau{}^{(m)}\right). \quad (4.18)$$

where $\lambda_2{}^{(m)}$ is the step length at iteration m $(0 < \lambda_2{}^{(m)} < 1)$.

Step 5: Convergence verification: If $\left\|\mathbf{yf}_\tau{}^{(m+1)} - \mathbf{f}_\tau{}^{(m)}\right\| \le \sigma$, go to step 6 and $\{f_{\tau,k}^{rs(m)}\}$, $\{y_{\tau,k,ij}^{rs}{}^{(l)}\}$, $\{s_{\tau,k,ij}^{rs}{}^{(l)}\}$, $\{t_{\tau,ij}{}^{(l)}\}$ and $\{z_{\tau,ij}{}^{(l)}\}$ are solutions in period τ; otherwise, $m = m + 1$ and go to step 2.

Step 6: Flow propagation: Propagate residual flow $\{y_{\tau,k,ij}^{rs}{}^{(l)}\}$ to the next period as travel demand from node *j* to node s.

Handling the double-looped fixed-point problem triggers the calculational time problem and weakens the applicability of the semi-DTA model into a real road network. In this study, therefore, the sensitivity analysis method will be used to reduce computational cost and increase the applicability of the model. We think of the results from the static SUE model to approximate the semi-DTA model. Especially the use of link-based approaches, such as the STOCH3 algorithm and DFS loading procedure proposed in Chapter 3, is a remarkably effective approach both in computational time and memory consumption.

4.4 Sensitivity analysis method for solving semi-dynamic stochastic user equilibrium model

4.4.1 Route-based approach

Due to the influence of the residual traffic volume, in general, the solution for **Equation (4.13)** cannot be formulated as an optimization problem. In the previous researches, Ha et al.[5] and Itagaki et al.[6] considered how to obtain a unique approximate solution. They assumed that residual flow for flow propagation and eliminated flow of links are calculated by using the results of the static SUE solution.

The issue to consider is that if the flow is eliminated from the links, not only the link travel time changed but also the inflow changed via network equilibrium. To solve this problem, the sensitivity analysis is applied and eliminated flow is considered as a perturbation parameter. By using sensitivity analysis, the ratios change of route flow respect to perturbation of eliminated link flow is computed, i.e. the corresponding partial derivatives at SUE state need to be calculated. However, Ha et al.[5] and Itagaki et al.[6] solved these by route-based approaches.

The proposed method for semi-DTA SUE model using sensitivity analysis method with route-based approach includes the following four steps for each period:

Step 1: Calculate a static SUE in a given period with the OD demand matrix in this period by using the MSA method with the route-based approach.

The results are as follows: $\mathbf{f}_0 = (f_{0,1}^{rs}, \dots, f_{0,|K^{rs}|}^{rs}, \dots, f_{0,|K^{uv}|}^{uv})^T$, $\mathbf{p}_0 = (p_{0,1}^{rs}, \dots, p_{0,|K^{rs}|}^{rs}, \dots, p_{0,|K^{uv}|}^{uv})^T$, $\mathbf{c}_0 = (c_{0,1}^{rs}, \dots, c_{0,|K^{rs}|}^{rs}, \dots, c_{0,|K^{uv}|}^{uv})^T$, $\mathbf{x}_0 = (x_{0,12}, \dots, x_{0,ij}, \dots, x_{0,mn})^T$ and $\mathbf{t}_0 = (t_{0,12}, \dots, t_{0,ij}, \dots, t_{0,mn})^T$ are the vector of route traffic flows, probability of choosing routes, route travel times, link traffic flows, link travel times at the static SUE, respectively.

Step 2: Based on the results at the static SUE, calculate residual flow for flow propagation, eliminated flow, and obtain the reference route flows with sensitivity analysis method.

Residual link flows are calculated by

$$y_{\tau,ij}^{rs} = \sum_{k \epsilon K^{rs}} \frac{f_{0,k}^{rs} \delta_{ij,k}^{rs}}{L} t_{0,ij}, \quad (4.19)$$

where $y_{\tau,ij}^{rs}$ is the residual link flow of link ij in the current period that is propagated to the next period.

Eliminated link flows are calculated by

$$s_{\tau,ij} = \sum_{rs \in W_\tau} \sum_{k \in K^{rs}} \sum_{gh \epsilon H_{k,ij}^{rs}} \frac{f_{0,k}^{rs} \delta_{gh,k}^{rs}}{L} t_{0,gh}, \quad (4.20)$$

where $s_{\tau,ij}$ is the eliminated link flow of link ij that is removed from link ij in the current period. If eliminated link flow is calculated and fixed, the problem of the semi-DTA SUE model becomes solving below fixed-point problem with reference route flow

$$\mathbf{f}_\tau = \mathbf{Q}_\tau \mathbf{p}_\tau \left(\Delta^T \mathbf{t}_\tau (\Delta \mathbf{f}_\tau - \mathbf{s}_\tau) \right), \quad (4.21)$$

where $\mathbf{s}_\tau = (s_{\tau,12}, \dots, s_{\tau,ij}, \dots, s_{\tau,mn})^T$ is the vector of eliminated link flows.

For solving the above problem, the sensitivity analysis method and the results at the static SUE model are used. Because the SUE solution has been achieved in the static traffic assignment model, the SUE solution in the semi-DTA model will be approximated based on the equilibrium results of the static model. Assuming that $\mathbf{s}_0 = (s_{0,12}, \dots, s_{0,ij}, \dots, s_{0,mn})^T$ is the vector of eliminated link flow at the static SUE model. The static SUE problem using the OD demand matrix in period τ is the following fixed-point problem with route traffic flow:

$$\mathbf{f}_0 = \mathbf{Q}_\tau \mathbf{p}_0 (\Delta^T \mathbf{t}_0 (\Delta \mathbf{f}_0 - \mathbf{s}_0)), \quad (4.22)$$

There are no eliminated link flows at the static SUE model ($\mathbf{s}_0 = 0$). When eliminated link flow is calculated and removed from the links, the link travel time changes and the equilibrium solution also changes.

Firstly, sensitivity analysis with the route-based approach is applied to calculate the partial derivative matrix of the route flow with respect to eliminated link flow at the static SUE

$$\boldsymbol{\nabla_s \mathbf{f}_0} = -\frac{1}{L}\left[(\mathbf{I} - \mathbf{Q}_\tau\, \boldsymbol{\nabla_c \mathbf{p}_0}\, \Delta^T\, \boldsymbol{\nabla_x \mathbf{t}_0}\, \Delta)^{-1} \mathbf{Q}_\tau\, \boldsymbol{\nabla_c}\, \mathbf{p}_0\, \Delta^T\, \boldsymbol{\nabla_x \mathbf{t}_0}\right] \quad (4.23)$$

where $\boldsymbol{\nabla_s \mathbf{f}_0}$ is the partial derivative matrix of route flow with respect to eliminated link flow, $\boldsymbol{\nabla_c \mathbf{p}_0}$ is the partial derivative matrix of route choice probability with respect to route travel time, $\boldsymbol{\nabla_x \mathbf{t}_0}$ is the partial derivative matrix of link travel time with respect to link traffic flow at the static SUE model

$$\boldsymbol{\nabla_s \mathbf{f}_0} = \begin{pmatrix} \frac{\partial f_{0,1}^{rs}}{\partial s_{0,12}} & \cdots & \frac{\partial f_{0,1}^{rs}}{\partial s_{0,ij}} & \cdots & \frac{\partial f_{0,1}^{rs}}{\partial s_{0,mn}} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \frac{\partial f_{0,|K^{rs}|}^{rs}}{\partial s_{0,12}} & \cdots & \frac{\partial f_{0,|K^{rs}|}^{rs}}{\partial s_{0,ij}} & \cdots & \frac{\partial f_{0,|K^{rs}|}^{rs}}{\partial s_{0,mn}} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f_{0,|K^{uv}|}^{uv}}{\partial s_{0,12}} & \cdots & \frac{\partial f_{0,|K^{uv}|}^{uv}}{\partial s_{0,ij}} & \cdots & \frac{\partial f_{0,|K^{uv}|}^{uv}}{\partial s_{0,mn}} \end{pmatrix} \quad (4.24)$$

$$\boldsymbol{\nabla_c \mathbf{p}_0} = \begin{pmatrix} \frac{\partial p_{0,1}^{rs}}{\partial c_{0,1}^{rs}} & \cdots & \frac{\partial p_{0,1}^{rs}}{\partial c_{0,|K^{rs}|}^{rs}} & \cdots & \frac{\partial p_{0,1}^{rs}}{\partial c_{0,|K^{uv}|}^{uv}} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \frac{\partial p_{0,|K^{rs}|}^{rs}}{\partial c_{0,1}^{rs}} & \cdots & \frac{\partial p_{0,|K^{rs}|}^{rs}}{\partial c_{0,|K^{rs}|}^{rs}} & \cdots & \frac{\partial p_{0,|K^{rs}|}^{rs}}{\partial c_{0,|K^{uv}|}^{uv}} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial p_{0,|K^{uv}|}^{uv}}{\partial c_{0,1}^{rs}} & \cdots & \frac{\partial p_{0,|K^{uv}|}^{uv}}{\partial c_{0,|K^{rs}|}^{rs}} & \cdots & \frac{\partial p_{0,|K^{uv}|}^{uv}}{\partial c_{0,|K^{uv}|}^{uv}} \end{pmatrix} \quad (4.25)$$

$$\boldsymbol{\nabla_x \mathbf{t}_0} = \begin{pmatrix} \frac{\partial t_{0,12}}{\partial x_{0,12}} & \cdots & \frac{\partial t_{0,12}}{\partial x_{0,ij}} & \cdots & \frac{\partial t_{0,12}}{\partial x_{0,mn}} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \frac{\partial t_{0,ij}}{\partial x_{0,12}} & \cdots & \frac{\partial t_{0,ij}}{\partial x_{0,ij}} & \cdots & \frac{\partial t_{0,ij}}{\partial x_{0,mn}} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial t_{0,mn}}{\partial x_{0,12}} & \cdots & \frac{\partial t_{0,mn}}{\partial x_{0,ij}} & \cdots & \frac{\partial t_{0,mn}}{\partial x_{0,mn}} \end{pmatrix} \quad (4.26)$$

Secondly, the influence of the eliminated link flow to the solution of

route flow could be computed by using the result of the sensitivity analysis, and the new static SUE solution with new route flow is approximated. The route flow at the new static SUE state is the reference route flow at the semi-DTA model because the semi-DTA model is the static SUE model with $\mathbf{s}_0 = \mathbf{s}_\tau$ according to **Equations (4.21) and (4.22)**. As a result, the reference route flow influenced by the elimination of residual flow in each period is estimated as follows:

$$\begin{aligned}\mathbf{f}_\tau &= \mathbf{f}_0 + \boldsymbol{\nabla}_\mathbf{s}\mathbf{f}_0\,\mathbf{s}_\tau \\ &= \mathbf{f}_0 - \frac{1}{L}\left[(\mathbf{I} - \mathbf{Q}_\tau\,\boldsymbol{\nabla}_\mathbf{c}\mathbf{p}_0\,\Delta^T\,\boldsymbol{\nabla}_\mathbf{x}\mathbf{t}_0\,\Delta)^{-1}\mathbf{Q}\,\boldsymbol{\nabla}_\mathbf{c}\,\mathbf{p}_0\,\Delta^T\,\boldsymbol{\nabla}_\mathbf{x}\mathbf{t}_0\right]\mathbf{s}_\tau\end{aligned} \tag{4.27}$$

Step 3: Recalculate a static equilibrium assignment by subtracting the propagated flow from the static equilibrium assignment.

Based on the results of the reference route flows in ***Step 2***, the reference link flow, $\mathbf{x}_\tau$, is calculated by using **Equation (4.3)**. Besides, the solution of adjusted link flow in period τ is calculated by

$$\mathbf{z}_\tau = \mathbf{x}_\tau - \mathbf{s}_\tau \tag{4.28}$$

Step 4: Deliver the propagated flow to the next period.

$\left\{y_{\tau,ij}^{rs}\right\}$ is delivered to the next period as the travel demand from node j to node s in the next period. The above calculation process is repeated for the next period.

The more details could be referred to as Itagaki et al.[6]. The above method needs to calculate the static SUE by the route-based approach and explicit route-set that consumes memory and computation time. Besides, by cause of the existing inverse route-based matrix in the formulation, applying the proposed approach to large-scale networks is very difficult. If the residual flow on a link is eliminated and propagated to the next period as travel demand from end node of this link to the original destination of this flow (***Step 4***), the number of OD pairs will increase dramatically. Therefore the number of routes will escalate greatly and the computer may be overloaded. To update this method, increase the applicability and reduce the computational cost, hence, we will propose a link-based approach with

STOCH3 algorithm (Leurent[7]).

4.4.2 Link-based STOCH3 approach

a. The first algorithm

From the viewpoint of practical use, we will first consider how to obtain a unique approximate solution and prevent route enumeration to achieve efficient calculation in our research.

With the same approach as the above route-based, we need to have the result of the static SUE using the semi-DTA OD matrix at the first step. Because it is impossible to enumerate all routes between each OD pair, applying the static SUE to a large road network needs to create a route-set for route choice that consumes computer memory. To solve this problem, the STOCH3-efficient route definition is used (see Chapters 2 and 3). We will calculate the SUE solution for efficient links on the road network without wasting memory to store the route variables. Let we denote as $\mathbf{x}_0$ and $\mathbf{t}_0$ are the vector of link flow and link travel time of the STA SUE model that can be calculated by running the DFS loading procedure using STOCH3-efficient route definition or link-based STOCH3 algorithm with the MSA until reaching equilibrium solution (see Chapter 3).

Consequently, we need to calculate $y_{\tau,ij}^{rs}$ and $s_{\tau,ij}$ with only link-based and node-based variables.

As introduced in Chapter 3, $x_{0,ij}^{rs}$ is the number of flows from r to s that choose some routes containing link ij, $x_{0,ij\to gh}^{rs}$ is the number of flows from r to s choosing some routes which consist of both links ij and gh satisfying link ij is travelled prior to link gh at static SUE. From the definition of $x_{0,ij}^{rs}$ and $x_{0,ij\to gh}^{rs}$, we have

$$x_{0,ij}^{rs} = \sum_{k\epsilon K^{rs}} f_{0,k}^{rs}\delta_{ij,k}^{rs}, \tag{4.29}$$

$$x_{0,ij\to gh}^{rs} = \sum_{k\in K_{ij\to gh}^{rs}} f_{0,k}^{rs}, \tag{4.30}$$

where $K_{ij\to gh}^{rs}$ denotes the set of efficient routes of OD pair rs in which both links ij and gh are used and link ij is used before link gh. Denoting $s_{\tau,gh,ij}^{rs}$ denotes the

eliminated flow of link gh caused by the residual flow of link ij of OD pair rs in period τ. If route $k \in K_{ij\to gh}^{rs}$, the residual flow of link ij on the route k of OD pair rs, $y_{0,k,ij}^{rs}$, cannot reach link gh. So,

$$\begin{aligned} s_{\tau,gh,ij}^{rs} &= \sum_{k\in K_{ij\to gh}^{rs}} y_{0,k,ij}^{rs} = \sum_{k\in K_{ij\to gh}^{rs}} \frac{f_{0,k}^{rs}}{L}\delta_{ij,k}^{rs} t_{0,ij} \\ &= \frac{t_{0,ij}}{L}\sum_{k\in K_{ij\to gh}^{rs}} f_{0,k}^{rs} = \frac{t_{0,ij}}{L} x_{0,ij\to gh}^{rs} \end{aligned} \tag{4.31}$$

Therefore, the residual flow for flow propagation and eliminated flow are calculated depending on $x_{0,ij}^{rs}$ and $x_{0,ij\to gh}^{rs}$ as follows:

$$y_{\tau,ij}^{rs} = \sum_{k\in K^{rs}} \frac{f_{0,k}^{rs}\delta_{ij,k}^{rs}}{L} t_{0,ij} = \frac{t_{0,ij}}{L}\sum_{k\in K^{rs}} f_{0,k}^{rs}\delta_{ij,k}^{rs} = \frac{t_{0,ij}}{L} x_{0,ij}^{rs}, \tag{4.32}$$

$$s_{\tau,gh} = \sum_{rs\in W_\tau} s_{\tau,gh}^{rs} = \sum_{rs\in W_\tau}\sum_{ij\in A} s_{\tau,gh,ij}^{rs} = \sum_{rs\in W_\tau}\sum_{ij\in A} \frac{x_{0,ij\to gh}^{rs}}{L} t_{0,ij}. \tag{4.33}$$

where $s_{\tau,gh}^{rs}$ is the eliminated flow of link gh of OD pair rs in period τ.

As shown in Chapter 3, $x_{0,ij}^{rs}$ and $x_{0,ij\to gh}^{rs}$ could be computed by running the DFS loading procedure or the STOCH3 algorithm once for each OD pair based on the link travel times, $\{t_{0,ij}\}$, achieved in ***Step 1*** (see pages 85 and 87 in Chapter 3). Correspondingly, we can calculate residual flow for flow propagation, $y_{\tau,ij}^{rs}$ and eliminated flow, $s_{\tau,gh}$, at the static SUE by using only link-based and node-based variables.

Equation (4.21) depicted the fixed-point problem of the semi-DTA SUE with reference route flow when the residual flow and the eliminated flow were fixed. Based on the relationship between reference route flow and reference link flow shown in **Equation (4.3)**, the problem of **Equation (4.21)** becomes the following fixed-point problem with reference link flow

$$\mathbf{x}_\tau = \mathbf{Q}_\tau \mathbf{p}_\tau(\Delta^T \mathbf{t}_\tau(\mathbf{x}_\tau - \mathbf{s}_\tau)). \tag{4.34}$$

Like the route-based approach, we can rely on the results of $\mathbf{x}_0$ and $\mathbf{t}_0$ in ***Step 1*** to approximate the result of $\mathbf{x}_\tau$. However, the problem here is calculating partial derivative matrix of the link flow with respect to eliminated link flow at the static

SUE, $\boldsymbol{\nabla_s \mathbf{x}_0}$:

$$\boldsymbol{\nabla_s \mathbf{x}_0} = \begin{pmatrix} \frac{\partial x_{0,12}}{\partial s_{0,12}} & \cdots & \frac{\partial x_{0,12}}{\partial s_{0,ij}} & \cdots & \frac{\partial x_{0,12}}{\partial s_{0,mn}} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \frac{\partial x_{0,ij}}{\partial s_{0,12}} & \cdots & \frac{\partial x_{0,ij}}{\partial s_{0,ij}} & \cdots & \frac{\partial x_{0,ij}}{\partial s_{0,mn}} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial x_{0,mn}}{\partial s_{0,12}} & \cdots & \frac{\partial x_{0,mn}}{\partial s_{0,ij}} & \cdots & \frac{\partial x_{0,mn}}{\partial s_{0,mn}} \end{pmatrix} \tag{4.35}$$

Note that there is no eliminated link flow at the static SUE model. Thus, the problem of the static SUE model is the following fixed-point problem with the link flow in the case $\mathbf{s}_0 = \mathbf{0}$:

$$x_{0,ij} = \sum_{rs \in W_\tau} Q_\tau^{rs} \frac{\sum_{k \epsilon K^{rs}} \delta_{ij,k}^{rs} \exp\left(-\theta c_{0,k}^{rs}\left(\mathbf{t}_0(\mathbf{x}_0, \mathbf{s}_0)\right)\right)}{\sum_{k \epsilon K^{rs}} \exp\left(-\theta c_{0,k}^{rs}\left(\mathbf{t}_0(\mathbf{x}_0, \mathbf{s}_0)\right)\right)}. \tag{4.36}$$

Denoting the right-hand side of the above equation by $g_{0,ij}$,

$$g_{0,ij} = \sum_{rs \in W_\tau} Q_\tau^{rs} \frac{\sum_{k \epsilon K^{rs}} \delta_{ij,k}^{rs} \exp\left(-\theta c_{0,k}^{rs}\left(\mathbf{t}_0(\mathbf{x}_0, \mathbf{s}_0)\right)\right)}{\sum_{k \epsilon K^{rs}} \exp\left(-\theta c_{0,k}^{rs}\left(\mathbf{t}_0(\mathbf{x}_0, \mathbf{s}_0)\right)\right)} \tag{4.37}$$

$\mathbf{g}_0 = (g_{0,12}, \dots, g_{0,ij}, g_{0,gh} \dots, g_{0,mn})^T$ is the vector form of $g_{0,ij}$. Owing to $g_{0,ij}$ is the function of two variables $\mathbf{x}_0, \mathbf{s}_0$, the following equation, $\mathbf{d}_0 = (d_{0,12}, \dots, d_{0,ij}, d_{0,gh} \dots, d_{0,mn})^T$ is also defined as a function with two variables $\mathbf{x}_0, \mathbf{s}_0$:

$$\mathbf{d}_0(\mathbf{x}_0, \mathbf{s}_0) = \mathbf{x}_0 - \mathbf{g}_0\left(\mathbf{t}_0(\mathbf{x}_0, \mathbf{s}_0)\right). \tag{4.38}$$

Accordingly, $\mathbf{d}_0(\mathbf{x}_0, \mathbf{s}_0) = \mathbf{0}$ indicates the static SUE state. Denoting $\boldsymbol{\nabla_t \mathbf{d}_0}$, $\boldsymbol{\nabla_x \mathbf{d}_0}$ and $\boldsymbol{\nabla_s \mathbf{d}_0}$ are the matrices of partial derivatives of $d_{0,ij}$ with respect to $t_{0,ij}$, $x_{0,ij}$, and $s_{0,ij}$ at the static SUE in which:

$$\nabla_{\mathbf{s}}\mathbf{d}_0 = \begin{pmatrix} \frac{\partial d_{0,12}}{\partial t_{0,12}} & \cdots & \frac{\partial d_{0,12}}{\partial t_{0,ij}} & \cdots & \frac{\partial d_{0,12}}{\partial t_{0,mn}} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \frac{\partial d_{0,ij}}{\partial t_{0,12}} & \cdots & \frac{\partial d_{0,ij}}{\partial t_{0,ij}} & \cdots & \frac{\partial d_{0,ij}}{\partial t_{0,mn}} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial d_{0,mn}}{\partial t_{0,12}} & \cdots & \frac{\partial d_{0,mn}}{\partial t_{0,ij}} & \cdots & \frac{\partial d_{0,mn}}{\partial t_{0,mn}} \end{pmatrix} \tag{4.39}$$

$$\nabla_{\mathbf{x}}\mathbf{d}_0 = \begin{pmatrix} \frac{\partial d_{0,12}}{\partial x_{0,12}} & \cdots & \frac{\partial d_{0,12}}{\partial x_{0,ij}} & \cdots & \frac{\partial d_{0,12}}{\partial x_{0,mn}} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \frac{\partial d_{0,ij}}{\partial x_{0,12}} & \cdots & \frac{\partial d_{0,ij}}{\partial x_{0,ij}} & \cdots & \frac{\partial d_{0,ij}}{\partial x_{0,mn}} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial d_{0,mn}}{\partial x_{0,12}} & \cdots & \frac{\partial d_{0,mn}}{\partial x_{0,ij}} & \cdots & \frac{\partial d_{0,mn}}{\partial x_{0,mn}} \end{pmatrix} \tag{4.40}$$

$$\nabla_{\mathbf{s}}\mathbf{d}_0 = \begin{pmatrix} \frac{\partial d_{0,12}}{\partial s_{0,12}} & \cdots & \frac{\partial d_{0,12}}{\partial s_{0,ij}} & \cdots & \frac{\partial d_{0,12}}{\partial s_{0,mn}} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \frac{\partial d_{0,ij}}{\partial s_{0,12}} & \cdots & \frac{\partial d_{0,ij}}{\partial s_{0,ij}} & \cdots & \frac{\partial d_{0,ij}}{\partial s_{0,mn}} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial d_{0,mn}}{\partial s_{0,12}} & \cdots & \frac{\partial d_{0,mn}}{\partial s_{0,ij}} & \cdots & \frac{\partial d_{0,mn}}{\partial s_{0,mn}} \end{pmatrix} \tag{4.41}$$

$\nabla_{\mathbf{t}}\mathbf{g}_0$, $\nabla_{\mathbf{x}}\mathbf{g}_0$ and $\nabla_{\mathbf{s}}\mathbf{g}_0$ are the partial derivatives matrices of $g_{0,ij}$ with respect to $t_{0,ij}$, $x_{0,ij}$, and $s_{0,ij}$ at the static SUE, respectively. $\nabla_{\mathbf{x}}\mathbf{t}_0$ and $\nabla_{\mathbf{s}}\mathbf{t}_0$ are the matrix of partial derivatives of $t_{0,ij}$ with respect to $x_{0,ij}$ and $s_{0,ij}$ at the static SUE, respectively. These matrices have the same form as the partial derivative matrices of function $\mathbf{d}_0$. By the chain rule of differentiation, we have:

$$\nabla_{\mathbf{x}}\mathbf{d}_0 = \mathbf{I} - \nabla_{\mathbf{t}}\mathbf{g}_0\nabla_{\mathbf{x}}\mathbf{t}_0 \tag{4.42}$$

$$\nabla_{\mathbf{s}}\mathbf{d}_0 = -\nabla_{\mathbf{t}}\mathbf{g}_0\nabla_{\mathbf{s}}\mathbf{t}_0 \tag{4.43}$$

To calculate $\nabla_{\mathbf{s}}\mathbf{x}_0$ at equilibrium, the general formula for derivative of the implicit function is adopted as:

$$\nabla_{\mathbf{s}}\mathbf{x}_0 = -\nabla_{\mathbf{x}}\mathbf{d}_0{}^{-1}\nabla_{\mathbf{s}}\mathbf{d}_0, \tag{4.44}$$

Substituting **Equations (4.42) and (4.43)** to **Equation (4.44)**, it is stated that:

$$\boldsymbol{\nabla_s \mathbf{x}_0 = (\mathbf{I} - \nabla_t \mathbf{g}_0 \nabla_x \mathbf{t}_0)^{-1} (\nabla_t \mathbf{g}_0 \nabla_s \mathbf{t}_0)}, \tag{4.45}$$

If we use the BPR curves to represent the travel time function, we have:

$$t_{0,ij} = t_{ij}^0 \left[1 + \alpha \left(\frac{z_{0,ij}}{Cap_{ij}}\right)^\beta\right] = t_{ij}^0 \left[1 + \alpha \left(\frac{x_{0,ij} - s_{0,ij}}{Cap_{ij}}\right)^\beta\right], \tag{4.46}$$

where t_{ij}^0 is the free-flow travel time of link ij, Cap_{ij} is the capacity of link ij, α, and β are parameters. $\boldsymbol{\nabla_x \mathbf{t}_0}$ and $\boldsymbol{\nabla_s \mathbf{t}_0}$ become diagonal matrices. Each diagonal elements of these matrices are represented by

$$\left.\frac{\partial t_{0,ij}}{\partial x_{0,ij}}\right|_{s_{0,ij}=0} = \frac{t_{ij}^0 \beta \alpha {x_{0,ij}}^{\beta-1}}{{Cap_{ij}}^\beta}, \tag{4.47}$$

$$\left.\frac{\partial t_{0,ij}}{\partial s_{0,ij}}\right|_{s_{0,ij}=0} = -\frac{t_{ij}^0 \beta \alpha {x_{0,ij}}^{\beta-1}}{{Cap_{ij}}^\beta}. \tag{4.48}$$

With the same approach as in Chapter 3, each element of $\boldsymbol{\nabla_t \mathbf{g}_0}$ is represented by $\frac{\partial g_{0,ij}}{\partial t_{0,gh}}$ and the calculation of this partial derivative is the same as the calculation of **Equation (3.46)** in Chapter 3. Since the final form of **Equation (3.46)** is depicted by **Equation (3.51)**, $\frac{\partial g_{0,ij}}{\partial t_{0,gh}}$ is given as

$$\frac{\partial g_{0,ij}}{\partial t_{0,gh}} = \theta \sum_{rs \in W_\tau} \left(\left[-x_{0,ij,gh}^{rs} + \frac{x_{0,ij}^{rs} x_{0,gh}^{rs}}{Q_\tau^{rs}}\right]\right), \tag{4.49}$$

where $x_{0,ij,gh}^{rs}$ is the number of flows from r to s choosing some routes which consist of both links ij and gh at the static SUE.

As highlighted in Chapter 3, **Equation (4.49)** is repeatedly computed for each OD pair rs, so we only need variables including link-based variables and node-based variables that reduce storage cost. The DFS-based algorithm proposed in Chapter 3 could be used to calculate $\boldsymbol{\nabla_t \mathbf{g}_0}$. Bui et al.[8] also used this technique. In the paper of Bui et al. [9], another use of the DFS algorithm was also proposed. Another option for this calculation is the link-based STOCH3 algorithm (see pages 87, 88 in Chapter 3). As shown in Chapter 3, calculations based on the DFS algorithm or STOCH3 algorithm are both effective and accurate.

Since $\mathbf{s}_\tau$ and $\mathbf{\nabla_s x}_0$ are completely calculated, the reference link flow influenced by the elimination of residual flow is approximated by (according to first-order Taylor expansion)

$$\mathbf{x}_\tau = \mathbf{x}_0 + \mathbf{\nabla_s x}_0 \mathbf{s}_\tau, \tag{4.50}$$

As a result, the first algorithm with the link-based STOCH3 approach in this study includes the following four steps. This process is computed in each period.

Step 1: Calculate static SUE by using the STOCH3 algorithm or the DFS loading procedure with the MSA method in period $\tau = 1$. The results of $\mathbf{x}_0$ and $\mathbf{t}_0$ are achieved after this step.

Step 2: Create an $|N| \times |N|$ matrix, **B**, to store the value of $\{y_{\tau,ij}^{rs}\}$. This matrix represents the flow propagation matrix for the next period. Based on the results at static SUE state, for each OD pair:

- Using STOCH3 algorithm once to calculate $x_{0,ij}^{rs}$ and running STOCH3 algorithm once from j to s in the case $x_{0,ij}^{rs} \neq 0$ to calculate $x_{0,ij\to gh}^{rs}$. After that, $x_{0,ij,gh}^{rs}$ is calculated based on $x_{0,ij}^{rs}$ and $x_{0,ij\to gh}^{rs}$ (detailed on pages 87, 88 in Chapter 3). $x_{0,ij}^{rs}$, $x_{0,ij\to gh}^{rs}$ and $x_{0,ij,gh}^{rs}$ could be also calculated by using the DFS-based algorithm (detailed on pages 85, 86 in Chapter 3).
- Calculate the residual link flow, flow propagation to the next period as follows:

$$y_{\tau,ij}^{rs} = \frac{t_{0,ij}}{L} x_{0,ij}^{rs}$$

$\{y_{\tau,ij}^{rs}\}$ is added to the row j and column s of matrix **B**.

- Compute the eliminated flow $s_{\tau,gh}^{rs}$ by using the following equation:

$$s_{\tau,gh}^{rs} = \sum_{ij\in A} \frac{x_{0,ij\to gh}^{rs}}{L} t_{0,ij}$$

- Compute $\frac{\partial g_{0,ij}}{\partial t_{0,gh}}$ by using **Equation (4.49)**.

After each calculation loop for each OD pair, we reuse $x_{0,ij}^{rs}$, $x_{0,ij\to gh}^{rs}$, $x_{0,ij,gh}^{rs}$ and $y_{\tau,ij}^{rs}$ variables for saving memory without

OD-pair-based variables and update $\mathbf{B}$, $\mathbf{s}_\tau$ and $\mathbf{\nabla_t g}_0$. After all OD pairs are considered, $\mathbf{B}$, $\mathbf{s}_\tau$ and $\mathbf{\nabla_t g}_0$ are completely computed.

Step 3: Obtain the results of reference link flows, $\mathbf{x}_\tau$, with the sensitivity analysis method and subtract the propagated flows from the results of link flows in the current period.

For each link, using **Equations (4.47) and (4.48)** to calculate $\frac{\partial t_{0,ij}}{\partial x_{0,ij}}$ and $\frac{\partial t_{0,ij}}{\partial s_{0,ij}}$. $\mathbf{\nabla_x t}_0$ and $\mathbf{\nabla_s t}_0$ are completely computed after all links are considered. After that, $\mathbf{\nabla_s x}_0$ is calculated by using **Equation (4.45)** and $\mathbf{x}_\tau$ is computed by using **Equation (4.50)**. Also, the solution of adjusted link flow in period τ is given as

$$\mathbf{z}_\tau = \mathbf{x}_\tau - \mathbf{s}_\tau$$

Step 4: Deliver the propagated flow to the subsequent period.

The matrix **B** is propagated to the next period and combined with the new travel demand appearing in the next period to create the travel demand matrix for the next period.

The above steps are repeated in the next period. This process will stop when all study periods are mentioned.

b. The second algorithm

In all of these above researches, because the residual flows were assumed and calculated at the static SUE, the equilibrium solution of semi-DTA with flow propagation was not shown in these models. By using DFS and STOCH3 algorithms, the first algorithm calculated the semi-DTA model with flow propagation and residual flow for the next period. However, because the residual link flows and eliminated link flows were assumed and calculated at the static SUE, either overestimation or underestimation may occur and the equilibrium solution of original semi-DTA with flow propagation may deviate from the approximate model. As can be seen from algorithm 1, after adjusted link flow is calculated in ***Step 3***, the link travel time is also calculated based on this link flow. The residual link flow and eliminated link flow in period τ depend on the link travel times in this period. So if we continue to calculate the residual link flow and eliminated

link flow based on the results of link travel times in ***Step 3***, the result of them may be different from the results in ***Step 2*** (because the results in ***Step 2*** based on the link travel time at the static SUE). Accordingly, we think of the new calculation process in which the reference link flow of the semi-DTA model is also calculated based on the results of $\mathbf{x}_0$ and $\boldsymbol{\nabla}_{\mathbf{s}}\mathbf{x}_0$ at the static SUE model, but the residual link flow and eliminated link flow are not fixed and be considered as unknown variables. A fixed-point problem of the eliminated link flow needs to be resolved.

We start from the calculation of $\mathbf{x}_\tau$ in the **Equation (4.50)**. Because only $\mathbf{x}_0$ and $\boldsymbol{\nabla}_{\mathbf{s}}\mathbf{x}_0$ are fixed, the reference link flow is the function of $\mathbf{s}_\tau$ as follows

$$\mathbf{x}_\tau = \mathbf{x}_\tau(\mathbf{x}_0 + \boldsymbol{\nabla}_{\mathbf{s}}\mathbf{x}_0\mathbf{s}_\tau), \tag{4.51}$$

where eliminated link flow $\mathbf{s}_\tau$ is the unknown variable and $\mathbf{x}_\tau(.)$ is the vector-valued function of the reference link flow time in period τ.

If we use the above results, the adjusted link flow and the link travel time are also the functions of $\mathbf{s}_\tau$ as follows:

$$\mathbf{z}_\tau = \mathbf{z}_\tau(\mathbf{x}_\tau(\mathbf{s}_\tau) - \mathbf{s}_\tau), \tag{4.52}$$

$$\mathbf{t}_\tau = \mathbf{t}_\tau(\mathbf{z}_\tau(\mathbf{x}_\tau(\mathbf{s}_\tau) - \mathbf{s}_\tau)). \tag{4.53}$$

where $\mathbf{z}_\tau(.)$ and $\mathbf{t}_\tau(.)$ are the vector-valued function of the adjusted link flow and the link travel time in period τ.

Let we denote $x_{\tau,ij}^{rs}$ is the amount of flows from r to s in period τ that choose some routes containing link ij and $x_{\tau,ij\to gh}^{rs}$ is the amount of flows from r to s choosing some routes which consist of both links ij and gh satisfying link ij is expected to use prior to link gh in period τ. Using the results in Chapter 3 and also in the first link-based algorithm of this chapter in the calculation of $x_{0,ij}^{rs}$ and $x_{0,ij\to gh}^{rs}$ based on $\{t_{0,ij}\}$, we can state that $x_{\tau,ij}^{rs}$ and $x_{\tau,ij\to gh}^{rs}$could be computed by running the DFS loading procedure or the STOCH3 algorithm once for each OD pair based on the link travel time value in period τ, $\{t_{\tau,ij}\}$. Thus, the residual flow and the eliminated flow that is calculated from $\{x_{\tau,ij}^{rs}\}$, $\{x_{\tau,ij\to gh}^{rs}\}$ and $\{t_{\tau,ij}\}$ are also the function of the link travel time value in period τ as follows:

$$y_{\tau,ij}^{rs} = \frac{x_{\tau,ij}^{rs}(\mathbf{t}_\tau)}{L} t_{\tau,ij} \tag{4.54}$$

$$s_{\tau,gh} = \sum_{rs \in W_\tau} \sum_{ij \in A} \frac{x^{rs}_{\tau,ij \to gh}(\mathbf{t}_\tau)}{L} t_{\tau,ij}. \quad (4.55)$$

From **Equations (4.53) and (4.55)**, unknown variable $s_{\tau,gh}$ for each link gh in each period is decided by solving the following fixed-point problem

$$s_{\tau,gh} = \sum_{rs \in W_\tau} \sum_{ij \in A} \frac{x^{rs}_{\tau,ij \to gh}(\mathbf{t}_\tau(\mathbf{s}_\tau))}{L} t_{\tau,ij}\,(\mathbf{s}_\tau). \quad (4.56)$$

If we can solve the fixed-point problem in **Equation (4.56)**, the equilibrium state gives the results of eliminated link flows, reference link flows, adjusted link flow, and link travel times. A new equilibrium has been established with taking into account the effect of the link travel time on the eliminated link flow as in the original model. With this manner, the new equilibrium has more chance to closer to the equilibrium of the original model than the equilibrium attained in the first link-based algorithm.

Note that the residual link flow does not appear in fixed-point problem in **Equation (4.56)** because we only consider the problem of $s_{\tau,gh}$ in the relationship with the link traffic flow. As can be seen, if we use this relationship, the residual link flow does not affect the results of $s_{\tau,gh}$. After achieving equilibrium, we can use the travel link at the semi-DTA SUE results and run the STOCH3 algorithm or DFS based algorithm once to calculate $x^{rs}_{\tau,ij}$ and use the **Equation (4.54)** to get the result of the residual link for flow propagation.

For solving the fixed-point problem $\mathbf{X} = \boldsymbol{F}(\mathbf{X})$, we can use the well-known MSA or some of the other efficient methods (see Liu et al. [10]). With the solution $\mathbf{X}^{(l)}$ at the current calculation loop l, the general MSA will find an auxiliary point by calculating $\mathbf{YX}^{(l)} = \boldsymbol{F}(\mathbf{X}^{(l)})$ and updating the current solution with the step size $\lambda^{(l)}$ by: $\mathbf{X}^{(l+1)} = \mathbf{X}^{(l)} + \lambda^{(l)}\left(\mathbf{YX}^{(l)} - \mathbf{X}^{(l)}\right)$, $l = 1;\ 2;\ \ldots$

The MSA method always used the step size $\lambda^{(l)} = \frac{1}{\vartheta^{(l)}}$ in which $\vartheta^{(l)}$ is expanded by a stable increment at each iteration as the following:

$$\vartheta^{(l)} = \begin{cases} 1, & l = 1 \\ \vartheta^{(l-1)} + 1, & l \geq 2 \end{cases} \quad (4.57)$$

However, it makes the convergence speed becomes painfully slow (Liu et al.

[10]). The well-known MSA has been modified by many researchers using alternative step size sequences (Liu et al. [10], Polyak[11]). However, the predetermined step size sequence may be too aggressive at the beginning iterations and the convergence speed of averaging methods could be slower than the adaptive averaging method. According to Liu et al. [10], due to the change of distance $\left\|\mathbf{X}^{(l)} - \mathbf{YX}^{(l)}\right\|$ from the previous iteration to the current one presents the convergence tendency, the increment of ϑ_k maybe chosen to speed up the convergence by using this information. Thereby, the self-regulated averaging method (SRA) was proposed, and the choice of step size increment was as follows

$$\vartheta^{(l)} = \begin{cases} \vartheta^{(l-1)} + \eta, \text{if } \left\|\mathbf{YX}^{(l)} - \mathbf{X}^{(l)}\right\| \geq \left\|\mathbf{YX}^{(l-1)} - \mathbf{X}^{(l-1)}\right\| \\ \vartheta^{(l-1)} + \gamma, \text{if } \left\|\mathbf{YX}^{(l)} - \mathbf{X}^{(l)}\right\| < \left\|\mathbf{YX}^{(l-1)} - \mathbf{X}^{(l-1)}\right\| \end{cases} \quad (4.58)$$

where η and γ are parameters ($\eta \in [1.5,2]$ and $\gamma \in [0.01,0.5]$) according to Liu et al.[10]. Liu et al.[10] showed that the SRA method outperformed the conventional MSA in convergence speed in the wide range of precise solutions. Thus, we will apply the SRA method for solving the fixed-point problem of **Equation (4.56)**.

As a result, the algorithm 2 with link-based and node-based approach in this study includes the following four steps. This process is computed in each period.

Step 1: Calculate static SUE by using the STOCH3 algorithm or DFS loading procedure with the MSA method in period $\tau = 1$. The results of $\mathbf{x}_0$ and $\mathbf{t}_0$ are attained after this step.

Step 2: Set current iteration counter $l := 1$, $\vartheta^{(l-1)} = 0$ and convergence test value σ. Based on the results at SUE state, calculating $\boldsymbol{\nabla_s \mathbf{x}}_0$ and the initial solution of eliminated flow $\mathbf{s}_\tau{}^{(l)}$. The calculation process is the same as the calculation process shown in ***Step 2*** of algorithm 1, however, we do not need to calculate $y_{\tau,ij}^{rs}$ and matrix **B**.

Step 3: Solving the fixed-point problem of **Equation (4.56)**:

(a) Based on the current solution of eliminated flow $\mathbf{s}_\tau{}^{(l)}$, calculating the reference link flow $\mathbf{x}_\tau{}^{(l)}$ by using **Equation (4.51)**. Adjusted link flow $\mathbf{z}_\tau{}^{(l)}$ is calculated and used to calculate link travel time $\mathbf{t}_\tau{}^{(l)}$ by using **Equation (4.52) and (4.53)**.

(b) Based on $\mathbf{t}_\tau{}^{(l)}$, running STOCH3 algorithm or DFS loading

procedure (see Chapter 3) to calculate $\left\{x_{\tau,ij\to gh}^{rs}{}^{(l)}\right\}$ and obtain the auxiliary eliminated link flow as:

$$sy_{\tau,gh}{}^{(l)} = \sum_{rs\in W_\tau}\sum_{ij\in A}\frac{x_{\tau,ij\to gh}^{rs}}{L}t_{\tau,ij}$$

where $sy_{\tau,gh}{}^{(l)}$ is the auxiliary solution of the eliminated link flow of link gh in period τ.

(c) Let $rg_{\tau,gh}{}^{(l)}$ denotes the gap between the current solution and the auxiliary solution of the eliminated link flow of link gh in period τ. Set $rg_{\tau,gh}{}^{(l)} = sy_{\tau,gh}{}^{(l)} - s_{\tau,gh}{}^{(l)}$

(d) Update $s_{\tau,gh}{}^{(l)} = s_{\tau,gh}{}^{(l)} + \frac{1}{\vartheta^{(l)}}rg_{\tau,gh}{}^{(l)}$ where $\vartheta^{(l)}$ is calculated by using **Equation (4.58)**.

(e) Stopping test: If $\max_{gh\in A}\left\{\left|rg_{\tau,gh}{}^{(l)}\right|\right\} \le \sigma$, stop. $\{s_{ij}^{(l)}\}$, $\{x_{ij}^{(l)}\}$, $\left\{z_{ij}{}^{(l)}\right\}$ and $\left\{t_{ij}{}^{(l)}\right\}$ depict the equilibrium solutions. Otherwise, go to substep (a).

Step 4: Calculate the residual link flow for flow propagation and deliver the propagated flow to the next period.

Create an $|N| \times |N|$ matrix, **B**, to store the value of $\left\{y_{\tau,ij}^{rs}\right\}$. Based on the equilibrium results at step 3, for each OD pair: Using STOCH3 algorithm or DFS-based algorithm once to calculate $x_{\tau,ij}^{rs}$ and calculate the residual link flow:

$$y_{\tau,ij}^{rs} = \frac{t_{\tau,ij}}{L}x_{\tau,ij}^{rs}$$

$\left\{y_{\tau,ij}^{rs}\right\}$ is added to the row j and column s of matrix **B**. After all OD pairs are considered, **B** is completely computed. Matrix **B** is propagated to the period $\tau + 1$ and integrated with the new travel demand departing in period $\tau + 1$ to create the travel demand matrix for period $\tau + 1$.

4.5 Applications to virtual and real-scale networks

4.5.1 Applications to a virtual road network

A small network containing 6 nodes, 6 links, and 3 OD pairs, as shown in **Figure 4.4**, is used. The link parameters are written in the form of [free-flow travel time (min), capacity (pcu)]. **Table 4.1** shows the travel demand in two time periods. OD pairs of this network are from node 1 to node 6, node 2 to node 6, node 3 to node 6. We assume $\theta = 0.5, \alpha = 0.15, \beta = 4$, and $L = 60$ (min).

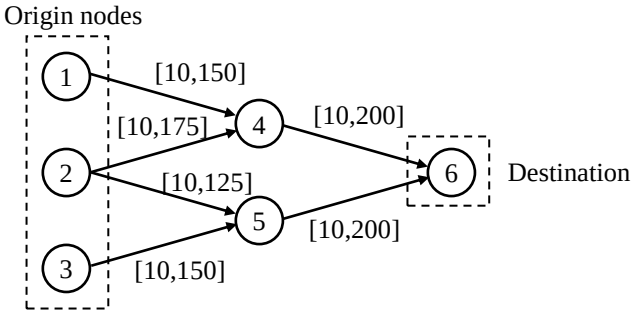


Figure 4.4 A small traffic network

Table 4.1 Travel demand in two time periods in the small traffic network

No.	O–D	Travel demand in time 1 (pcu)	Travel demand in time 2 (pcu)
1	1 → 6	70	60
2	2 → 6	350	300
3	3 → 6	70	60

With all the above input data, the Fortran programming language is used for compiling and the output results are attained.

Firstly, we detail the results of the route-based approach and the first algorithm link-based approach in period 1. The results of the first algorithm of the link-based approach and route-based approach will be compared.

Step 1: SUE static computation.

- Route-based approach:

With the route-based approach, we need to enumerate routes connecting each OD pair. In the given small network, OD pair 1 includes 1 route, OD pair 2 includes 2 routes, and OD pair 3 consists of 1 route. The results of the probability of choosing a route, route travel times, and route flows are shown in the following table:

Table 4.2 The results of routes at static SUE in period 1 of the small network

OD	Route	Links included	Probability route choice	Route travel time (min)	Route flow (pcu)
1	1	14, 46	1.00	24.34	70.00
2	2	24, 46	0.54	26.34	189.78
	3	25, 56	0.46	26.68	160.22
3	4	35, 56	1.00	22.70	70.00

- *Link-based approach*

By applying STOCH3 algorithm to this example, the link traffic flows and link travel time at the static SUE are:

Table 4.3 The results of links at static SUE in period 1 of the small network

Link	Link flow (pcu)	Link travel time (min)
14	70.00	10.07
24	189.78	12.07
25	160.22	14.05
35	70.00	10.07
46	259.78	14.27
56	230.22	12.63

Step 2: Calculating residual link flows, eliminated link flows and the reference link flows with the sensitivity analysis method.

- *Route-based approach*

By using **Equations from (4.6)-(4.9)**, we can calculate residual link flows and eliminated link flows

Table 4.4 The results of links at static SUE in period 1 of the small network

Link	Residual link flow (pcu)	Eliminated link flow (pcu)
14	11.75	0.00
24	38.19	0.00
25	37.51	0.00
35	11.75	0.00
46	61.78	49.94
56	48.47	49.26

Computing the derivative of route traffic flows concerning eliminated flow, $\nabla_s \mathbf{f}_0$. This calculation needs to calculate an inverse matrix with a route-based variable, so if there are a huge amount of routes in the route-set, the calculational time is very big. The result of this calculation is given by:

Table 4.5 The derivative of route traffic flows concerning eliminated flow in period 1 of the small network

Link / Route	14	24	25	35	46	56
1	0.00	0.00	0.00	0.00	0.00	0.00
2	0.00	0.16	-0.36	0.00	0.24	-0.16
3	0.00	-0.16	0.36	0.00	-0.24	0.16
4	0.00	0.00	0.00	0.00	0.00	0.00

Computing the reference route traffic flows: The results are shown as the following table:

Table 4.6 The reference route traffic flows in period 1 of the small network

Route	Reference route traffic flow (pcu)
1	70.00
2	193.46
3	156.54
4	70.00

From the link-route incidence matrix, we can calculate the reference link traffic flows that are shown as the following table:

Table 4.7 The reference link traffic flows in period 1 of the small network

Link	Reference link traffic flow (pcu)
14	70.00
24	193.46
25	156.54
35	70.00
46	263.46
56	226.54

- Link-based approach

The results of calculating $x_{0,ij}^{rs}$ is given by:

Table 4.8 The reference link traffic flows of each OD pair in period 1 of the small network

OD	link	$x_{0,ij}^{rs}$
1	14	70.00
	46	70.00
2	24	189.78
	25	160.22
	46	189.78
	56	160.22
3	35	70.00
	56	70.00

The results of calculating computing $x_{0,ij,gh}^{rs}$ is shown as:

Table 4.9 The result of $x_{0,ij,gh}^{rs}$ in period 1 of the small network

OD	*ij*	*gh*	$x_{0,ij,gh}^{rs}$
1	14	14	70.00
	14	46	70.00
	46	14	70.00
	46	46	70.00
2	24	24	189.78
	24	46	189.78
	25	25	160.22
	25	56	160.22
	46	24	189.78
	46	46	189.78
	56	25	160.22
	56	56	160.22
3	35	35	70.00
	35	56	70.00
	56	35	70.00
	56	56	70.00

And, the residual link flows and eliminated link flows are calculated at static SUE as the following results:

Table 4.10 The residual link flows and eliminated link flows in period 1 of the small network

Link	Residual link flow (pcu)	Eliminated link flow (pcu)
14	11.75	0.00
24	38.19	0.00
25	37.52	0.00
35	11.75	0.00
46	61.78	49.94
56	48.48	49.27

After that, the computing $\nabla_t \mathbf{g_0}$ and $\nabla_s \mathbf{x_0}$ are conducted by using the above results. We have:

Table 4.11 The result of $\nabla_t \mathbf{g_0}$ in period 1 of the mall network

Link / Link	24	25	46	56
24	-43.44	43.44	-43.44	43.44
25	43.44	-43.44	43.44	-43.44
46	-43.44	43.44	-43.44	43.44
56	43.44	-43.44	43.44	-43.44

Table 4.12 The result of $\nabla_s \mathbf{x_0}$ in period 1 of the small network

Link / Link	24	25	46	56
24	0.16	-0.36	0.24	-0.16
25	-0.16	0.36	-0.24	0.16
46	0.16	-0.36	0.24	-0.16
56	-0.16	0.36	-0.24	0.16

Finally, the reference link flows are calculated and the results are depicted through **Table 4.13**.

Table 4.13 The reference link flows in period 1 of the small network

Link	Reference link flow (pcu)
14	70.00
24	193.46
25	156.54
35	70.00
46	263.46
56	226.54

We can see that it is no difference between the two approaches in the results of the calculation. However, the link-based approach does not enumerate routes and not use the route-based variables, so it is feasible to apply it to the road network because it does not depend on the number of routes connecting each OD pair.

Step 3: Subtract the propagated flows from the results of link flows in the previous step.

The results of link-based and route-based approaches are almost the same and given by the following table

Table 4.14 The adjusted link flows in period 1 of the small network

Link	Adjusted link flow (pcu)
14	70.00
24	193.46
25	156.54
35	70.00
46	213.52
56	177.27

Step 4: Deliver the propagated flow to the next period. The propagated flow of a link is considered as the demand between the end node of that link and the original destination.

Because in the semi-DTA model, we do not only apply the STA model but also consider flow propagate to the next period. So, in period 2, there is also flow propagate derived from period 1. So, travel demand in period 2 will be raised in the route from node 4 to node 6 and there is one more OD pair from node 5 to node 6. Travel demand in period 2 is detailed in the following table

Table 4.15 Travel demand in period 2 of the small network after flow propagation

No.	O–D	Travel demand (pcu)
1	1 →6	60
2	2 →6	300
3	3 →6	60
4	4 →6	49.94
5	5 →6	49.27

With the same manner, our results in period 2 are listed in the following table

Table 4.16 The results of links in period 2 of the small network after using sensitivity analysis

Link	Adjusted link flow (pcu)	Residual Link flow (pcu)	Eliminated link flow (pcu)
14	60.00	10.04	0.00
24	160.86	29.15	0.00
25	139.14	29.30	0.00
35	60.00	10.04	0.00
46	231.61	66.67	39.19
56	209.07	57.16	39.33

After all, we will show the comparison of results of the route-based semi-DTA and our link-based semi-DTA using the first algorithm. The results showed that there are no significant differences between the results of the route-based approach and our proposed link-based approach of semi-DTA with sensitivity analysis. Nevertheless, with the link-based approach, we can apply the semi-DTA model to a real network because of efficient computation.

Table 4.17 The comparison between the results of route-based approach and the first algorithm of the link-based approach of the small network after using sensitivity analysis

Period	Link	Reference Link flow (pcu)		Residual link flow (pcu)		Eliminated link flow (pcu)	
		Route-based	Link-based	Route-based	Link-based	Route-based	Link-based
1	14	70.00	70.00	11.75	11.75	0.00	0.00
	24	193.46	193.46	38.19	38.19	0.00	0.00
	25	156.54	156.54	37.52	37.51	0.00	0.00
	35	70.00	70.00	11.75	11.75	0.00	0.00
	46	213.52	213.52	61.78	61.78	49.94	49.94
	56	177.27	177.27	48.48	48.47	49.27	49.26
2	14	60.00	60.00	10.04	10.04	0.00	0.00
	24	160.86	160.86	29.15	29.15	0.00	0.00
	25	139.14	139.14	29.30	29.30	0.00	0.00
	35	60.00	60.00	10.04	10.04	0.00	0.00
	46	231.61	231.61	54.28	54.28	39.19	39.19
	56	209.07	209.07	45.92	45.92	39.33	39.33

Consequently, we show that the second algorithm is more accurate to reflex the original model. The second algorithm and the original semi-DTA model are also applied to the small network. In this small example, it is possible to calculate the original semi-DTA model with the double-looped fixed-point problem by using the MSA method. The results of the original model (model a), the approximated model using sensitivity analysis with the first algorithm (model b), and the second

algorithm (model c) are performed in **Table 4.18**. Two indicators $RMSE$ and $\%RMS$ introduced in Chapter 3 are also used for the comparisons between the results of two approximated approaches and the original model.

Table 4.18 Comparisons between the results of models in the small example network. (a) Not the approximated model, (b) Approximate model using the first link-based algorithm, (c) Approximate model using the second link-based algorithm.

Period	Link	Adjusted link flow (pcu)			Residual link flow (pcu)			Eliminated link flow (pcu)		
		(a)	(b)	(c)	(a)	(b)	(c)	(a)	(b)	(c)
1	14	70.00	70.00	70.00	11.75	11.75	11.75	0.00	0.00	0.00
	24	193.90	193.46	194.00	39.62	38.19	39.30	0.00	0.00	0.00
	25	156.10	156.54	156.00	35.51	37.52	35.85	0.00	0.00	0.00
	35	70.00	70.00	70.00	11.75	11.75	11.75	0.00	0.00	0.00
	46	212.53	213.52	212.94	52.40	61.78	52.14	51.37	49.94	51.05
	56	178.84	177.27	178.40	41.30	48.48	41.56	47.26	49.27	47.60
	$RMSE$		0.80	0.25		4.93	0.24		1.01	0.19
	$\%RMS$		0.54	0.17		15.37	0.76		6.12	1.17
2	14	60.00	60.00	60.00	10.04	10.04	10.04	0.00	0.00	0.00
	24	160.58	160.86	160.69	29.61	29.15	29.77	0.00	0.00	0.00
	25	139.42	139.14	139.31	28.63	29.30	28.44	0.00	0.00	0.00
	35	60.00	60.00	60.00	10.04	10.04	10.04	0.00	0.00	0.00
	46	232.31	231.61	231.93	57.70	66.67	57.73	39.65	39.19	39.81
	56	208.01	209.07	208.44	48.33	57.16	48.29	38.67	39.33	38.48
	$RMSE$		0.54	0.24		5.15	0.10		0.33	0.10
	$\%RMS$		0.38	0.17		16.76	0.34		2.53	0.78

Table 4.18 shows that the second algorithm performs better with smaller values of both the $RMSE$ and $\%RMS$ indicators than the first algorithm does. It is easy to see that there is no significant difference in the two-time period between the original model and the second algorithm. Moreover, although the outcome of the first approach is good when only referring to reference link flows, it is an overestimate when referring to the results of residual link flows and eliminated

link flows. Thusly, sensitivity analysis in the second approach can be used to represent the approximate value of the semi-dynamic SUE traffic assignment model. With the link-based approach, we do not need to use route-variable for saving memory and the proposed model using STOCH3 has shown superiority in computing time. The efficiency in the calculation time of the proposed model is depicted in **Figure 4.5**. With this small network, the first algorithm and the second algorithm depict the same calculation time. Comparing to the original model, the total saving time of both approaches are up to 98.5%.

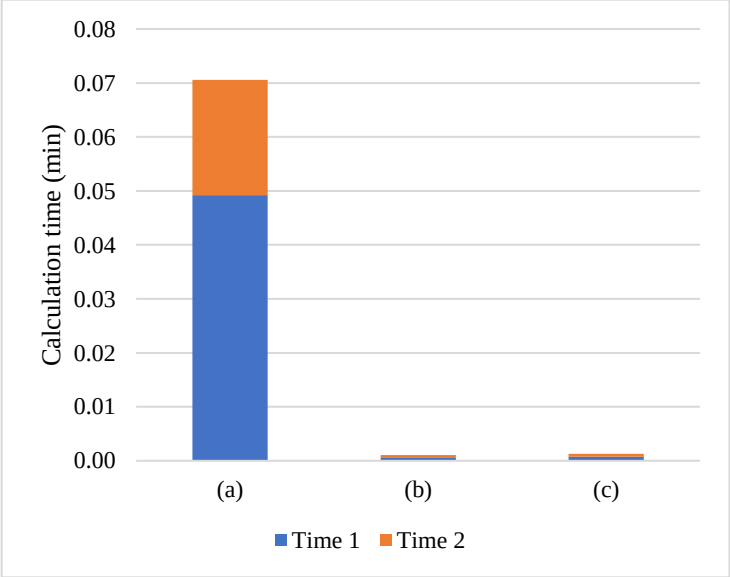


Figure 4.5 The calculation time of different models with the small network. (a) Not the approximated model, (b) Approximate model using the first link-based algorithm, (c) Approximate model using the second link-based algorithm.

4.5.2 Applications to Kanazawa road network

To examine the applicability of the proposed model to a real network. This subsection will show the application to the Kanazawa road network in Japan including 272 nodes and 964 links. The duration of a period (L) is set at 60 minutes and there are three periods including period 1 (6:00-7:00 AM), 2 (7:00-8:00 AM), 3 (8:00-9:00 AM) with the previously personal trip survey OD demand data. The proposed semi-dynamic traffic assignment model is calculated for each period so it is possible to calculate for more periods. Here we only apply to 3 periods because

of limited data input. Moreover, the model will be suitable in the peak periods because the model could represent the movement of flows between periods. At the morning peak periods, there will be a huge amount of flows cannot reach the destination, so the semi-dynamic traffic model considering the flow propagation between periods will be appreciated. **Figure 4.6** shows the method of setting the OD pairs, which are classified according to the departure time. Specifically, those departing between 6:00 AM and 6:59 AM are treated as traffic generated at 6:00 AM, and the same applies to subsequent periods. In this way, the OD traffic volume at 6 AM is the traffic volume that completes the trip within 6 o'clock (between 6:00 AM and 6:59 AM) and the traffic that departs between 6:00 AM and 6:59 AM and arrives between 7:00 AM and 7:59 AM. In **Figure 4.6**, the OD traffic volume at 6 o'clock is represented by the sum of pattern 1 and pattern 2. Similarly, set the OD traffic volume at 7 o'clock, 8 o'clock.

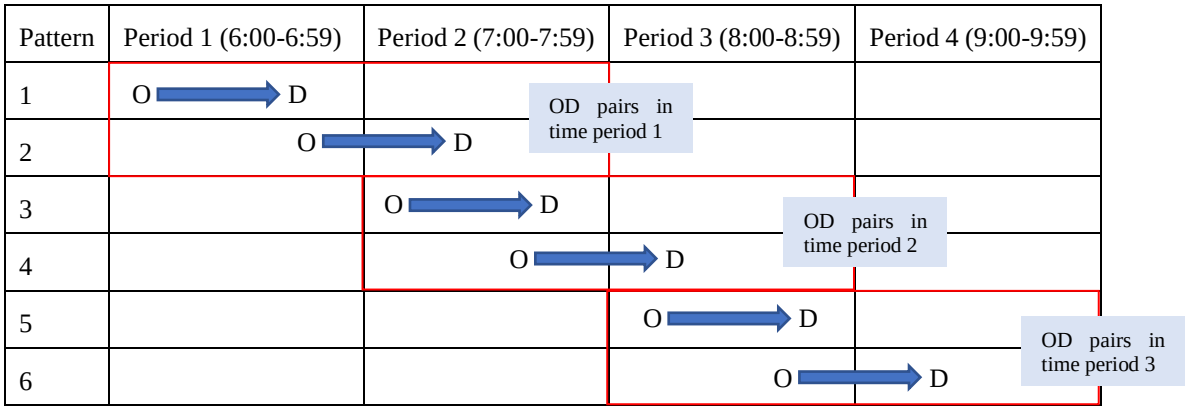


Figure 4.6 Classification of OD traffic

Data with an extra-long trip time, such as departure at 6:00 AM and arrival at 9:00 AM, is removed. This is because it is thought that it may affect the result. As a result of cleaning the data, the total number of OD pairs and travel demand was as shown in **Table 4.19**.

Table 4.19 OD matrix of the Kanazawa road network

Period	Number of OD pairs	Travel demand (pcu)
1	383	10,152
2	2,409	71,112
3	1,590	44,236

Next, the link traffic volume is described. The link traffic volume is the sum of three types of traffic volume: the traffic volume that can complete a trip within a certain period, the traffic volume that has remained from the previous period, and the residual traffic volume that is carried over to the subsequent period. The residual traffic of the model treated in this study is the traffic existing on the link at the end of the period. Therefore, the link traffic volume is the sum of the above three types of traffic volume, in other words, the traffic volume that can pass through the link in each period. **Figure 4.7** shows an illustration of the link traffic volume.

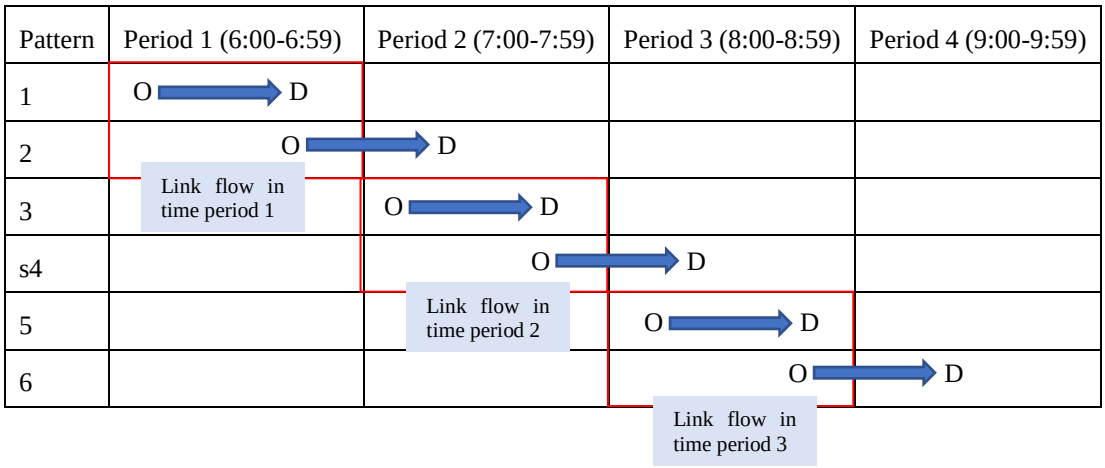


Figure 4.7 Link traffic flow illustration

Firstly, we will compare two approximated approaches: the proposed first algorithm with the link-based STOCH3 approach and the route-based approach. For comparison, the set of route choices is identical. The BPR function of the travel time of the car uses assumed parameters ($\alpha = 1, \beta = 2$) and $\theta = 0.2$. Because the number of OD pairs and routes in periods 2 and 3 are too big that makes the route-based approach cannot run in periods 2 and 3, we will use the data in period 1 for comparison. The results of the comparison are shown in **Table 4.20** and **Figure 4.8**. While the error of comparing two approaches is approximately 0 that suggests the same results, the link-based approach demonstrates the time-efficient calculation. In this comparison, using the proposed method will reduce the computational time above 97%. This effect is increased as the amount of route choice set increases. The differences between the link-based approach and route-

based approach are in the calculation time and applicability. Itagaki et al.[6] limited the number of routes in the route-set to 3 routes at the maximum per 1 OD which reduced the stochastic in route selection. Either the extremely large alternative routes connecting an OD pair in a real network or the increase of OD pairs due to the propagation of residual flow can cause overloaded calculation in a route-based approach. The link-based approach shows independence on the number of implicit route-set. The results suggest that the proposed link-based method is highly applicable.

Table 4.20 The results of the comparison between two approximated approaches with Kanazawa road network in period 1

Number of routes	*RMSE*	Calculation Time (min)	
		Link-based	Route-based
3092	0.00029	0.35	15.37
3819	0.00028	0.36	26.96
4995	0.00027	0.43	65.07
5893	0.00027	0.41	130.16
6852	0.00023	0.40	204.71

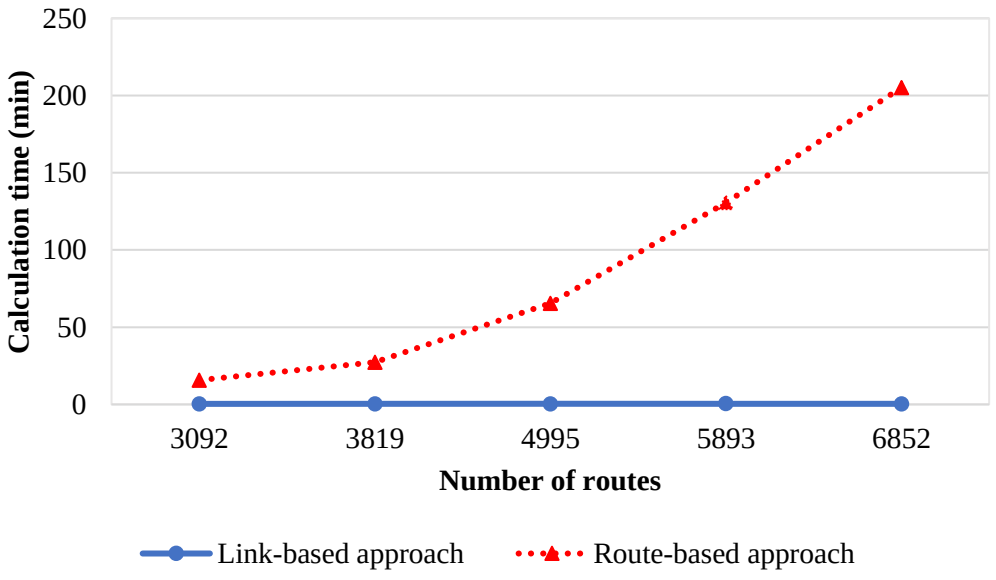
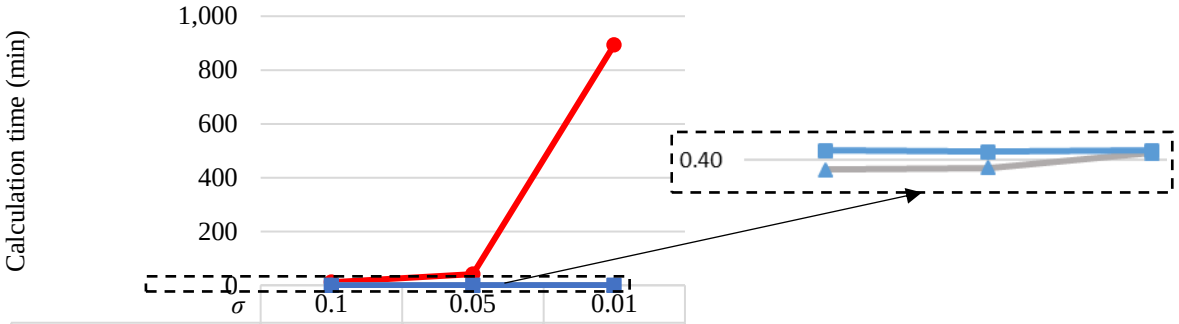


Figure 4.8 Calculation time comparison between two approximated approaches with Kanazawa road network in period 1

Secondly, to compare the real effectiveness of the two link-based STOCH3 approaches when applied in practice, the next two link-based approaches will be applied and compared to the true model. The "elongation ratio" parameter to define the STOCH3-efficient route (see Leurent [7]) is set at 1.5 for every link of the network. Because the route-based approach of the original model must solve the double-looped fixed-point problem, the calculation time for reaching equilibrium in period 1 (with convergence error $\sigma = 10^{-3}$ that is used to assess the convergence level of the fixed-point problem, see ***Step 5*** on page 108) and in period 2 exceed 1 week, and the route-based approach cannot run in period 2. Thus, we only conduct tests in period 1 (with $\sigma \in [10^{-1}, 10^{-2}]$) to compare different models. While **Figure 4.9** shows the comparison in the calculational time of different models in these tests, **Figure 4.10** depicts the $RMSE$ and $\%RMS$ indicators in the adjusted link flows, residual link flows, and eliminated link flows. The computational time of the proposed method is very small when compared to the calculational time of the route-based approach of the original model. With σ set to 0.1, the calculation time of the original model is 11.36 minutes, while the calculation time of the proposed method is only 0.5 minutes. The proposed method will save at least 96% in this comparison. Especially, this effect will increase as the error decreases. This demonstrates the time-efficient calculation of the proposed method. The first algorithm and the second algorithm also show the same calculation time in this test. The efficiency of calculation time is very high in both approaches. However, **Figure 4.10** illustrates that the second algorithm creates more exact results. Moreover, the very small of two indicators $RMSE$ and $\%RMS$ in **Figure 4.10** reflect the calculation accuracy of the proposed method. Moreover, this accuracy will also enhance when the error chosen diminishes.



σ	0.1	0.05	0.01
Not approximated model	11.36	41.37	893.67
The first link-based algorithm	0.36	0.37	0.43
The second link-based algorithm	0.44	0.44	0.44

Figure 4.9 The calculation time of different models with Kanazawa road network in period 1

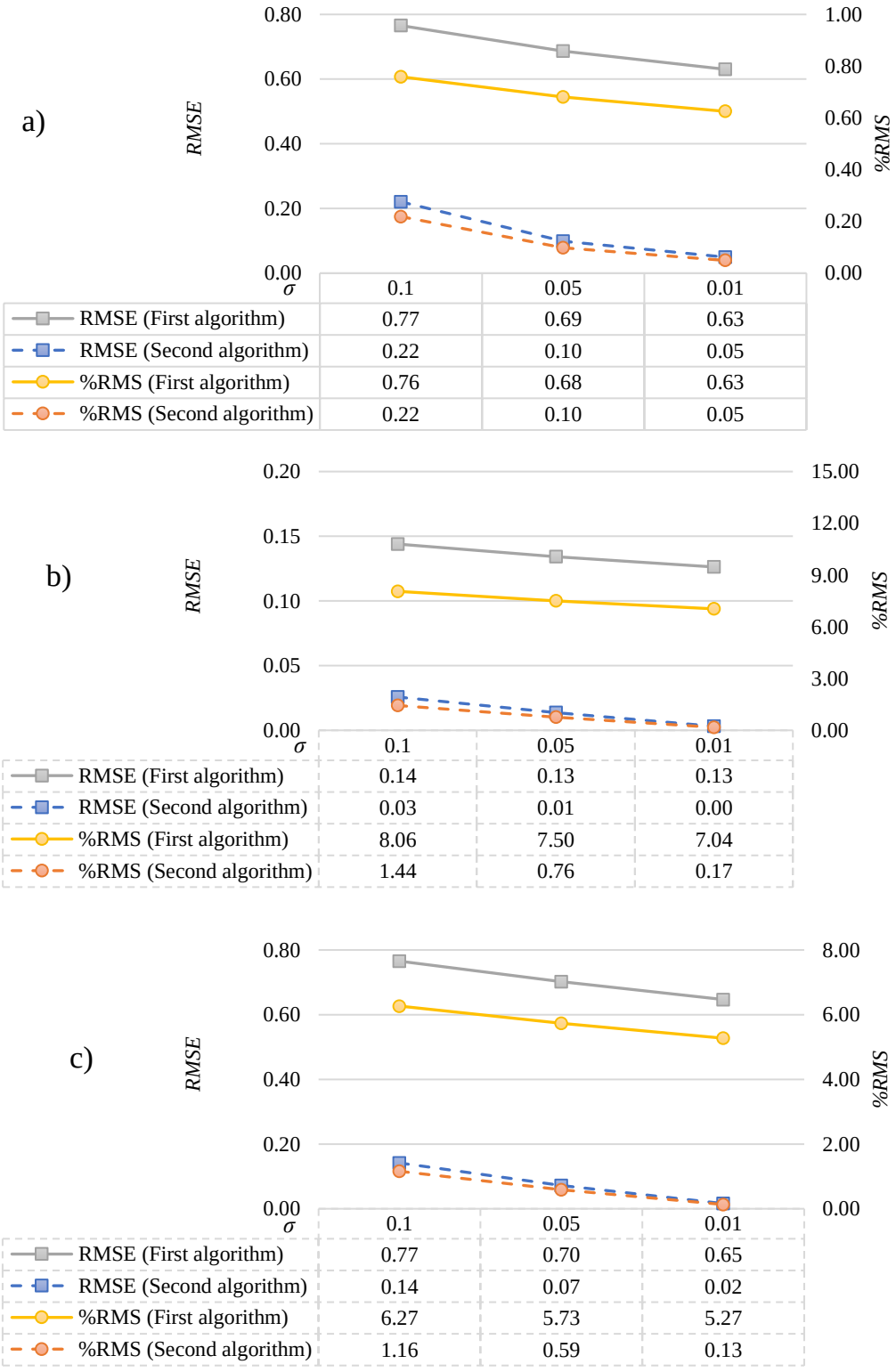


Figure 4.10 *RMSE* and *%RMS* indicators between the original model and approximate model using two link-based algorithms with Kanazawa road network in period 1. (a) Adjusted link flow results. (b) Residual link flow results. (c) Total eliminated link flow results.

Consequently, we apply both link-based algorithms to the Kanazawa road network for all three time periods. The total calculation time of both approaches is quite small, the first approach is about 2 minutes 46 seconds and the second approach is about 3 minutes. The calculation time of the second algorithm is larger than the calculation time of the first approach but the difference is insignificant. The second algorithm must solve one more problem of finding a fixed point. Moreover, **Figures 4.11** and **4.12** shows the comparisons between observed link flows and calculated link flows of the static SUE model and two link-based algorithms of the semi-DTA SUE model.

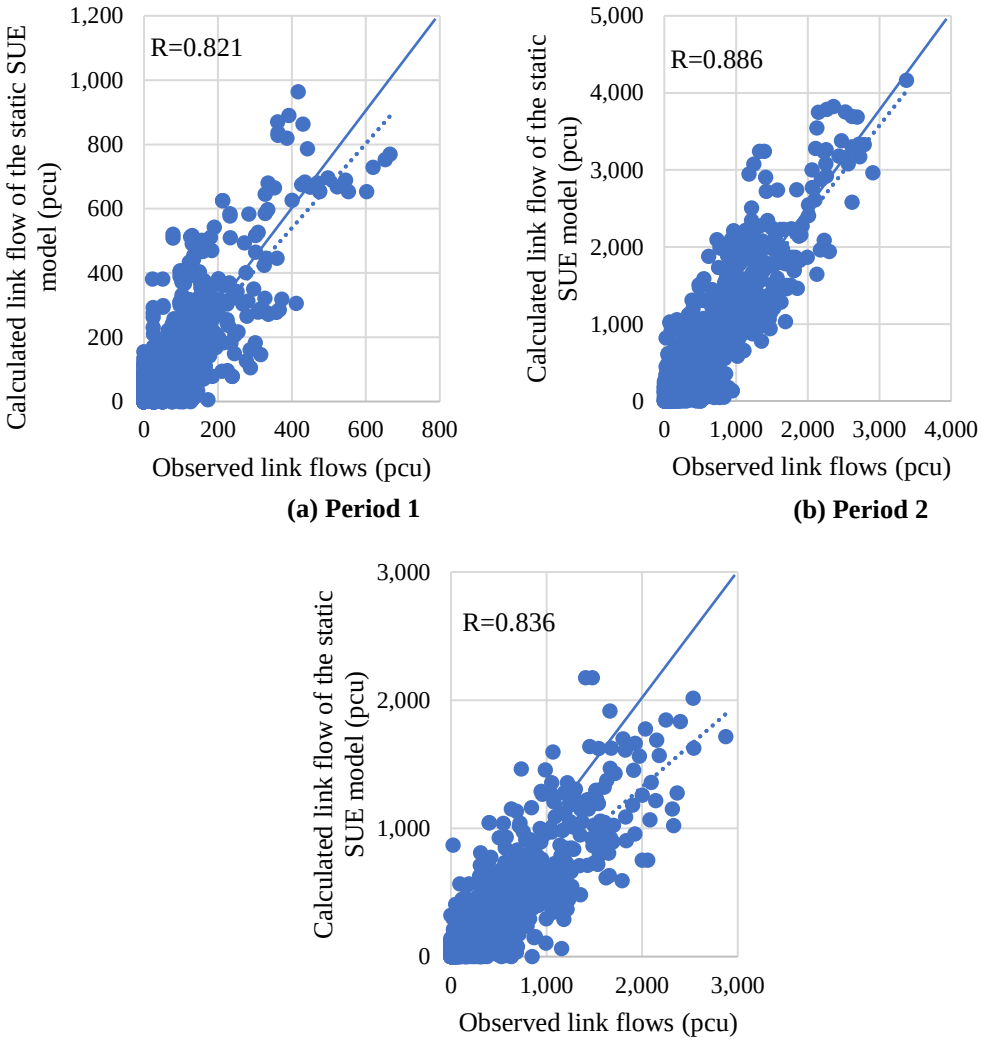


Figure 4.11 Scatter plots between the observed and calculated link flow of the static SUE model in each period of Kanazawa road network

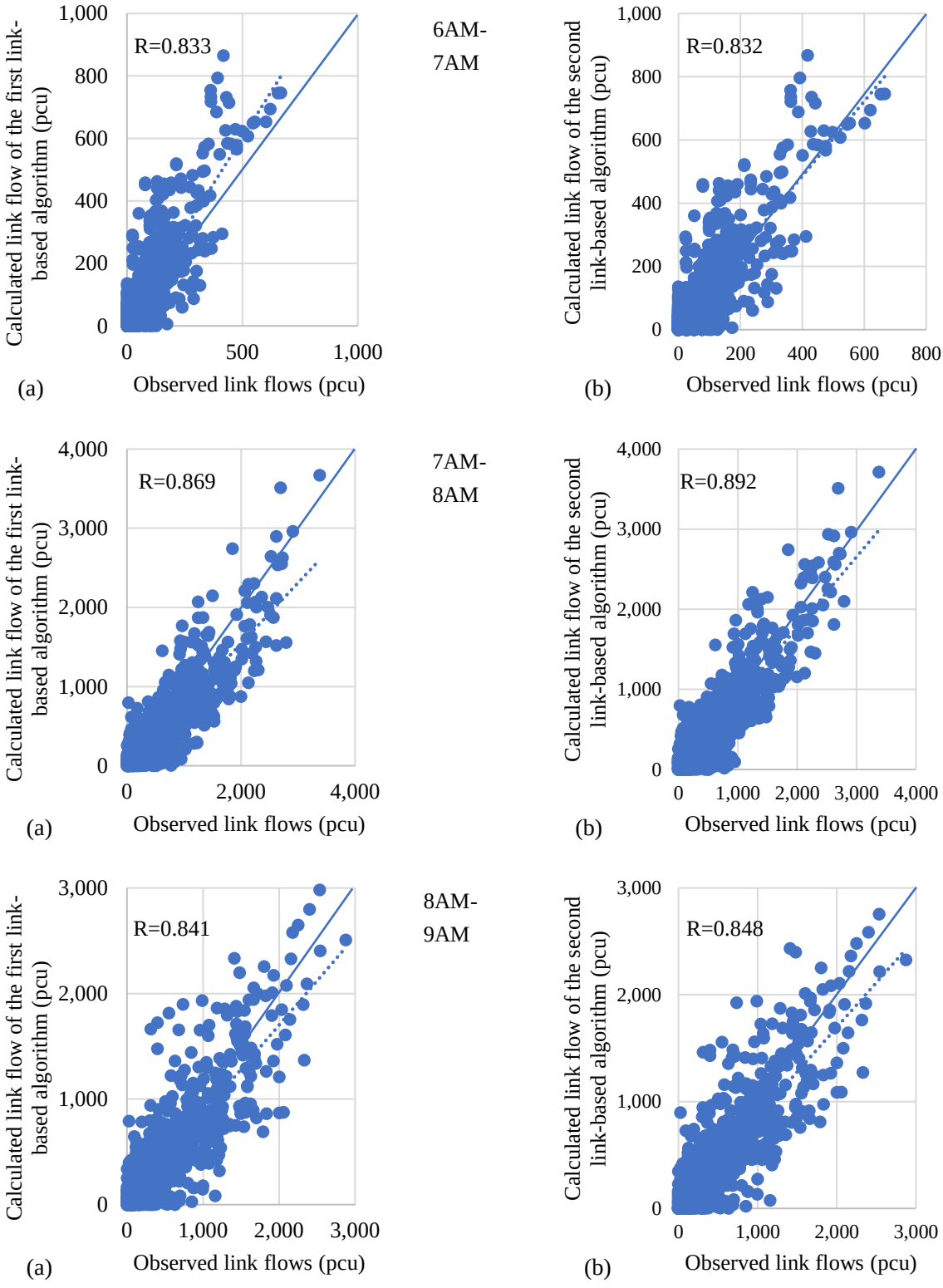


Figure 4.12 Scatter plots between the observed and calculated link flow of the semi-DTA model using the sensitivity analysis in each period of Kanazawa road network. (a) The first link-based algorithm, (b) The second link-based algorithm

It can be seen that the two algorithms to solve the semi-DTA SUE model give better results than the static SUE model. Because the static model does not address the movement of congested traffic from one period to another, it cannot accurately reflect the level of traffic congestion at each period. The static model assumes that all travel demands departing at each period can reach the destination within this period. It may reflect the traffic flow at the period when the congestion has not yet occurred, however, there will be underestimation in the congestion period. As can be seen from **Figure 4.11**, the static model is still capable of reflecting the traffic flow in period 2. However, when congestion occurs in period 2 due to the huge amount of travel demand (see **Table 4.19**), a large amount of travel demand in period 2 that cannot be reached at this period continues to travel in period 3. The static model is unable to mention this flow propagation. Therefore, the static model produces underestimation results in period 3. Meanwhile, the semi-DTA model considering the flow propagation between periods solved by two algorithms gives closer results to observed link flows. Moreover, both algorithms in the semi-DTA model give guaranteed results in correlation coefficients. **Figure 4.12** also shows that period 2 experiences the biggest traffic volumes because it is the peak hour and the congestions appear during periods 2 and 3. However, there are a tendency of slight overestimation in period 1 and the opposite trend in the remaining periods. The reason is that periods 2 and 3 are congestion periods and residual flow that is dependent on the travel time is computed a lot in these periods. Besides, the absence of a suitable OD matrix for the semi-DTA model is also considered as a reason. Future research needs to focus on solving these issues. Furthermore, the calculation results of the second approach are better than the first one in terms of correlation coefficients in all 3 periods. In period 2 of which the congestion level is highest, the results of the first approach tend to be more underestimated because the calculation of residual flow is excluded. Meanwhile, the second approach gives results closer to reality.

For more detail, we have compared the calculation results of the semi-DTA model with sensitivity analysis based on two link-based algorithms and the static SUE model in **Figure 4.13**.

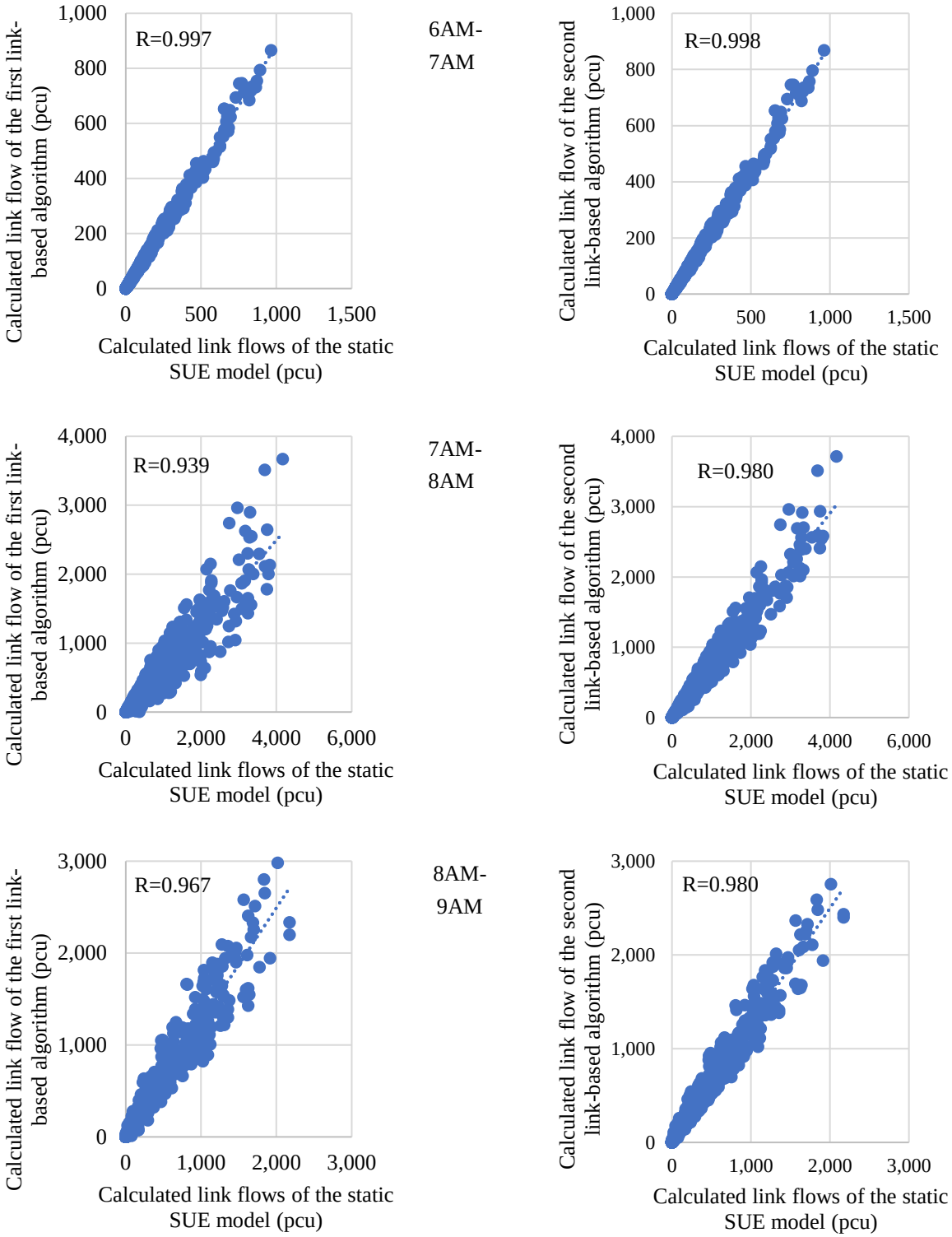


Figure 4.13 Scatter plots between the calculated link flows of the static SUE model and two link-based algorithms of the semi-DTA model using the sensitivity analysis in each period of Kanazawa road network.

Figure 4.13 indicates that the semi-DTA model can replace the static model and have a close relationship with the static model. Through the above figure, the semi-DTA model is very close to the static model in period 1 because there is no congestion in this period. However, the differences between the two models occur in periods 2 and 3 when there are traffic congestion and the flow propagation between periods. As mentioned above, residual flow between periods plays an important role in the proposed model. Consequently, **Figure 4.14** will show the residual flows on the links between periods of the second algorithm. Because the residual flow from period 1 is very small in all links (less than 25 pcu), **Figure 4.14** only shows the residual flow from period 2 and period 3. The residual flows in period 2 are larger than those in period 3. As can be seen, the more congested the road is, the more the residual flows increase. From **Figure 4.14** we also see that the congestion trend is towards the city center. This propagation has resulted in a change in the total number of OD pairs per period and the number of traffic flows processed in each period. **Table 4.21** shows the new OD matrix of the semi-DTA model using the second link-based algorithm. This OD matrix is constructed from the original OD matrix of **Table 4.19** and by adding the residual flows from the previous period. Compared with data in **Table 4.19**, the traffic flow in period 3 is quite large when the number of OD pairs increases 10 times and traffic demand also increases by nearly 25,000 (pcu) in this period. This suggests that the semi-DTA model is likely to reflect the movement of traffic congestion. However, to address this problem, the route-based model is difficult to solve because the number of OD pairs will increase significantly, the number of routes will also increase and the problem in the calculation time and computer memory will appear. In this case, the link-based approach is appropriate.

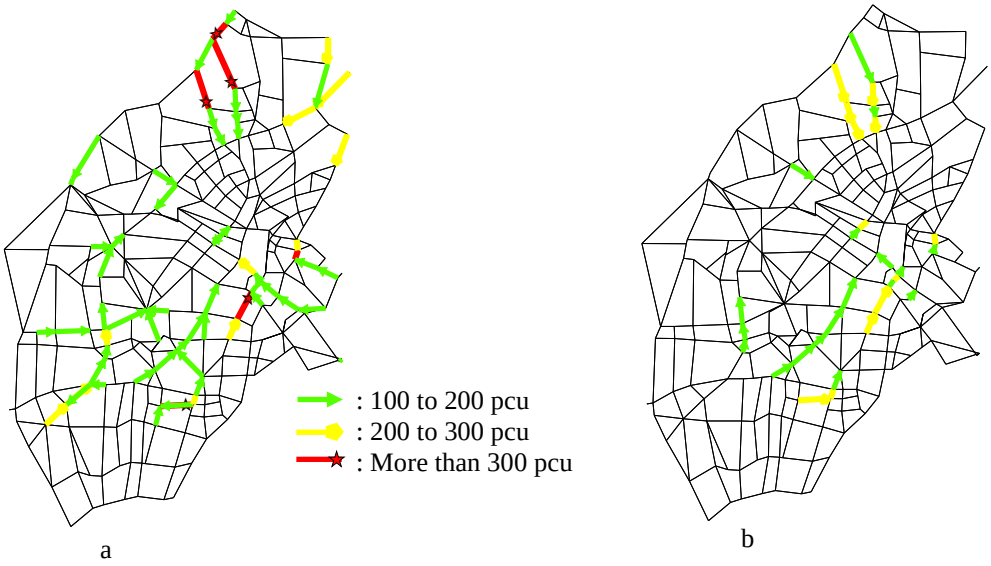


Figure 4.14 Residual link flows between periods of Kanazawa road network. (a) From period 2. (b) From period 3.

Table 4.21 Semi-dynamic OD matrix of the Kanazawa road network

Period	Total number of OD pairs included residual flows of the previous period	Travel demand (pcu)
1	383	10,152
2	5,650	72,587
3	15,298	70,279

References

1. Y. Sheffi, *Urban Transportation Networks: Equilibrium Analysis with Mathematical Programming Methods*. 1985.
2. H. Robbins and S. Monro, "A Stochastic Approximation Method," *Ann. Math. Stat.*, vol. 22, pp. 400–407, 1951.
3. J. R. Blum, "Multidimensional Stochastic Approximation Methods," *Ann. Math. Stat.*, vol. 25, no. 4, pp. 737–744, 1954.
4. M. Maher, "Algorithms for logit-based stochastic user equilibrium assignment," *Transp. Res. Part B*, vol. 32, no. 8, pp. 539–549, Nov. 1998.
5. P. T. T. Ha, S. Nakayama, and J. Takayama, "A semi-dynamic traffic assignment model and its application," *Civ. Eng. Stud. Res. Lect. Collect. vol. 45*, 2012.
6. Y. Itagaki, S. Nakayama, and J. Takayama, "A semi-dynamic traffic assignment model with endogenous traffic congestion using the sensitivity analysis," *J. Japan Soc. Civ. Eng. D3 (Civil Eng. Stud.)*, vol. 70, pp. 569–577, 2014 (in Japanese).
7. F. M. Leurent, "Curbing the Computational Difficulty of the Logit Equilibrium Assignment Model," *Transp. Res. Part B*, vol. 31, no. 4, pp. 315–326, 1997.
8. T. T. Bui, S. Nakayama, and H. Yamaguchi, "Semi-dynamic Traffic Assignment with Sensitivity Analysis Using Link-based STOCH3 algorithm," *Paper presented in the 24th international conference of Hong Kong Society for Transportation Studies*, 2019.
9. T. T. Bui, S. Nakayama, H. Yamaguchi, and K. Koike, "Link-based Approach for Semi-dynamic Stochastic User Equilibrium Traffic Assignment Model with Sensitivity Analysis," *Infrastruct. Plan. Manag.*, vol. 75, no. 5, pp. 615–625, 2019.
10. H. X. Liu, X. He, and B. He, "Method of Successive Weighted Averages (MSWA) and Self-regulated Averaging Schemes for Solving Stochastic User Equilibrium Problem," *Networks Spat. Econ.*, vol. 9, pp. 485–503, 2009.

Chapter 5

Sensitivity analysis method and semi-dynamic stochastic user equilibrium traffic assignment for cross-nested logit model and *q*-generalized logit model

5.1 Overview

Previous chapters solved the size of the route choice set by STOCH3 algorithm[1)] and proposed the sensitivity analysis method and semi-dynamic SUE traffic assignment based on the multinomial logit (MNL) model[2)]. Although widely used, the well-known logit-based traffic assignment assuming an independent distribution and identical variance has also been criticized. Based on McFadden's multivariate extreme value (MEV) theory[3)], the independent distribution assumption, also known as the overlapping problem in route choice context, has been dealt with. For example, the cross-nested logit model (CNL) model[4)5)], the generalized nested logit (GNL) model[6)], and the paired combinatorial logit (PCL) model[7)] are special extensions of McFadden's model. To relax the identical-variance assumption, a single closed-form expression integrating the logit and weibit models, that was proposed by Nakayama[8)] and Nakayama and Chikaraishi[9)]. The CNL model has illustrated the simplicity and flexible correlation structure keeping the closed-form equation and the *q*-generalized logit model has demonstrated generalization and flexibility. However, iteratively solving the SUE problem with the CNL model in the case of the degree of nesting where $0 < \mu < 1$ and the *q*-generalized logit traffic assignments require an explicit route choice set that causes memory waste and large computing time. Moreover, repeated the CNL and the *q*-generalized logit traffic assignments for large-scale-traffic networks require a significant computational cost. To reduce the calculation costs while ensuring the accuracy of traffic-assignment results, this chapter proposes a programming method for sensitivity analysis for the CNL and the *q*-generalized logit traffic assignments. Our method is developed from a link-based-variables approach with an implicit route setting using the DFS algorithm by using the STOCH3-efficient route definition. Besides, while both computational cost and accuracy are assured, the derivative of the link flow

concerning some parameters has been effectively calculated. This chapter will also develop a semi-DTA model based on the CNL model and a semi-DTA model based on the *q*-generalized logit model. Because the computational cost for the equilibrium solution of this model is exceptionally large, we propose the approximation method using the sensitivity analysis method. Only link-based and node-based variables are used in the calculation process to achieve computational efficiency while ensuring the accuracy of the method.

This chapter will reuse the symbols in Chapters 2, 3, and 4. The chapter is organized as follows. Firstly, a sensitivity analysis method for the CNL model will be shown in the next section. In Section 5.3, we formulate the sensitivity analysis method for the *q*-generalized logit model. To shorten calculation time and save memory for semi-dynamic SUE traffic assignment for the CNL model and the *q*-generalized logit model. Section 5.4 presents an efficient sensitivity analysis method for solving these semi-DTA models. The effectiveness of the proposed methods will be verified in the case study of the simple road networks and Kanazawa road network.

5.2 Sensitivity analysis method for the cross-nested logit model

5.2.1 Cross-nested logit model with Depth-first Search algorithm

The overview of the CNL model for route choice context was shown in Section 2.2.3 of Chapter 2. The problem of fixed-point with route traffic flow was also considered in **Equation (2.27)**. In general, the choice-set generation has been considered as a preferred solution. Regarding route-set generation, there are two main methods including *k*-shortest routes and deterministic-based generation in the route-based approach[10]. However, the route-set needs to be stored to compute the probability of choosing each route, causing a memory-occupancy problem. Also, the MSA algorithm with predetermined step-size requires a very large number of iterations to converge. This section will solve these problems with the use of the DFS loading procedure based on the STOCH3-efficient-route definition proposed in Chapter 3 and the self-regulated averaging method (SRA)[11] introduced in Chapter 4.

Instead of calculating the probability of selecting a route, the probability of selecting a link is considered because after all, the purpose of traffic assignment is to know the flow and travel time on the links. By the SUE condition, the probability of choosing link ij is the total of the probability of choosing route k containing link ij and given by the following expression:

$$p_{ij}^{rs} = \sum_{k \in K^{rs}} p_k^{rs} \, \delta_{ij,k}^{rs} \tag{5.1}$$

where p_{ij}^{rs} is the probability of choosing link ij between OD pair rs, $\delta_{ij,k}^{rs}$, p_k^{rs} and K^{rs} symbols are the same as the symbols introduced in Chapter 2 where p_k^{rs} could be calculated by using **Equation (2.27)**. Denoting K_{ij}^{rs} is the set of routes connecting OD pair rs that consist of link ij, we do not need to consider $\delta_{ij,k}^{rs}$ for simplicity, **Equation (5.1)** becomes

$$p_{ij}^{rs} = \sum_{k \in K_{ij}^{rs}} p_k^{rs}$$

$$= \frac{\sum_{k \in K_{ij}^{rs}} \sum_{mn \in A} \left(\left\{ \alpha_{mn,k}^{rs} \exp(-\theta c_k^{rs}) \right\}^{\frac{1}{\mu}} \left[\sum_{p \in K_{mn}^{rs}} \left\{ \alpha_{mn,p}^{rs} \exp\left(-\theta c_p^{rs}\right) \right\}^{\frac{1}{\mu}} \right]^{\mu-1} \right)}{\sum_{mn \in A} \left[\sum_{k \in K_{mn}^{rs}} \left\{ \alpha_{mn,k}^{rs} \exp(-\theta c_k^{rs}) \right\}^{\frac{1}{\mu}} \right]^{\mu}} \quad (5.2)$$

The link flow is represented as a function of p_{ij}^{rs}:

$$x_{ij} = \sum_{rs \in W} Q_0^{rs} p_{ij}^{rs}. \quad (5.3)$$

As shown in Chapter 2, if the link travel time is the function of link flow, the route travel time is also the function of link flow. Thus, p_{ij}^{rs} is also the function of the link flow from **Equation (5.2)**. Thus, we need to solve the fixed-point problem with link flow as follows:

$$x_{ij} = \sum_{rs \in W} Q_0^{rs} p_{ij}^{rs}(\mathbf{x}). \quad (5.4)$$

Chapters 2 and 3 presented the "STOCH3-efficient routes" definition and confirmed that if the route choice set includes routes defined as "STOCH3-efficient routes", we can use the DFS algorithm to explicit all these routes. From there, we think about the use of DFS algorithm to calculate **Equation (5.2)** without considering route-based variables.

The difficulty, here, in the calculating p_{ij}^{rs} without route-based variables that can be divided into two parts:

$$u_{1,ij,k}^{rs} = \left\{ \alpha_{ij,k}^{rs} \exp(-\theta c_k^{rs}) \right\}^{\frac{1}{\mu}}, \quad (5.5)$$

$$u_{2,ij}^{rs} = \sum_{k \in K_{ij}^{rs}} u_{1,ij,k}^{rs} = \sum_{k \in K_{ij}^{rs}} \left\{ \alpha_{ij,k}^{rs} \exp(-\theta c_k^{rs}) \right\}^{\frac{1}{\mu}}. \quad (5.6)$$

If we use the two above denotations, p_{ij}^{rs} becomes

$$p_{ij}^{rs} = \frac{\sum_{k \in K_{ij}^{rs}} \sum_{mn \in A} \left[u_{1,mn,k}^{rs} \left(\left[u_{2,mn}^{rs} \right]^{\mu-1} \right) \right]}{\sum_{ij \in A} \left[u_{2,ij}^{rs} \right]^{\mu}}. \quad (5.7)$$

Let we denote $numc_{ij}^{rs}$ and $dnmc^{rs}$ as the numerator and the denominator of **Equation (5.7)**. The calculation of Equation (5.7) could be solved by calculating $numc_{ij}^{rs}$ and $dnmc^{rs}$ that are calculated through $u_{1,ij,k}^{rs}$ and $u_{2,ij}^{rs}$ variables. To calculate $u_{2,ij}^{rs}$, a prerequisite is $u_{1,ij,k}^{rs}$ that has been calculated. $u_{2,ij}^{rs}$ is calculated if we know all the routes passing through link ij, K_{ij}^{rs}, which require running the DFS algorithm across the network that can be stated as follows. If we save $u_{1,ij,k}^{rs}$ variable, it requires a huge amount of memory. So, the DFS algorithm is used for each OD pair without saving $\alpha_{ij,k}^{rs}$, $u_{1,ij,k}^{rs}$ and c_k^{rs} variables. For example, if the following network consists of efficient links in **Figure 5.1** and a network is saved as adjacent links instead of matrices, the DFS algorithm will traverse nodes in the following order: *r*, 1, 2, 3, 4, *s*, 5, 4, *s*.

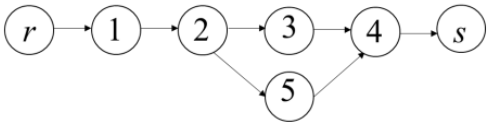


Figure 5.1 A virtual network with an OD pair

As stated in Chapter 3, we can use a one-dimensional array to mark the route from origin *r* to destination *s* that can be reused for saving memory. After first reaching destination *s*, the mark array consists of the links *r*1, 12, 23, 34, 4*s* that represent a route. Then, the algorithm comes back to node 2 because no adjacent links need to be visited from nodes 3 to node *s*. It then continues to chase links 25, 54, and 4*s* to reach destination *s* for the second time. The mark array is reused by removing the backtracked elements (links 4*s*, 34, 23) and updated by adding the new visited links (links 25, 54, and 4*s*), and now consists of links *r*1, 12, 25, 54, 4*s*, which represent another route. With a recursive technique, we can reach the destination node *s* by $|K^{rs}|$ times that represent the number of efficient routes in route-set and we know all the routes passing through link ij.

Some variables will be reused from chapter 3: Ω_{ij}^{rs} denotes indicator variable

($\Omega_{ij}^{rs} \coloneqq 1$ if link ij is efficient from r to s and $\Omega_{ij}^{rs} := 0$ otherwise) that can be defined by STOCH3-efficient-route (see Chapters 2 and 3); a_{ij} is the likelihood of link ij $a_{ij} = \exp(-\theta t_{ij})$; $mark_{ij}$ array to mark the efficient route in the DFS algorithm and node-based variable, vn_n^{rs}. The process of calculating $numc_{ij}^{rs}$ and $dnmc^{rs}$ as follows:

- For each link ij, set $a_{ij} \coloneqq \exp(-\theta t_{ij}), u_{2,ij}^{rs} = 0$. We can run the DFS algorithm from origin node r to destination node s with the recursive technique as follows: Set all vn_n^{rs} to 0 and vn_r^{rs}:=1. After each link and node with the nonzero value of Ω_{ij}^{rs} (efficient link) is visited (see Chapter 3) we update $vn_j^{rs} = a_{ij} vn_i^{rs}$. When reaching destination node *s*, we know all the links visited from node *r* to node *s* that is exactly information of an efficient route k and $vn_s^{rs} = \exp(-\theta c_k^{rs})$ (see **Equation (3.15)** in Chapter 3). Each time we reach the destination node, we have information about links that belong to an efficient route through mark array, $mark_{ij}$, we can calculate $\alpha_{ij,k}^{rs}$ each time *s* is reached and use this to calculate $u_{1,ij,k}^{rs}$ without saving $\alpha_{ij,k}^{rs}$ variable. $u_{2,ij}^{rs}$ is also calculated at each time reaching s, if link ij is a member of mark array, we add $u_{1,ij,k}^{rs}$ to $u_{2,ij}^{rs}$ without saving $u_{1,ij,k}^{rs}$ variable. When s is reach $|K^{rs}|$ time (total efficient routes), $u_{2,ij}^{rs}$ is completely calculated. When the DFS algorithm stops, $dnmc^{rs}$ is computed by summing $u_{2,ij}^{rs}$ for all links ij in the network.
- To calculate $numc_{ij}^{rs}$, we must have the result of $u_{2,ij}^{rs}$ that is only completely computed when the DFS algorithm stops, i.e. all efficient routes are considered. Thus, we must run the DFS algorithm again to calculate $numc_{ij}^{rs}$. We set $numc_{ij}^{rs} = 0$ and continue running the DFS algorithm for the second time. vn_s^{rs} is calculated by using the same procedure as the first time. Note that $u_{1,mn,k}^{rs} = 0$ if route k does not consist of link mn. Thus, we only need to calculate $u_{1,mn,k}^{rs}$ for all links mn belonging to mark array, $mark_{ij}$. At each time reaching destination node *s*, we calculate $u_{1,gh,k}^{rs}\left(\left[u_{2,gh}^{rs}\right]^{\mu-1}\right)$ that is summed up all the links gh in the mark array and added to $numc_{ij}^{rs}$ if link ij is

a member of mark array. When s is reach $|K^{rs}|$ time, $numc_{ij}^{rs}$ is completely calculated.

From the above techniques, we do not need to use route variables to calculate **Equation (5.7)** and because it is calculated for each OD pair, we can reuse variables for each calculation loop to save memory. In a large-scale network, while the number of links is fixed, the number of routes will increase rapidly when the number of OD pairs increases. Therefore, using link-based variables and calculating link flows will save memory.

The DFS loading procedure using STOCH3-efficient-route definition with the SRA method for the CNL traffic assignment is shown as follows:

Step 0: Preliminaries.

(a) Set current iteration counter $l := 1$, $\eta \in [1.5,2]$, $\gamma \in [0.01,0.5]$, $\vartheta^{(l-1)} = 0$, and convergence test value σ.

(b) Define reasonable links by setting the value of Ω_{ij}^{rs}.

(c) Find an initial feasible flow pattern based on free flow link times $\left(t_{ij}^{0}\right)$:

- For each link ij, set $t_{ij} = t_{ij}^{0}$, $a_{ij} \coloneqq \exp(-\theta t_{ij})$ and, $x_{ij}^{(l)}$=0.

- Performing the DFS algorithm from origin node r to destination node s with recursive technique twice to calculate p_{ij}^{rs}.

- Contribution to total link-flows as follows:

$$x_{ij}^{(l)} \coloneqq x_{ij}^{(l)} + Q_0^{rs} p_{ij}^{rs}. \tag{5.8}$$

Step 1: Link travel time and link impedances update.

(a) Set $l := l + 1$.

(b) Update $t_{ij}^{(l)} \coloneqq t_{ij}\left(x_{ij}^{(l)}\right)$ and $a_{ij} \coloneqq \exp(-\theta\, t_{ij}^{(l)})$.

Step 2: Direction finding.

Based on link travel time $\{t_{ij}^{(l)}\}$ and a_{ij}, running DFS algorithm from origin node r to destination node s twice to update p_{ij}^{rs}. When the DFS

algorithm stop, the auxiliary link flow is yielded as the following equation:

$$yx_{ij}^{(l)} \coloneqq yx_{ij}^{(l)} + Q_0^{rs} p_{ij}^{rs} \tag{5.9}$$

Step 3: Link flow update

(a) Calculate $\vartheta^{(l)}$ as follows:

$$\vartheta^{(l)} = \begin{cases} \vartheta^{(l-1)} + \eta, \text{if } \left\|\mathbf{x}^{(l)} - \mathbf{yx}^{(l)}\right\| \geq \left\|\mathbf{x}^{(l-1)} - \mathbf{yx}^{(l-1)}\right\| \\ \vartheta^{(l-1)} + \gamma, \text{if } \left\|\mathbf{x}^{(l)} - \mathbf{yx}^{(l)}\right\| < \left\|\mathbf{x}^{(l-1)} - \mathbf{yx}^{(l-1)}\right\|. \end{cases} \tag{5.10}$$

(b) Let $rg_{ij}^{(l)} = yx_{ij}^{(l)} - x_{ij}^{(l)}$.

(c) Set $x_{ij}^{(l+1)} = x_{ij}^{(l)} + \frac{1}{\vartheta^{(l)}} rg_{ij}^{(l)}$.

Step 4: Stopping the test.

If $max_{ij}\{|rg_{ij}^{(l)}|\} \leq \sigma$, stop. The solution is $\{x_{ij}^{(l)}\}$. Otherwise, go to step 2.

In the proposed method, we do not need to save the route choice set to reduce computational storage. Also, the DFS algorithm with recursive solving tree data structures helps reduce computational time.

5.2.2 Sensitivity analysis for the cross-nested logit model

In many cases, we must repeat the calculation of traffic equilibrium with the CNL model for route choice because of the requirements of the optimal model. The reduction of calculation time is very important, this is solved by sensitivity analysis. How the change of parameters will affect the links flow in the model is shown by calculating the derivative of the link flow according to the change of parameters. Here we consider the change of the link flow according to the change of toll pricing. Other cases are calculated similarly.

Using flow and toll pricing-dependent BPR function, a link travel time on link ij is given by

$$t_{ij}(x_{ij}, \pi_{ij}) = t_{ij}^0 \left[1 + \alpha \left(\frac{x_{ij}}{Cap_{ij}}\right)^{\beta}\right] + \frac{\pi_{ij}}{\varphi}, \tag{5.11}$$

where π_{ij} is toll fare on link ij and φ is the value of time.

Now, the problem of the CNL model becomes the following fixed-point problem:

$$x_{ij} = \sum_{rs \in W} Q_0^{rs} p_{ij}^{rs}\left(\mathbf{t}(\mathbf{x}, \boldsymbol{\pi})\right). \tag{5.12}$$

where $\mathbf{t}(\mathbf{x}, \boldsymbol{\pi}) = \left(t_{12}(\mathbf{x}, \boldsymbol{\pi}), \dots, t_{ij}(\mathbf{x}, \boldsymbol{\pi}), \dots, t_{mn}(\mathbf{x}, \boldsymbol{\pi})\right)^T$ is the vector-valued function of link travel time with respect $\mathbf{x}$ and $\boldsymbol{\pi}$. Besides, $\mathbf{x} = (x_{12}, \dots, x_{ij}, \dots, x_{mn})^T$ and $\boldsymbol{\pi} = (\pi_{12}, \dots, \pi_{ij}, \dots, \pi_{mn})^T$ are the vector forms of link traffic flow and link toll fare, respectively.

Denoting the right-hand side of the above equation by h_{ij}. h_{ij} is a function of $\mathbf{x}$ and $\boldsymbol{\pi}$. We can define the following function:

$$\mathbf{d}(\mathbf{x}, \boldsymbol{\pi}) = \mathbf{x} - \mathbf{h}\left(\mathbf{t}(\mathbf{x}, \boldsymbol{\pi})\right). \tag{5.13}$$

where $\mathbf{h}\left(\mathbf{t}(\mathbf{x}, \boldsymbol{\pi})\right) = \left(h_{12}\left(\mathbf{t}(\mathbf{x}, \boldsymbol{\pi})\right), \dots, h_{ij}\left(\mathbf{t}(\mathbf{x}, \boldsymbol{\pi})\right), \dots, h_{mn}\left(\mathbf{t}(\mathbf{x}, \boldsymbol{\pi})\right)\right)^T$ and $\mathbf{d}(\mathbf{x}, \boldsymbol{\pi}) = \left(d_{12}(\mathbf{x}, \boldsymbol{\pi}), \dots, d_{ij}(\mathbf{x}, \boldsymbol{\pi}), \dots, d_{mn}(\mathbf{x}, \boldsymbol{\pi})\right)^T$ are the vector-valued functions of h_{ij} and d_{ij}. The gap at both sides of the above equation should be zero at the equilibrium state. If we know the equilibrium solution at any value of toll fare of highway link, the problem is the calculation of partial derivatives of link flow concerning the change of toll fare. Denoting $\boldsymbol{\nabla}_{\mathbf{x}}\mathbf{d}$, $\boldsymbol{\nabla}_{\mathbf{x}}\mathbf{h}$ and $\boldsymbol{\nabla}_{\mathbf{x}}\mathbf{t}$ are the matrix of partial derivatives of d_{ij}, h_{ij} and t_{ij} with respect to x_{ij}, $\boldsymbol{\nabla}_{\boldsymbol{\pi}}\mathbf{d}$, $\boldsymbol{\nabla}_{\boldsymbol{\pi}}\mathbf{h}$ and $\boldsymbol{\nabla}_{\boldsymbol{\pi}}\mathbf{t}$ are the matrix of partial derivatives of d_{ij}, h_{ij} and t_{ij} with respect to π_{ij}. By applying the general formulation of derivative of the implicit function and the chain rule of differentiation, $\boldsymbol{\nabla}_{\boldsymbol{\pi}}\mathbf{x}$ is calculated as follows:

$$\begin{aligned}
\boldsymbol{\nabla}_{\boldsymbol{\pi}}\mathbf{x} &= -\boldsymbol{\nabla}_{\mathbf{x}}\mathbf{d}^{-1}\boldsymbol{\nabla}_{\boldsymbol{\pi}}\mathbf{d} \\
&= -\left(\boldsymbol{\nabla}_{\mathbf{x}}\left[\mathbf{x} - \mathbf{h}\left(\mathbf{t}(\mathbf{x}, \boldsymbol{\pi})\right)\right]\right)^{-1}\boldsymbol{\nabla}_{\boldsymbol{\pi}}\left(\mathbf{x} - \mathbf{h}\left(\mathbf{t}(\mathbf{x}, \boldsymbol{\pi})\right)\right) \\
&= -(\mathbf{I} - \boldsymbol{\nabla}_{\mathbf{t}}\mathbf{h}\boldsymbol{\nabla}_{\mathbf{x}}\mathbf{t})^{-1}(-\boldsymbol{\nabla}_{\mathbf{t}}\mathbf{h}\boldsymbol{\nabla}_{\boldsymbol{\pi}}\mathbf{t}) \\
&= (\mathbf{I} - \boldsymbol{\nabla}_{\mathbf{t}}\mathbf{h}\boldsymbol{\nabla}_{\mathbf{x}}\mathbf{t})^{-1}(\boldsymbol{\nabla}_{\mathbf{t}}\mathbf{h}\boldsymbol{\nabla}_{\boldsymbol{\pi}}\mathbf{t}),
\end{aligned} \tag{5.14}$$

where $\boldsymbol{\nabla}_{\boldsymbol{\pi}}\mathbf{x}$ denotes the matrix of partial derivatives of link flow with respect to toll pricing

$$\boldsymbol{\nabla_{\pi}\mathbf{x}} = \begin{pmatrix} \frac{\partial x_{12}}{\partial \pi_{12}} & \cdots & \frac{\partial x_{12}}{\partial \pi_{ij}} & \cdots & \frac{\partial x_{12}}{\partial \pi_{mn}} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \frac{\partial x_{ij}}{\partial \pi_{12}} & \cdots & \frac{\partial x_{ij}}{\partial \pi_{ij}} & \vdots & \frac{\partial x_{ij}}{\partial \pi_{mn}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial x_{mn}}{\partial \pi_{12}} & \cdots & \frac{\partial x_{mn}}{\partial \pi_{ij}} & \cdots & \frac{\partial x_{mn}}{\partial \pi_{mn}} \end{pmatrix}. \quad (5.15)$$

To calculate $\boldsymbol{\nabla_{\pi}\mathbf{x}}$, we need to calculate $\boldsymbol{\nabla_{\mathbf{x}}\mathbf{t}}$, $\boldsymbol{\nabla_{\pi}\mathbf{t}}$ and $\boldsymbol{\nabla_{\mathbf{t}}\mathbf{g}}$. From **Equation (5.11)**, it is clear that $\boldsymbol{\nabla_{\mathbf{x}}\mathbf{t}}$ and $\boldsymbol{\nabla_{\pi}\mathbf{t}}$ are diagonal matrices with diagonal elements are represented as follows

$$\frac{\partial t_{ij}}{\partial x_{ij}} = \frac{\partial \left(t_{ij}^{0} \left[1 + \alpha \left(\frac{x_{ij}}{Cap_{ij}} \right)^{\beta} \right] + \frac{\pi_{ij}}{\varphi} \right)}{\partial x_{ij}} = \frac{t_{ij}^{0} \alpha \beta {x_{ij}}^{\beta-1}}{{Cap_{ij}}^{\beta}}, \quad (5.16)$$

$$\frac{\partial t_{ij}}{\partial \pi_{ij}} = \frac{\partial \left(t_{ij}^{0} \left[1 + \alpha \left(\frac{x_{ij}}{Cap_{ij}} \right)^{\beta} \right] + \frac{\pi_{ij}}{\varphi} \right)}{\partial \pi_{ij}} = \frac{1}{\varphi}, \quad (5.17)$$

The forms of $\boldsymbol{\nabla_{\mathbf{x}}\mathbf{t}}$, $\boldsymbol{\nabla_{\pi}\mathbf{t}}$ and $\boldsymbol{\nabla_{\mathbf{t}}\mathbf{g}}$ are shown as follows:

$$\boldsymbol{\nabla_{\mathbf{x}}\mathbf{t}} = \begin{pmatrix} \frac{\partial x_{12}}{\partial t_{12}} & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & \frac{\partial x_{ij}}{\partial t_{ij}} & \vdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & \frac{\partial x_{mn}}{\partial t_{mn}} \end{pmatrix}, \quad (5.18)$$

$$\boldsymbol{\nabla_{\pi}\mathbf{t}} = \begin{pmatrix} \frac{1}{\varphi} & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & \frac{1}{\varphi} & \vdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & \frac{1}{\varphi} \end{pmatrix} = \frac{1}{\varphi}\mathbf{I}, \quad (5.19)$$

$$\nabla_{\mathbf{t}}\mathbf{h} = \begin{pmatrix} \frac{\partial h_{12}}{\partial t_{12}} & \cdots & \frac{\partial h_{12}}{\partial t_{ij}} & \cdots & \frac{\partial h_{12}}{\partial t_{mn}} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \frac{\partial h_{ij}}{\partial t_{12}} & \cdots & \frac{\partial h_{ij}}{\partial t_{ij}} & \vdots & \frac{\partial h_{ij}}{\partial t_{mn}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial h_{mn}}{\partial t_{12}} & \cdots & \frac{\partial h_{mn}}{\partial t_{ij}} & \cdots & \frac{\partial h_{mn}}{\partial t_{mn}} \end{pmatrix}. \quad (5.20)$$

Elements of $\nabla_{\mathbf{t}}\mathbf{h}$ are calculated by the generalized form

$$\frac{\partial h_{ij}}{\partial t_{gh}} = \frac{\partial \sum_{rs \in W} Q_0^{rs} p_{ij}^{rs}}{\partial t_{gh}} = \sum_{rs \in W} Q_0^{rs} \frac{\partial p_{ij}^{rs}}{\partial t_{gh}} \quad (5.21)$$

Applying the derivative rule of the quotient (see **Appendix B**), we have

$$\frac{\partial p_{ij}^{rs}}{\partial t_{gh}} = \frac{\partial \left[\sum_{k \in K_{ij}^{rs}} \sum_{mn \in A} \left[u_{1,mn,k}^{rs} \left(\left[u_{2,mn}^{rs} \right]^{\mu-1} \right) \right] \right]}{\partial t_{gh}} \Big/ \sum_{ij \in A} \left[u_{2,ij}^{rs} \right]^{\mu} - p_{ij}^{rs} \frac{\partial \sum_{ij \in A} \left[u_{2,ij}^{rs} \right]^{\mu}}{\partial t_{gh}} \Big/ \sum_{ij \in A} \left[u_{2,ij}^{rs} \right]^{\mu} \quad (5.22)$$

Using the following denotation:

$$v_{mn,gh}^{rs} = \sum_{k \in K_{mn}^{rs}} u_{1,mn,k}^{rs} \delta_{gh,k}^{rs} = \sum_{k \in K_{mn,gh}^{rs}} u_{1,mn,k}^{rs}. \quad (5.23)$$

where $K_{mn,gh}^{rs}$ denotes the set of routes containing both link mn and gh.

Since p_{ij}^{rs} and the denominator of Equation (5.22) could be calculated by using the DFS algorithm without route-based variables shown in the previous section, the difficulty in the calculating $\frac{\partial p_{ij}^{rs}}{\partial t_{gh}}$ lies in the calculation of the numerator.

Some minor problems are solved first as follows:

$$\begin{aligned}\frac{\partial u_{1,mn,k}^{rs}}{\partial t_{gh}} &= \frac{\partial\left\{\alpha_{mn,k}^{rs}\exp(-\theta c_k^{rs})\right\}^{\frac{1}{\mu}}}{\partial t_{gh}} \\ &= \frac{1}{\mu}\left\{\alpha_{mn,k}^{rs}\exp(-\theta c_k^{rs})\right\}^{\left(\frac{1}{\mu}-1\right)}\frac{\partial\left\{\alpha_{mn,k}^{rs}\exp(-\theta c_k^{rs})\right\}}{\partial t_{gh}} \\ &= \frac{1}{\mu}\left\{\alpha_{mn,k}^{rs}\exp(-\theta c_k^{rs})\right\}^{\left(\frac{1}{\mu}-1\right)}\left\{\alpha_{mn,k}^{rs}\exp(-\theta c_k^{rs})\right\}\frac{\partial(-\theta c_k^{rs})}{\partial t_{gh}} \\ &= \frac{1}{\mu}\left\{\alpha_{mn,k}^{rs}\exp(-\theta c_k^{rs})\right\}^{\frac{1}{\mu}}(-\theta)\delta_{gh,k}^{rs} = \frac{-\theta}{\mu}u_{1,mn,k}^{rs}\delta_{gh,k}^{rs}\end{aligned} \tag{5.24}$$

$$\begin{aligned}\frac{\partial u_{2,mn}^{rs}}{\partial t_{gh}} &= \frac{\partial\sum_{k\in K_{mn}^{rs}}u_{1,mn,k}^{rs}}{\partial t_{gh}} = \sum_{k\in K_{mn}^{rs}}\frac{\partial u_{1,mn,k}^{rs}}{\partial t_{gh}} \\ &= \sum_{k\in K_{mn}^{rs}}\frac{-\theta}{\mu}u_{1,mn,k}^{rs}\delta_{gh,k}^{rs} = \frac{-\theta}{\mu}\sum_{k\in K_{mn}^{rs}}u_{1,mn,k}^{rs}\delta_{gh,k}^{rs} \\ &= \frac{-\theta}{\mu}v_{mn,gh}^{rs}\end{aligned} \tag{5.25}$$

$$\begin{aligned}\frac{\partial\left[u_{2,mn}^{rs}\right]^{\mu-1}}{\partial t_{gh}} &= (\mu-1)\left[u_{2,mn}^{rs}\right]^{\mu-2}\frac{\partial u_{2,mn}^{rs}}{\partial t_{gh}} \\ &= (\mu-1)\left[u_{2,mn}^{rs}\right]^{\mu-2}\left\{\frac{-\theta}{\mu}v_{mn,gh}^{rs}\right\} \\ &= \frac{\mu-1}{\mu}(-\theta)\left[u_{2,mn}^{rs}\right]^{\mu-2}v_{mn,gh}^{rs}\end{aligned} \tag{5.26}$$

$$\begin{aligned}\frac{\partial\left[u_{2,ij}^{rs}\right]^{\mu}}{\partial t_{gh}} &= \mu\left[u_{2,ij}^{rs}\right]^{\mu-1}\frac{\partial u_{2,ij}^{rs}}{\partial t_{gh}} \\ &= \mu\left[u_{2,ij}^{rs}\right]^{\mu-1}\left\{\frac{-\theta}{\mu}v_{ij,gh}^{rs}\right\} \\ &= (-\theta)\left[u_{2,ij}^{rs}\right]^{\mu-1}v_{ij,gh}^{rs}\end{aligned} \tag{5.27}$$

Each element of **Equation (5.21)** is solved as

$$\frac{\partial\left[\sum_{k\in K_{ij}^{rs}}\sum_{mn\in A}\left[u_{1,mn,k}^{rs}\left(\left[u_{2,mn}^{rs}\right]^{\mu-1}\right)\right]\right]}{\partial t_{gh}}=\sum_{k\in K_{ij}^{rs}}\sum_{mn\in A}\frac{\partial\left[u_{1,mn,k}^{rs}\left(\left[u_{2,mn}^{rs}\right]^{\mu-1}\right)\right]}{\partial t_{gh}}$$

$$=\sum_{k\in K_{ij}^{rs}}\sum_{mn\in A}\left[\frac{\partial u_{1,mn,k}^{rs}}{\partial t_{gh}}\left(\left[u_{2,mn}^{rs}\right]^{\mu-1}\right)+\frac{\partial\left(\left[u_{2,mn}^{rs}\right]^{\mu-1}\right)}{\partial t_{gh}}\left(u_{1,mn,k}^{rs}\right)\right] \tag{5.28}$$

$$=\sum_{k\in K_{ij}^{rs}}\sum_{mn\in A}\left[\left(\frac{-\theta}{\mu}u_{1,mn,k}^{rs}\delta_{gh,k}^{rs}\right)\left(\left[u_{2,mn}^{rs}\right]^{\mu-1}\right)+\left(\frac{\mu-1}{\mu}(-\theta)\left[u_{2,mn}^{rs}\right]^{\mu-2}v_{mn,gh}^{rs}\right)\left(u_{1,mn,k}^{rs}\right)\right]$$

$$\frac{\partial\sum_{ij\in A}\left[u_{2,ij}^{rs}\right]^{\mu}}{\partial t_{gh}}=\sum_{ij\in A}\frac{\partial\left[u_{2,ij}^{rs}\right]^{\mu}}{\partial t_{gh}}=\sum_{ij\in A}(-\theta)\left[u_{2,ij}^{rs}\right]^{\mu-1}v_{ij,gh}^{rs} \tag{5.29}$$

After having the formula, the remaining difficulty lies in the calculation technique when using information brought from the DFS algorithm. Adopting the DFS algorithm from origin r to destination s shown in Section 5.2.1. At each time of reaching destination node s, we know all the links included each route k by saving the route in an array, that make $u_{1,ij,k}^{rs}$ easily computed, and directly add $u_{1,ij,k}^{rs}$ to $u_{2,ij}^{rs}$ if link ij is a member of mark array and $v_{ij,gh}^{rs}$ if both link ij and gh are members of mark array. When the DFS algorithm is stopped with the recursive technique, we have the results of $u_{2,ij}^{rs}$, $v_{ij,gh}^{rs}$ for all links of the network and the denominator of **Equation (5.22)**. Also, **Equation (5.29)** is computed because $u_{2,ij}^{rs}$, $v_{ij,gh}^{rs}$ are computed.

At the second algorithm running, we calculate p_{ij}^{rs} that is presented in Section 5.2.1 and **Equations (5.28), (5.29)**.

The first part of **Equation (5.28)** is denoted as $p_{1,ij,gh}^{rs}$. If route k does not include link gh, $\delta_{gh,k}^{rs}=0$. We only need to compute $p_{1,ij,gh}^{rs}$ for the routes including both links ij and gh. Also, $\delta_{mn,k}^{rs}=0$ and $u_{1,mn,k}^{rs}=0$ if link mn does not belong to route k. Thus, $p_{1,ij,gh}^{rs}$ is also need to computed by summing up $\left(\frac{-\theta}{\mu}u_{1,mn,k}^{rs}\right)\left(\left[u_{2,mn}^{rs}\right]^{\mu-1}\right)$ of links mn that also belongs to the route k containing

both links ij, gh. Hence, $p_{1,ij,gh}^{rs}$ could be presented as

$$p_{1,ij,gh}^{rs} = \sum_{k \in K_{ij}^{rs}} \sum_{mn \in A} \left[\left(\frac{-\theta}{\mu} u_{1,mn,k}^{rs} \delta_{gh,k}^{rs} \right) \left(\left[u_{2,mn}^{rs} \right]^{\mu-1} \right) \right] = \sum_{k \in K_{ij,gh,mn}^{rs}} \sum_{mn \in A_k^{rs}} \left[\left(\frac{-\theta}{\mu} u_{1,mn,k}^{rs} \right) \left(\left[u_{2,mn}^{rs} \right]^{\mu-1} \right) \right] \quad (5.30)$$

where $K_{ij,gh,mn}^{rs}$ is the set of routes including triple links ij, gh, mn of OD pair rs, A_k^{rs} is the set of links that are parts of route k of OD pair rs. As highlighted in Section 5.2.1, $u_{2,mn}^{rs}$ for all links mn could be completely computed after running the DFS algorithm for the first time. By using the DFS algorithm for the second time, at each time of reaching destination node s, we know the information of an efficient route including efficient links in the mark array. $\left(\frac{-\theta}{\mu} u_{1,mn,k}^{rs} \right) \left(\left[u_{2,mn}^{rs} \right]^{\mu-1} \right)$ is computed for all links mn in the mark array, summed up and added to $p_{1,ij,gh}^{rs}$ of all links ij and gh in mark array. After the DFS algorithm stops for the second time, $p_{1,ij,gh}^{rs}$ is completely computed for all links ij and gh.

The second part of **Equation (5.28)** is denoted as $p_{2,ij,gh}^{rs}$. If route k does not include link mn, $u_{1,mn,k}^{rs} = 0$. We only need to compute $p_{1,ij,gh}^{rs}$ by summing up $\left(\frac{\mu-1}{\mu} (-\theta) \left[u_{2,mn}^{rs} \right]^{\mu-2} v_{mn,gh}^{rs} \right) \left(u_{1,mn,k}^{rs} \right)$ of links mn that also belongs to the route k containing links ij. Hence, $p_{2,ij,gh}^{rs}$ could be presented as

$$p_{2,ij,gh}^{rs} = \sum_{k \in K_{ij}^{rs}} \sum_{mn \in A} \left(\frac{\mu - 1}{\mu} (-\theta) \left[u_{2,mn}^{rs} \right]^{\mu-2} v_{mn,gh}^{rs} \right) \left(u_{1,mn,k}^{rs} \right) = \sum_{k \in K_{ij,mn}^{rs}} \sum_{mn \in A_k^{rs}} \left(\frac{\mu - 1}{\mu} (-\theta) \left[u_{2,mn}^{rs} \right]^{\mu-2} \right) \left(u_{1,mn,k}^{rs} \right) v_{mn,gh}^{rs}, \quad (5.31)$$

This part could be computed in the second time using the DFS algorithm. At each reaching destination node s, an efficient route is represented in the mark array. For each link ij in the mark array, we compute $p_{2,ij,gh}^{rs}$ for all link gh by summing up $\left(\frac{\mu-1}{\mu} (-\theta) \left[u_{2,mn}^{rs} \right]^{\mu-2} \right) \left(u_{1,mn,k}^{rs} \right) v_{mn,gh}^{rs}$ of all links mn in mark array.

After the DFS algorithm stops, $p_{2,ij,gh}^{rs}$ is completely computed for all links ij and gh. Also, **Equation (5.29)** is computed after

After all, the process of calculating $\boldsymbol{\nabla_{\pi}\mathbf{x}}$ is presented in the following steps:

Step 1: Performing the DFS loading procedure using the STOCH3-efficient-route definition with the SRA method shown in Section 5.2.1 to achieve the SUE CNL solution.

Step 2: At the initial SUE solution with the values of $\{t_{ij}\}$ and $\{x_{ij}\}$, calculating $\boldsymbol{\nabla_{\mathbf{x}}\mathbf{t}}$ and $\boldsymbol{\nabla_{\pi}\mathbf{t}}$ by using **Equations (5.16) – (5.19)**. Set $a_{ij} \coloneqq \exp(-\theta t_{ij})$. For each OD pair rs, running the DFS algorithm twice. After the first running of the DFS algorithm, we can compute $u_{2,ij}^{rs}$, $v_{ij,gh}^{rs}$ for all links of the network, the denominator of **Equation (5.22),** and **Equation (5.29)**. Other parts of $\frac{\partial p_{ij}^{rs}}{\partial t_{gh}}$ is computed after the second running of the DFS algorithm. After all OD pairs are considered, $\boldsymbol{\nabla_{\mathbf{t}}\mathbf{g}}$ is totally computed by summing over the index rs.

Step 3: Compute $\boldsymbol{\nabla_{\pi}\mathbf{x}}$ by using **Equation (5.14).**

5.2.3 Applications

The following 5 nodes, 7 links, 1 OD pair virtual network shown in **Figure 5.2** is used to demonstrate the effectiveness of the proposed method. The dotted lines are the general roads and the solid line represents the expressway. The OD demand is 1000 pcu and the toll fare of link 13 is 500 yen. The parameters of the BPR function are $\alpha = 1$, $\beta = 2$, $\varphi = 50$ yen/min. The CNL parameter θ is 0.5 and μ is 0.5. The link parameters are shown in **Table 5.1** and the information of route choice set and nest are depicted in **Figure 5.2**.

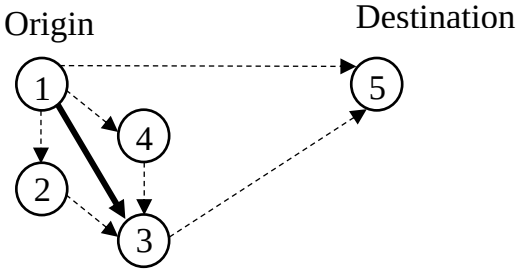


Figure 5.2 A virtual network

Table 5.1 The link parameters

No.	Link	Free-flow travel time (min)	Capacity (pcu)
1	12	10	150
2	13	5	100
3	14	5	150
4	15	30	500
5	23	10	125
6	35	20	500
7	43	5	125

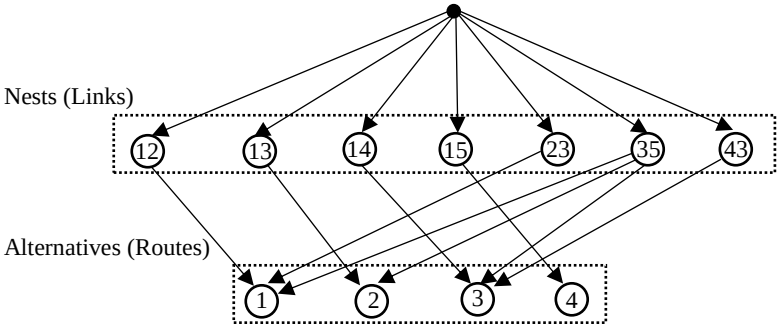


Figure 5.3 Cross-nested structure of the virtual network

The results of explicit route flow at SUE state using the CNL model with the MSA method are represented by the following table:

Table 5.2 The results of route flow of the virtual network

No.	Route	Route flow (pcu)	Probability of choosing the route
1	1→2→3→5	98.586	0.10
2	1→3→5	171.777	0.17
3	1→4→3→5	189.918	0.19
4	1→5	539.718	0.54

If we do not use the route-based approach to calculate SUE model, the DFS algorithm is used to calculate p_{ij}^{rs} and solve the route choice set implicitly. The results of the link flow are represented by the below table:

Table 5.3 The results of link flow of the virtual network

No.	Link	Link flow (pcu)	Probability of choosing the link
1	12	98.586	0.10
2	13	171.777	0.17
3	14	189.919	0.19
4	15	539.716	0.54
5	23	98.586	0.10
6	35	460.282	0.46
7	43	189.919	0.19

After using the SA method, we have the result of $\boldsymbol{\nabla_{\pi}\mathbf{x}}$ as follows:

Table 5.4 The results of derivative of link flow with respect to toll fare of link 13 of the virtual network

Link / Link	13
12	0.02
13	-0.08
14	0.03
15	0.03
23	0.02
35	-0.03
43	0.03

As can be seen, because we consider that the expressway includes only link 13, the change of link flow only depends on the toll fare of link 13. We also assume that toll fare of link 13 ranges from 100 to 1000. We compare the results of the route-based SUE CNL model (RBSUESCNL) and the proposed sensitivity analysis method for the CNL model (SACNL). The former is recomputed at each toll fare and the latter is calculated from $\nabla_{\pi}\mathbf{x}$. **Table 5.5** shows the results of the comparison and **Figure 5.4** depicts the comparison in the expressway road. The results of link flow may be the same between two methods with the small value of two indicators *RMSE* and *%RMS*. However, with the SA method, we will save computational time because we do not need to recompute the traffic assignment. Based on the initial traffic assignment result, we only need to calculate $\nabla_{\pi}\mathbf{x}$. In this case, the total saving time is up to 88.6% and we also save the memory because we do not need to explicit route choice set. In the real road network with a huge amount of OD pairs and routes connecting each OD pair, the effectiveness of the SA method will exponentially increase.

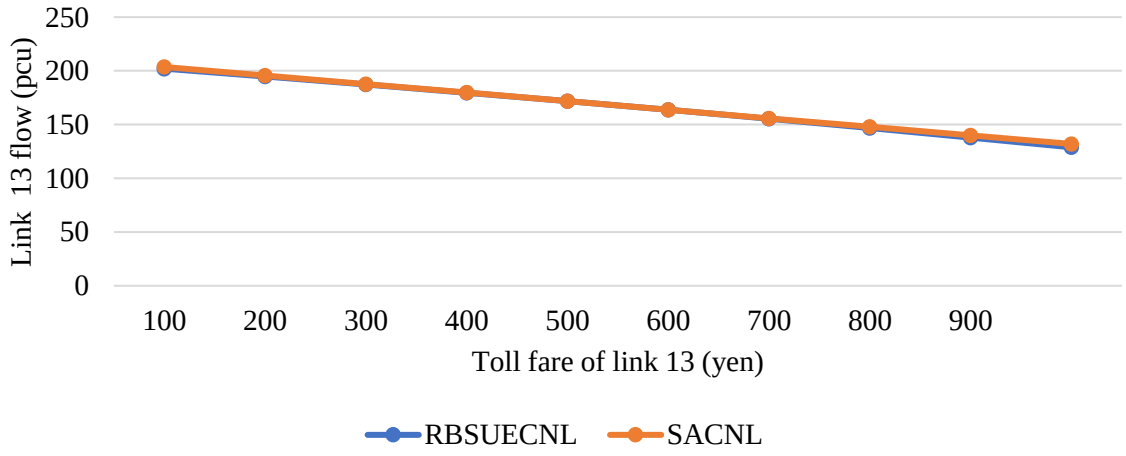


Figure 5.4 The comparison between two methods in the link 13 flow depending on toll fare of link 13

Table 5.5 The comparison between two methods when toll fare of link 13 changes from 100 yen to 1000 yen

Toll fare of link 13 (yen)			**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**
RBSUECNL	Link flow (pcu)	Link 12	88.930	91.291	93.686	96.116	98.586	101.099	103.660	106.274	108.946	111.683
		Link 13	202.050	194.754	187.286	179.631	171.777	163.707	155.404	146.845	138.009	128.869
		Link 14	179.551	182.055	184.615	187.235	189.919	192.671	195.498	198.404	201.398	204.487
		Link 15	529.468	531.897	534.411	537.015	539.716	542.520	545.437	548.475	551.644	554.958
		Link 23	88.930	91.291	93.686	96.116	98.586	101.099	103.660	106.274	108.946	111.683
		Link 35	470.530	468.101	465.587	462.983	460.282	457.478	454.561	451.523	448.354	445.040
		Link 43	179.551	182.055	184.615	187.235	189.919	192.671	195.498	198.404	201.398	204.487
	Required iteration		105116	104779	104393	103950	103446	102871	102215	101466	100609	99624
	Calculation time (sec)		0.094	0.078	0.094	0.078	0.078	0.078	0.078	0.078	0.094	0.078
	Total calculation time (sec)		0.828									
SACNL	Link flow (pcu)	Link 12	88.623	91.114	93.605	96.095	98.586	101.077	103.567	106.058	108.549	111.040
		Link 13	203.614	195.655	187.696	179.737	171.778	163.819	155.859	147.900	139.941	131.982
		Link 14	179.050	181.768	184.485	187.202	189.919	192.636	195.353	198.071	200.788	203.505
		Link 15	528.712	531.463	534.215	536.966	539.717	542.468	545.220	547.971	550.722	553.473
		Link 23	88.623	91.114	93.605	96.095	98.586	101.077	103.567	106.058	108.549	111.040
		Link 35	471.288	468.537	465.785	463.034	460.283	457.532	454.780	452.029	449.278	446.527
		Link 43	179.050	181.768	184.485	187.202	189.919	192.636	195.353	198.071	200.788	203.505
	Total calculation time (sec)		0.094									
RMSE			0.782	0.450	0.205	0.053	0.001	0.055	0.227	0.526	0.963	1.552
%RMS			0.315	0.181	0.082	0.021	0.000	0.022	0.091	0.210	0.383	0.617

5.3 Sensitivity analysis method for the *q*-generalized logit model

5.3.1 The q-generalized logit traffic assignment with the implicit route-based approach

For solving the route length problem, Nakayama[8] and Nakayama et al.[9] proposed the integrated framework of *q*-generalization with *q*-analysis regarding the MEV distribution. In these studies, the operation of *q*-generalized exponential and *q*-logarithm functions[12] was used to alleviate the homogenous variance of the logit model. The *q*-generalized exponential and *q*-logarithm functions are given respectively as follows:

$$\exp_q(x) = (1 + [1-q]x)^{\frac{1}{1-q}}; \tag{5.32}$$

$$\ln_q(x) = \frac{x^{1-q} - 1}{1-q}. \tag{5.33}$$

When $q = 1$, $\exp_1(x) = \exp(x)$, $\ln_1(x) = \ln(x)$, and the q-generalized logit-traffic assignment collapses to the MNL form. When $q = 0$, it becomes a weibit model.

In the route-choice problem, the proposed model allows the relaxation of the limitations of the logit model. The parameter *q* in the proposed model helps to add a heteroscedastic variance and flexible shape to the discrete-choice models. Nakayama et al. [9] also formulated the *q*-generalized logit-traffic assignment as a fixed-point problem in the following equation:

$$p_k^{rs} = \frac{\exp_{2-q}(-\theta c_k^{rs}(\mathbf{p}))}{\sum_{k \epsilon K^{rs}} \exp_{2-q}(-\theta c_k^{rs}(\mathbf{p}))} \tag{5.34}$$

When link time-flow functions such as the well-known BPR functions are used, the solution to the SUE problem is iterative in the sense of a fixed-point problem. The fixed-point problem with the route-traffic flow is given by:

$$f_k^{rs} = Q_0^{rs} \frac{\exp_{2-q}(-\theta c_k^{rs}(\mathbf{f}))}{\sum_{k \epsilon K^{rs}} \exp_{2-q}(-\theta c_k^{rs}(\mathbf{f}))} \tag{5.35}$$

To avoid route variable in a logit stochastic loading, the STOCH3-efficient route definition and the DFS algorithm were used for solving the CNL model. This section also expresses the q-generalized logit-traffic assignment with the implicit route-based STOCH3-efficient-route approach based on a DFS algorithm.

The fixed-point problem with the link-traffic flow can be derived as:

$$x_{ij} = \sum_{rs \in W} Q_0^{rs} \frac{\sum_{k \in K^{rs}} \delta_{ij,k}^{rs} \exp_{2-q}(-\theta c_k^{rs}(\mathbf{x}))}{\sum_{p \epsilon K^{rs}} \exp_{2-q}\left(-\theta c_p^{rs}(\mathbf{x})\right)} \tag{5.36}$$

The difficulties in the calculation with the implicit route-set including c_k^{rs} and $\delta_{ij,k}^{rs}$ are solved by the DFS algorithm. For example, if the network consists of efficient links in **Figure 5.1** is also used and a network is saved as adjacent links instead of matrices, the DFS algorithm will traverse nodes in the following order: r, 1, 2, 3, 4, s, 5, 4, s. Each time we reach the destination node, we have information concerning a route ($\delta_{ij,k}^{rs}$ is depicted in the mark array each time s is reached) and the travel time on a route (c_k^{rs}) is easily computed by the total of t_{ij} in the mark array. We only use single variables to save c_k^{rs} and $\exp_{2-q}(-\theta c_k^{rs})$, respectively, and reuse them at each step reaching s to save memory. We denote the numerator of **Equation (5.36)** by a link-based variable ($numq_{ij}^{rs}$) and the denominator of **Equation (5.36)** by another single variable ($dnmq^{rs}$). Because **Equation (5.36)** is calculated for each OD pair, we can reuse variables to save memory. When reaching destination node s, the c_k^{rs} and $\exp_{2-q}(-\theta c_k^{rs})$ are calculated and we add c_k^{rs} to $dnmq^{rs}$ and $numq_{ij}^{rs}$ (if the mark variable consists of link ij). When the DFS algorithm stops, the numerator and denominator of **Equation (5.36)** are completely calculated with a link-based variable $dnmq^{rs}$ and a single variable $numq_{ij}^{rs}$. Thus, we do not need to use route variables to calculate **Equation (5.36)**. In a large-scale network, while the number of links is fixed, the

number of routes will increase rapidly when the number of OD pairs increases. Therefore, using link-based variables and calculating link flows will save memory.

In the implicit route-based approach, the DFS-loading procedure with the SRA method for *q*-generalized traffic assignment is given as follows:

Step 0: Preliminaries

(a) Set iteration counter $l := 1$, $\eta \in [1.5,2]$, $\gamma \in [0.01,0.5]$, $\vartheta^{(l-1)} = 0$ and the stop criteria $\sigma > 0$.

(b) Using the free-flow travel time $\{t_{ij}^0\}$, for each OD pair rs, calculate the minimum travel time to all other nodes.

(c) Efficient link definition: For each link ij, set $\Omega_{ij}^{rs} := 1$ if $(1 + h_{ij}^r)(C_{rj}^0 - \mathrm{C}_{ri}^0) \geq t_{ij}^0$, otherwise $\Omega_{ij}^{rs} := 0$.

(d) Find an initial feasible flow pattern based on free-flow-link times

- For each link ij, set $t_{ij} = t_{ij}^0$ and $x_{ij}^{(0)} = 0$, $y_{ij}^{(0)} = 0$, $x_{ij}^{(1)} = 0$;

- For each OD: set all $numq_{ij}^{rs}$, $dnmq^{rs}$ to 0. Running the DFS algorithm from the origin node *r* to the destination node *s* with a recursive technique for all efficient links. Route travel time is calculated each time *s* is reached and added to $dnmq^{rs}$ and $numq_{ij}^{rs}$ (if link ij is included in the mark array). When the DFS algorithm stops, $dnmq^{rs}$ and $numq_{ij}^{rs}$ are calculated and contribute to the total link flow as follows:

$$x_{ij}^{(1)} := x_{ij}^{(1)} + Q_0^{rs} \frac{numq_{ij}^{rs}}{dnmq^{rs}} \tag{5.37}$$

Step 1: Link travel-time update

(a) Set $l := l + 1$;

(b) Update $t_{ij}^{(l)} := t_{ij}\left(x_{ij}^{(l)}\right)$.

Step 2: Direction finding

Based on link travel time $\{t_{ij}^{(l)}\}$, run the DFS algorithm for each OD pair to recalculate the $numq_{ij}^{rs}$ and $dnmq^{rs}$ variables. When the

DFS algorithm stops, the auxiliary link flow is yielded as follows:

$$yx_{ij}^{(l)} := yx_{ij}^{(l)} + Q_0^{rs} \frac{numq_{ij}^{rs}}{dnmq^{rs}} \quad (5.38)$$

Step 3: Link-flow update

(a) Compute the step length $\vartheta^{(l)}$ by using **Equation (5.10)**.

(b) Let $rg_{ij}^{(l)} = yx_{ij}^{(l)} - x_{ij}^{(l)}$.

(c) Set $x_{ij}^{(l+1)} = x_{ij}^{(l)} + \frac{1}{\vartheta^{(l)}} rg_{ij}^{(l)}$.

Step 4: Stopping test

If $max_{ij}\{|rg_{ij}^{(l)}|\} \leq \sigma$, stop. The solution is $\{x_{ij}^{(l)}\}$. Otherwise, go to step 1.

In the proposed method, we do not need to save the route-choice set and the route-based variables to reduce computational storage.

5.3.2 Sensitivity analysis for the q-generalized logit traffic assignment with the implicit route-based approach

Bui et al.[13] assumed the parameter related to free-flow travel time was being changed and the degree of its influence on equilibrium will be considered by the sensitivity analysis method. With no loss of generality, the parameter related to travel demand will be changed and the derivative of the link flow according to the change of this parameter will be also be calculated. With other parameters, the same calculation method could be applied.

The forms of equations of link travel time and travel demand are the same as **Equations (3.1) and (3.9)** in Chapter 3 in which ζ_{ij} represents a small change of the free-flow travel time of link ij and ξ^{rs} depicts a small change in the travel demand of the OD pair rs. In transport planning problem, we want to know how an improvement of the free-flow travel time of a link and how an implement of policies affecting the travel demand will affect the equilibrium, i.e. calculating $\boldsymbol{\nabla_{\zeta}\mathbf{x}}$ and $\boldsymbol{\nabla_{\xi}\mathbf{x}}$**.**

We start from the calculation procedure of $\boldsymbol{\nabla_{\zeta}\mathbf{x}}$ with $\boldsymbol{\xi} \equiv 0$ and fixed travel

demand $\{Q^{rs}\}$. From the q-generalized logit traffic assignment of **Equation (5.36)**, the link traffic flow is represented as the following form:

$$x_{ij} = \sum_{rs \in W} Q^{rs} p_{ij}^{rs}(\mathbf{c}(\mathbf{t}(\mathbf{x}, \boldsymbol{\zeta}))), \tag{5.39}$$

where:

$$p_{ij}^{rs} = \frac{\sum_{k \in K^{rs}} \delta_{ij,k}^{rs} \exp_{2-q}(-\theta c_k^{rs}(\mathbf{t}(\mathbf{x}, \boldsymbol{\zeta})))}{\sum_{p \epsilon K^{rs}} \exp_{2-q}\left(-\theta c_p^{rs}(\mathbf{t}(\mathbf{x}, \boldsymbol{\zeta}))\right)} \tag{5.40}$$

Denoting the right-hand side **Equation (5.40)** by v_{ij} (where v_{ij} is a function of $c_k^{rs}(t_{ij})$ and t_{ij} is a function of (x_{ij}, ζ_{ij})), with the same approach shown in Chapter 3, $\boldsymbol{\nabla_{\zeta}\mathbf{x}}$ is calculated by

$$\boldsymbol{\nabla_{\zeta}\mathbf{x}} = (\mathbf{I} - \boldsymbol{\nabla_{\mathbf{t}}\mathbf{v}\nabla_{\mathbf{x}}\mathbf{t}})^{-1}\left(\boldsymbol{\nabla_{\mathbf{t}}\mathbf{v}\nabla_{\zeta}\mathbf{t}}\right), \tag{5.41}$$

where $\boldsymbol{\nabla_{\mathbf{x}}\mathbf{t}}$ and $\boldsymbol{\nabla_{\zeta}\mathbf{t}}$ are easily computed from the explicit BPR function (see Chapter 3), the difficulty in calculating **Equation (5.41)** lies in calculating $\boldsymbol{\nabla_{\mathbf{t}}\mathbf{v}}$. The elements of $\boldsymbol{\nabla_{\mathbf{t}}\mathbf{v}}$ have the same form as **Equations (5.20) and (5.21)** with the difficult to calculate $\frac{\partial v_{ij}}{\partial t_{gh}}$ through $\frac{\partial p_{ij}^{rs}}{\partial t_{gh}}$.

By the same approach, the calculation of $\boldsymbol{\nabla_{\xi}\mathbf{x}}$ with $\boldsymbol{\zeta} \equiv 0$ is resolved by:

$$\boldsymbol{\nabla_{\xi}\mathbf{x}} = -(\mathbf{I} - \boldsymbol{\nabla_{\mathbf{t}}\mathbf{v}\nabla_{\mathbf{x}}\mathbf{t}})^{-1}\boldsymbol{\nabla_{\mathbf{Q}}\mathbf{v}}. \tag{5.42}$$

where the elements of $\boldsymbol{\nabla_{\mathbf{Q}}\mathbf{v}}$ is given by the identical form as **Equation (3.33)** (in Chapter 3). However, the difficulties in calculating $\boldsymbol{\nabla_{\mathbf{t}}\mathbf{v}}$ and $\boldsymbol{\nabla_{\mathbf{Q}}\mathbf{v}}$ for the q-generalized logit model is different from the MNL model. The calculations of $\boldsymbol{\nabla_{\mathbf{t}}\mathbf{v}}$ and $\boldsymbol{\nabla_{\mathbf{Q}}\mathbf{v}}$ are decomposed as follows.

Firstly, the elements of $\boldsymbol{\nabla_{\mathbf{t}}\mathbf{v}}$ is calculated by resolving $\frac{\partial p_{ij}^{rs}}{\partial t_{gh}}$. Denoting p_{ij}^{rs} by the following equation:

$$p_{ij}^{rs} = \frac{\sum_{k \in K^{rs}} \delta_{ij,k}^{rs} \exp_{2-q}(-\theta c_k^{rs})}{\sum_{p \epsilon K^{rs}} \exp_{2-q}(-\theta c_p^{rs})} = \frac{numq_{ij}^{rs}}{dnmq^{rs}}, \tag{5.43}$$

Using the rule of derivative of a quotient (see **Appendix B**), $\frac{\partial p_{ij}^{rs}}{\partial t_{gh}}$ is give as:

$$\frac{\partial p_{ij}^{rs}}{\partial t_{gh}} = \frac{\frac{\partial numq_{ij}^{rs}}{\partial t_{gh}}}{dnmq^{rs}} - \frac{numq_{ij}^{rs} \frac{\partial dnmq^{rs}}{\partial t_{gh}}}{(dnmq^{rs})^2}, \tag{5.44}$$

in which,

$$\frac{\partial numq_{ij}^{rs}}{\partial t_{gh}} = \frac{\partial \left(\sum_{k \in K^{rs}} \delta_{ij,k}^{rs} \exp_{2-q}(-\theta c_k^{rs}) \right)}{\partial t_{gh}} = \frac{\partial \left(\sum_{k \in K^{rs}} \delta_{ij,k}^{rs} \left(1 + [q-1](-\theta c_k^{rs})\right)^{\frac{1}{q-1}} \right)}{\partial t_{gh}}$$

$$= \frac{\sum_{k \in K^{rs}} \partial \left[\delta_{ij,k}^{rs} \left(1 + [q-1](-\theta c_k^{rs})\right)^{\frac{1}{q-1}} \right]}{\partial t_{gh}}$$

$$= \frac{1}{q-1} \left(\sum_{k \in K^{rs}} \left[\left(1 + [q-1](-\theta c_k^{rs})\right)^{\left(\frac{1}{q-1}-1\right)} \frac{\partial \left(1 + [q-1]\left(-\theta c_k^{rs} \delta_{ij,k}^{rs}\right)\right)}{\partial t_{gh}} \right] \right)$$

$$= \frac{1}{q-1} \left(\sum_{k \in K^{rs}} \left[\left(1 + [q-1](-\theta c_k^{rs})\right)^{\left(\frac{2-q}{q-1}\right)} [q-1]\left(-\theta \delta_{ij,k}^{rs}\right) \frac{\partial c_k^{rs}}{\partial t_{gh}} \right] \right)$$

$$= \frac{1}{q-1} \left(\sum_{k \in K^{rs}} \left[\exp_{2-q}(-\theta c_k^{rs}) \right]^{2-q} [q-1]\left(-\theta \delta_{ij,k}^{rs} \delta_{gh,k}^{rs}\right) \right)$$

$$= -\theta \left(\sum_{k \in K^{rs}} \left[\exp_{2-q}(-\theta c_k^{rs})\, \delta_{ij,k}^{rs} \delta_{gh,k}^{rs} \right]^{2-q} \right),$$

$$\frac{\partial dnmq^{rs}}{\partial t_{gh}} = \frac{\partial\left(\sum_{p\in K^{rs}} \exp_{2-q}\left(-\theta c_p^{rs}\right)\right)}{\partial t_{gh}} = \frac{\partial\left(\sum_{p\in K^{rs}} \left(1+[q-1]\left(-\theta c_p^{rs}\right)\right)^{\frac{1}{q-1}}\right)}{\partial t_{gh}}$$

$$= \frac{1}{q-1}\left(\sum_{p\in K^{rs}} \left[\exp_{2-q}\left(-\theta c_p^{rs}\right)\right]^{2-q}[q-1]\left(\delta_{gh,p}^{rs}\right)\right)$$

$$= -\theta\left(\sum_{p\in K^{rs}} \left[\exp_{2-q}\left(-\theta c_p^{rs}\right)\delta_{ij,p}^{rs}\right]^{2-q}\right).$$

Substituting $\frac{\partial numq_{ij}^{rs}}{\partial t_{gh}}$ and $\frac{\partial dnmq^{rs}}{\partial t_{gh}}$ to **Equation (5.44)**, we have:

$$\frac{\partial p_{ij}^{rs}}{\partial t_{gh}} = \frac{-\theta\left(\sum_{k\in K^{rs}} \left[\exp_{2-q}\left(-\theta c_k^{rs}\right)\delta_{ij,k}^{rs}\delta_{gh,k}^{rs}\right]^{2-q}\right)}{dnmq^{rs}} - \frac{-\theta\left(\sum_{p\in K^{rs}} \left[\exp_{2-q}\left(-\theta c_p^{rs}\right)\delta_{ij,p}^{rs}\right]^{2-q}\right) numq_{ij}^{rs}}{(dnmq^{rs})^2} \quad (5.45)$$

Using the following definitions:

$$b_{1,ij,gh}^{rs} = \sum_{k\in K^{rs}} \left[\exp_{2-q}\left(-\theta c_k^{rs}\right)\delta_{ij,k}^{rs}\delta_{gh,k}^{rs}\right]^{2-q}, \quad (5.46)$$

$$b_{2,gh}^{rs} = \sum_{p\in K^{rs}} \left[\exp_{2-q}\left(-\theta c_p^{rs}\right)\delta_{gh,p}^{rs}\right]^{2-q}, \quad (5.47)$$

Equations (5.45) becomes:

$$\frac{\partial p_{ij}^{rs}}{\partial t_{gh}} = -\theta\left(\frac{b_{1,ij,gh}^{rs}}{dnmq^{rs}} - \frac{numq_{ij}^{rs} b_{2,gh}^{rs}}{(dnmq^{rs})^2}\right). \quad (5.48)$$

Secondly, the elements of $\boldsymbol{\nabla_Q \mathbf{v}}$ is computed by

$$\frac{\partial v_{ij}}{\partial Q^{rs}} = \frac{\partial \sum_{rs \in W} Q^{rs} p_{ij}^{rs}}{\partial Q^{rs}} = p_{ij}^{rs}. \quad (5.49)$$

With the same technique used in Section 5.3.1, the DFS algorithm is utilized to calculate $\frac{\partial p_{ij}^{rs}}{\partial t_{gh}}$ and $\frac{\partial v_{ij}}{\partial Q^{rs}}$. Besides calculating two variables $numq_{ij}^{rs}$ and $dnmq^{rs}$, we have to calculate two more variables $b_{1,ij,gh}^{rs}$ and $b_{2,gh}^{rs}$. We will calculate for each OD pair. Thus, we can reuse variables to save memory. For example, we only need a $link\ x\ link$ matrix to store $b_{1,ij,gh}^{rs}$. At each time reaching destination node s, we have the information of c_k^{rs}, $\delta_{ij,k}^{rs}$, and $\delta_{gh,k}^{rs}$ variables. Because the DFS algorithm can be used to calculate route-travel time and depict the link-route relationship, two variables $b_{1,ij,gh}^{rs}$ and $b_{2,gh}^{rs}$ are easily computed and the remaining difficulty lies in the calculation technique when using information brought from the DFS algorithm. The following calculation process is performed for each OD pair in which the DFS algorithm is used. The process of calculating $\mathbf{\nabla_t g}$ and $\mathbf{\nabla_Q g}$ in the initial SUE state is presented by following three steps.

Step 1: Performing the DFS-loading procedure using the STOCH3-efficient-route definition with the SRA method for q-generalized traffic assignment to achieve link flow and link travel time at the equilibrium state.

Step 2: Calculate $\mathbf{\nabla_t v}$ and $\mathbf{\nabla_Q v}$

- For each OD pair: set all $numq_{ij}^{rs}$, $b_{1,ij,gh}^{rs}$, $b_{2,gh}^{rs}$ and $dnmq^{rs}$ to 0. Run the DFS algorithm from r to s with a recursive technique for all efficient links and use the one-dimensional array to mark the route. When reaching destination node s, we calculate the route travel time without saving and add directly to $dnmq^{rs}$ and $numq_{ij}^{rs}$ (if link ij is marked). At the same time, with the information about the links that belong to a route k ($\delta_{ij,k}^{rs}$ and $\delta_{gh,k}^{rs}$ variables are represented through the mark array), we compute the $b_{1,ij,gh}^{rs}$ and $b_{2,gh}^{rs}$ variables. With a recursive technique, we can reach the destination node s by $|K^{rs}|$

times (the number of efficient routes) and the $numq_{ij}^{rs}$, $dnmq^{rs}$, $b_{1,ij,gh}^{rs}$ and $b_{2,gh}^{rs}$ variables are calculated. Then, $\frac{\partial p_{ij}^{rs}}{\partial t_{gh}}$ and $\frac{\partial v_{ij}}{\partial Q^{rs}}$ are computed and contributed to the calculation of $\boldsymbol{\nabla_t \mathbf{v}}$ and $\boldsymbol{\nabla_Q \mathbf{v}}$.

- When all OD pairs are considered, $\boldsymbol{\nabla_t \mathbf{v}}$ and $\boldsymbol{\nabla_Q \mathbf{v}}$ are totally calculated.

Step 3: Calculate $\boldsymbol{\nabla_\zeta \mathbf{x}}$ and $\boldsymbol{\nabla_\xi \mathbf{x}}$ according to **Equations (5.41), (5.42)**.

After the derivatives of link traffic flow with respect to perturbations, $\boldsymbol{\zeta}$, and $\boldsymbol{\xi}$, are calculated, we can compute the approximate link flow with the changes of $\boldsymbol{\zeta}$ and $\boldsymbol{\xi}$, according to first-order Taylor expansion.

5.3.3 Applications

In this section, a small example and application to Kanazawa City are adopted to demonstrate our method's accuracy and effectiveness.

Small Example

We will use the network with 3 nodes and 3 links, which is equivalent to that used by Nakayama et al.[9]. The link free-flow travel time and capacity are written in the form [free-flow travel time (min), capacity (pcu)]. The network has two OD pairs from node 1 to node 3 (OD1) and node 2 to node 3 (OD2), in which the demands are both 150 (pcu). The parameters of the BPR-type performance function are $\alpha = 1.0$ and $\beta = 2.0$. Set $\theta = 2.0$ and the parameter q is the same for two OD pairs and equals 0.5.

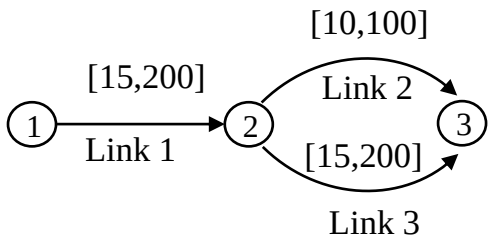


Figure 5.5 Small example network

In the implicit route-based approach, we assume that all links of the above networks are efficient. After using the DFS algorithm with the MSA method, the link flow and probability of choosing a link at the equilibrium state are presented in **Table 5.6.**

Table 5.6 Link-flow results of the example network

Link	Link flow (pcu)			Probability of choosing the link	
	OD1	OD2	Total	OD1	OD2
1	150.00	0.00	150.00	1.0	0.0
2	70.05	65.89	135.94	0.467	0.439
3	79.95	84.11	164.06	0.533	0.561

Note that the route flow by logit-based stochastic assignment satisfies the following equation representing the Markov property:

$$p_k^{rs} = \prod_{ij \in A} p(j/i)^{\delta_{ij,k}^{rs}}, \tag{5.50}$$

where $p(j/i)$ denotes the forward conditional probability of choosing node j from node i. With N_i^{out} being the set of nodes linked from node i, we have

$$p(j/i) = \frac{p_{ij}^{rs}}{\sum_{n \in N_i^{out}} p_{in}^{rs}}. \tag{5.51}$$

Thus, we can compute the probability of choosing route and route flow in **Table 5.7**.

Table 5.7 Route-flow results of an example network

OD	Route	Links included	Probability of choosing the route	Route flow (pcu)
1	1	1,2	0.467	70.05
	2	1,3	0.533	79.95
2	1	2	0.439	65.89
	2	3	0.561	84.11

The above results are equivalent to those using a route-based approach with the MSA method for the SUE *q*-generalized traffic assignment (*q*RBSUE). When using this approach, the number of iterations needed to achieve equilibrium depends on the initial value of the route flow chosen. If we choose this value to be 0, the number of iterations to achieve equilibrium is 104,419. In the implicit route-based approach with an MSA method, we only use free-flow travel time at the initial step and need 45 iterations to achieve equilibrium.

Since the travel-time differences between the two routes of each OD pair are equal, the probabilities of choosing route 1 of OD1 and route 1 of OD2 are the same using the MNL model assuming homogeneity of variance. However, they differ in the *q*-generalized logit model because it considers the influence of route length.

After using the SA method, we have the following results of $\boldsymbol{\nabla_{\zeta}\mathbf{x}}$ and $\boldsymbol{\nabla_{\xi}\mathbf{x}}$

$$\boldsymbol{\nabla_{\zeta}\mathbf{x}} = \begin{pmatrix} 0 & 0 & 0 \\ 0.058 & -4.293 & 2.793 \\ -0.058 & 4.293 & -2.793 \end{pmatrix}, \qquad \boldsymbol{\nabla_{\xi}\mathbf{x}} = \begin{pmatrix} 1 & 0 \\ 0.389 & 0.374 \\ 0.611 & 0.626 \end{pmatrix}.$$

As can be seen, the flow on link 1 does not change when we change the parameters $\boldsymbol{\zeta}$ and ξ^{23}. The equilibrium solution is the most sensitive with the change of ζ_2 and ξ^{13}. In the traffic networks, the paradox can occur in which network improvements (such as reducing free-flow travel time) can result in an overall increase in total system travel times, $\sum_{ij=1}^{|A|}(x_{ij}t_{ij})$. To examine whether this problem occurs in this small network, suppose that free-flow travel time on link 2 is reduced by 1 to 5 percent and travel demand of OD pair 13 is reduced by 1 to 5 percent, we need to calculate 10 *q*RBSUE traffic assignment times which causes computational time-consuming. Meanwhile, by using the sensitivity analysis method, we can easily calculate the approximation results with the saving time of the calculation. To assess the accuracy of the proposed sensitivity analysis method for *q*-generalized traffic assignment (*q*SA), the $RMSE$ and $\%RMS$ indicators are utilized.

The comparison results are expressed in **Table 5.8**. The link-flow results may be the same between the two methods with small values of the indicators $RMSE$ and $\%RMS$. The index of total network travel time when changing parameters is

almost the same. And the paradox does not happen when the travel demand of OD pair 13 and free-flow travel time on link 2 decreases. In this case, reducing the demand to $5\%Q^{13}$ and reducing the travel time to $5\%t_2^0$ is the best option in both calculational methods. Also, with the proposed qSA method, total saving time is up to 94.9% and we also save the memory because we do not need an explicit route-set. It is beyond doubt that the proposed method can maintain computational accuracy and reduce computational cost.

Table 5.8 Comparisons between two methods in the example network

Method	Results	ζ_2					ξ^{13}				
		$-1\%t_2^0$	$-2\%t_2^0$	$-3\%t_2^0$	$-4\%t_2^0$	$-5\%t_2^0$	$-1\%Q^{13}$	$-2\%Q^{13}$	$-3\%Q^{13}$	$-4\%Q^{13}$	$-5\%Q^{13}$
qRBSUE	Flow of link (pcu) Link 1	150.00	150.00	150.00	150.00	150.00	148.50	147.00	145.50	144.00	142.50
	Link 2	136.37	136.81	137.25	137.69	138.14	135.36	134.77	134.19	133.61	133.02
	Link 3	163.63	163.19	162.75	162.30	161.85	163.14	162.23	161.31	160.39	159.47
	Required iteration	104521	104624	104726	104828	104930	104456	104496	104538	104581	104627
	Calculation time (sec)	0.016	0.031	0.031	0.047	0.047	0.078	0.063	0.094	0.094	0.109
	Total calculation time (sec)	0.609									
	$\sum_{ij=1}^{\lvert A\rvert}(x_{ij}t_{ij})$	11,473.6	11,443.1	11,412.4	11,381.4	11,350.2	11,364.3	11,226.2	11,089.3	10,953.9	10,819.7
qSA	Flow of link (pcu) Link 1	150.00	150.00	150.00	150.00	150.00	148.50	147.00	145.50	144.00	142.50
	Link 2	136.37	136.80	137.23	137.66	138.09	135.36	134.77	134.19	133.60	133.02
	Link 3	163.63	163.20	162.77	162.34	161.91	163.14	162.23	161.31	160.40	159.48
	Required calculation time for the first solution (sec)	0.016									
	Total calculation time (sec)	0.031									
	$\sum_{ij=1}^{\lvert A\rvert}(\tilde{x}_{ij}t_{ij})$	11,473.7	11,443.2	11,412.2	11,380.9	11,349.3	11,364.5	11,226.3	11,089.5	10,954.0	10,819.8
$RMSE$		0.044	0.003	0.008	0.017	0.030	0.047	0.002	0.002	0.002	0.003
$\%RMS(\%)$		0.029	0.002	0.005	0.011	0.020	0.031	0.001	0.001	0.001	0.002

Application to Kanazawa road network

The performance of the proposed method will be shown in the application to the Kanazawa road network. The OD demand data from a previous personal-trip survey is used with 383 OD pairs of the morning peak from 6:00 to 7:00 AM. The BPR function of the travel time of the car uses $\alpha = 1.0$, $\beta = 2.0$, and the assumed parameter is $\theta = 1$. The elongation ratio h_{ij}^r is set to 1.5 for every link of the network. The parameter q is the same for all OD pairs and equals 0.5. The programs were coded in Fortran.90 programming language and ran on a personal computer.

The differences between the two methods are shown in **Table 5.9** and **Figure 5.6**. Almost identical results are obtained for traffic flow when parameters $\boldsymbol{\zeta}$ and $\boldsymbol{\xi}$ equal to 0. When we change the parameter $\boldsymbol{\zeta}$ for all links in the range $[1\%, 5\%]$ of the free-flow travel time $\mathbf{t}^0$ and the parameter $\boldsymbol{\xi}$ for all OD pairs in the range $[1\%, 5\%]$ of the fixed travel demand $\mathbf{Q}^0$ the proposed method performs well with small values of both the $RMSE$ and $\%RMS$ indicators. Like the above, to find out whether the transport network exists paradoxically, we need to calculate the total system travel time of the network for all changes by recalculating the SUE state or using the proposed sensitivity analysis method. The comparison results of the two methods, which are shown in **Figure 5.7**, are almost identical. Calculated results depict that a decrease in free-flow travel time and a reduction in travel demand will lead to a reduction in the total system travel time. Interestingly, using the *q*SA method will save computational time because we do not need to recompute the traffic assignment equilibrium problem. Based on the initial traffic-assignment result, we only need to calculate the $\boldsymbol{\nabla_{\zeta}\mathbf{x}}$ and $\boldsymbol{\nabla_{\xi}\mathbf{x}}$ that are used to compute approximate link flows influenced by any changes of parameters. As the number of scenarios increases, the effectiveness of the *q*SA method will exponentially increase. In this case study, we only considered 10 scenarios and the total cumulative calculation time for 11 cases was the sum of the calculation time of 10 scenarios and the case of parameters $\boldsymbol{\zeta}$ and $\boldsymbol{\xi}$ equals 0. The proposed method reduces the computational time by up to 97%. This proves the time-efficient calculation of the proposed method.

Table 5.9 Accuracy of the proposed sensitivity analysis method for the *q*-generalized logit model

Case	1	2	3	4	5	6	7	8	9	10	11
Parameters	$\boldsymbol{\zeta} = \mathbf{0}$;			$\boldsymbol{\zeta}$					$\boldsymbol{\xi}$		
	$\boldsymbol{\xi} = \mathbf{0}$	$-1\%\mathbf{t}^0$	$-2\%\mathbf{t}^0$	$-3\%\mathbf{t}^0$	$-4\%\mathbf{t}^0$	$-5\%\mathbf{t}^0$	$-1\%\mathbf{Q}^0$	$-2\%\mathbf{Q}^0$	$-3\%\mathbf{Q}^0$	$-4\%\mathbf{Q}^0$	$-5\%\mathbf{Q}^0$
$RMSE$	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.006	0.010	0.015
$\%RMS$ (%)	0.002	0.002	0.002	0.002	0.002	0.003	0.002	0.003	0.005	0.009	0.014

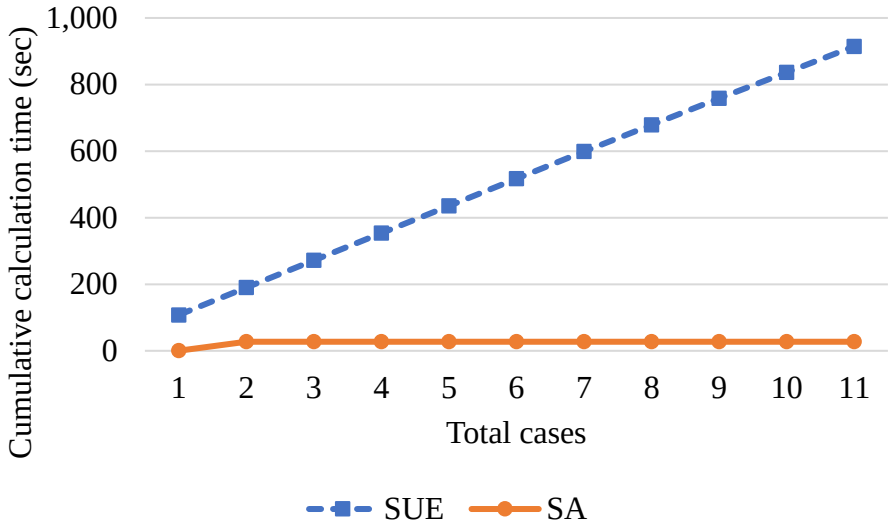


Figure 5.6 Total cumulative calculation time of two methods

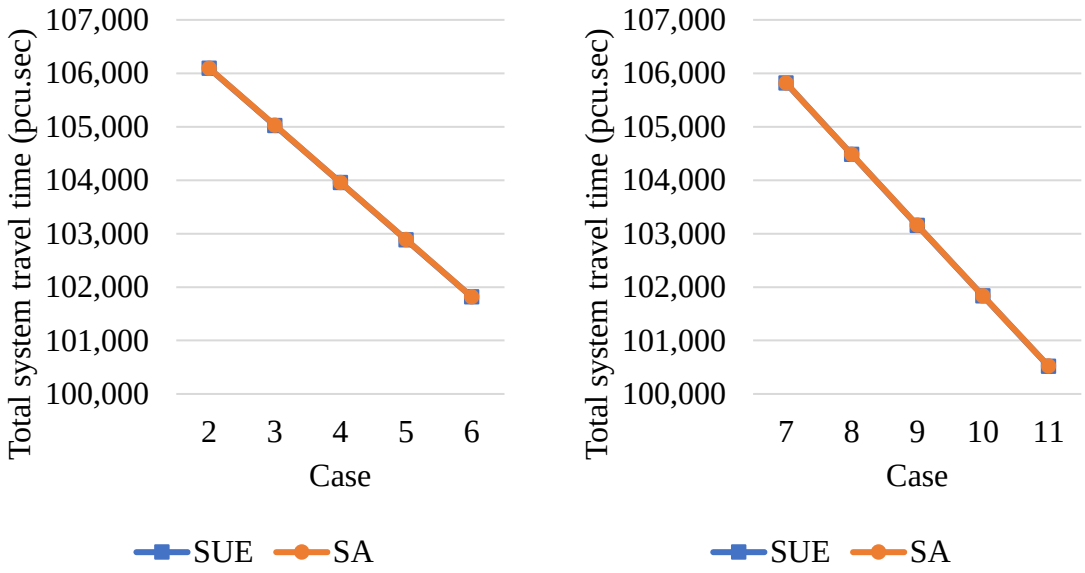


Figure 5.7 Total system travel time calculated by two methods

5.4 Semi-dynamic stochastic user equilibrium traffic assignment based on the extensions of the sensitivity analysis method for cross-nested logit model and *q*-generalized logit model

5.4.1 Modeling and algorithm

The semi-dynamic SUE traffic assignment model based on the logit model was proposed in Chapter 4. The CNL model and the *q*-generalized logit model are the extensions of the logit model, so they can be applied for the semi-DTA model as following double-loop fixed-point problem in each period:

$$\mathbf{f}_\tau = \mathbf{Q}_\tau \mathbf{p}_\tau(\Delta^T \mathbf{t}_\tau(\Delta \mathbf{f}_\tau - \mathbf{s}_\tau[\mathbf{f}_\tau, \mathbf{t}_\tau])), \quad (5.52)$$

All of the variables used in this section are the same as those presented in Chapter 4. Basically, we can use the algorithm proposed in Chapter 4 (on pages 107,108) for solving the double-loop fixed-point problem of the semi-DTA of CNL and q-generalized logit model. The difference between the considered logit-based models is only in the formulation and calculation of the probability of choosing route k between each OD pair rs in period τ. The details of these algorithms are shown in **Appendix G**.

However, because solving the above equation requires the computational time and memory, the method of sensitivity analysis using only link-based and node-based variables is proposed to achieve approximated results of the semi-DTA model. The two algorithms that can be applied are the same as those in Chapter 4. As in Chapter 4 mentioned the second algorithm gives better approximation results. The focus here will be on the second algorithm.

In this semi-DTA approach, we calculate the equilibrium state for each period τ. In each period, we can calculate the static SUE using the CNL model and the *q*-generalized logit model with the DFS algorithm and SRA method proposed in Sections 5.2 and 5.3. From these results and solving a fixed-point problem with eliminated link flow in each period τ, we can compute the semi-DTA model (see algorithm 2 in Chapter 4). As stated in Chapter 4, there are two difficulties including the calculation of $\boldsymbol{\nabla}_\mathbf{s}\mathbf{x}_0$ at the static SUE and $x_{\tau,ij}^{rs}$, $x_{\tau,ij\to gh}^{rs}$ variables.

Firstly, let we reuse the formulation of $\boldsymbol{\nabla}_\mathbf{s}\mathbf{x}_0$ shown in **Equation (4.45)**

$$\boldsymbol{\nabla}_\mathbf{s}\mathbf{x}_0 = (\mathbf{I} - \boldsymbol{\nabla}_\mathbf{t}\mathbf{g}_0\boldsymbol{\nabla}_\mathbf{x}\mathbf{t}_0)^{-1}(\boldsymbol{\nabla}_\mathbf{t}\mathbf{g}_0\boldsymbol{\nabla}_\mathbf{s}\mathbf{t}_0), \quad (5.53)$$

where all the above variables are the same as those used in Chapter 4. $\mathbf{\nabla_x t}_0$ and $\mathbf{\nabla_s t}_0$ are calculated from the explicit BRP function as shown in Chapter 4 from **Equations (4.46)- (4.48)**. If **Equation (5.53)** is applied to the CNL model, $\mathbf{\nabla_t g}_0$ is calculated by using the same procedures as the calculation of $\mathbf{\nabla_t h}$ shown in Section 5.2. Also, the calculation of $\mathbf{\nabla_t v}$ for the q-generalized logit model proposed in Section 5.3 is equivalent to the calculation of $\mathbf{\nabla_t g}_0$ if the q-generalized logit model is considered. After $\mathbf{\nabla_t g}_0$ is calculated, we can calculate $\mathbf{\nabla_s x}_0$.

Secondly, sections 5.2 and 5.3 also imply how to calculate $x_{\tau,ij}^{rs}$, we have

$$x_{\tau,ij}^{rs} = Q_{\tau}^{rs} p_{\tau,ij}^{rs} \tag{5.54}$$

In the semi-DTA CNL model, $p_{\tau,ij}^{rs}$ is computed as

$$p_{\tau,ij}^{rs} = \frac{\sum_{k \in K_{ij}^{rs}} \sum_{mn \in A} \left[u_{\tau,1,mn,k}^{rs} \left(\left[u_{\tau,2,mn}^{rs} \right]^{\mu-1} \right) \right]}{\sum_{ij \in A} \left[u_{\tau,2,ij}^{rs} \right]^{\mu}} = \frac{numc_{\tau,ij}^{rs}}{dnmc_{\tau}^{rs}} \tag{5.55}$$

where $u_{\tau,1,mn,k}^{rs}$, $u_{\tau,2,mn}^{rs}$, $numc_{\tau,ij}^{rs}$ and $dnmc_{\tau}^{rs}$ denote $u_{1,mn,k}^{rs}$, $u_{2,mn}^{rs}$, $numc_{ij}^{rs}$ and $dnmc^{rs}$ variables used in Section 5.2 for OD pair each period τ because $p_{\tau,ij}^{rs}$ is computed for each OD pair in each period τ. The calculation of $numc_{\tau,ij}^{rs}$ and $dnmc_{\tau}^{rs}$ are the same as the calculation of $numc_{ij}^{rs}$ and $dnmc^{rs}$ by running the DFS algorithm twice for each OD pair rs in each period τ (see Section 5.2).

Also, p_{ij}^{rs} could be computed in the semi-DTA q-generalized logit model

$$p_{\tau,ij}^{rs} = \frac{\sum_{k \in K^{rs}} \delta_{ij,k}^{rs} \exp_{2-q}\left(-\theta c_{\tau,k}^{rs}\right)}{\sum_{p \epsilon K^{rs}} \exp_{2-q}\left(-\theta c_{\tau,p}^{rs}\right)} = \frac{numq_{\tau,ij}^{rs}}{dnmq_{\tau}^{rs}} \tag{5.56}$$

where $numq_{\tau,ij}^{rs}$ and $dnmq_{\tau}^{rs}$ variables are computed for each OD pair rs in each period τ. Section 5.3 depicted how to calculate $numq_{ij}^{rs}$ and $dnmq^{rs}$ for the static q-generalized logit model, the same calculation is applied to the calculating $numq_{\tau,ij}^{rs}$ and $dnmq_{\tau}^{rs}$.

The final difficult is how to calculate $x_{\tau,ij \to gh}^{rs}$. From chapter 4,

$$x_{\tau,ij\to gh}^{rs} = \sum_{k\in K_{ij\to gh}^{rs}} f_{\tau,k}^{rs} = \sum_{k\in K_{ij\to gh}^{rs}} Q_{\tau,}^{rs} p_{\tau,k}^{rs} \tag{5.57}$$

where $K_{ij\to gh}^{rs}$ denotes the set of efficient routes of OD pair rs in which both links ij and gh are used and link ij is used before link gh.

In the semi-DTA CNL model, applying **Equation (2.27)** and the denotations in **Equation (5.55)** gives

$$\begin{aligned} x_{\tau,ij\to gh}^{rs} &= \sum_{k\in K_{ij\to gh}^{rs}} Q_{\tau,k}^{rs} \frac{\sum_{mn\in A}\left[u_{\tau,1,mn,k}^{rs}\left(\left[u_{\tau,2,mn}^{rs}\right]^{\mu-1}\right)\right]}{\sum_{ij\in A}\left[u_{\tau,2,ij}^{rs}\right]^{\mu}} \\ &= Q_{\tau,k}^{rs} \frac{\sum_{k\in K_{ij\to gh}^{rs}}\sum_{mn\in A}\left[u_{\tau,1,mn,k}^{rs}\left(\left[u_{\tau,2,mn}^{rs}\right]^{\mu-1}\right)\right]}{\sum_{ij\in A}\left[u_{\tau,2,ij}^{rs}\right]^{\mu}} \end{aligned} \tag{5.58}$$

where the denominator of the above equation could be computed through $dnmc_{\tau}^{rs}$ variable. Denoting the nominator of **Equation (5.58)** as $dc_{\tau,ij,gh}^{rs}$. Applying the same procedure shown in the calculation of **Equation (5.30)**, $dc_{\tau,ij,gh}^{rs}$ could be computed after having the results $u_{\tau,2,mn}^{rs}$ of all links mn as follows: By using the DFS algorithm, at each time of reaching destination node s, we know the information of a route in the mark array. $u_{\tau,1,mn,k}^{rs}\left(\left[u_{\tau,2,mn}^{rs}\right]^{\mu-1}\right)$ is computed for all links mn in the mark array, summed up and added to $dc_{\tau,ij,gh}^{rs}$ of the links ij and gh if link ij is saved before link gh in mark array. After the DFS algorithm stops, $dc_{\tau,ij,gh}^{rs}$ is completely computed for all links ij and gh.

In the semi-DTA q-generalized logit model, using **Equation (5.34)** and the denotations in **Equation (5.56)** gives

$$x_{\tau,ij\to gh}^{rs} = \sum_{k\in K_{ij\to gh}^{rs}} Q_{\tau,k}^{rs} \frac{\exp_{2-q}\left(-\theta c_{\tau,k}^{rs}\right)}{\sum_{p\epsilon K^{rs}} \exp_{2-q}\left(-\theta c_{\tau,p}^{rs}\right)}$$
$$= Q_{\tau,k}^{rs} \frac{\sum_{k\in K_{ij\to gh}^{rs}} \exp_{2-q}\left(-\theta c_{\tau,k}^{rs}\right)}{\sum_{p\epsilon K^{rs}} \exp_{2-q}\left(-\theta c_{\tau,p}^{rs}\right)} \quad (5.59)$$

where the denominator of **Equation (5.59)** could be computed through $dnmq_{\tau}^{rs}$ variable. Denoting the nominator of **Equation (5.59)** as $dq_{\tau,ij,gh}^{rs}$. This variable is calculated as follows: By using the information at each time reaching destination node s from the DFS algorithm, $\exp_{2-q}\left(-\theta c_{\tau,k}^{rs}\right)$ is computed and added to $dq_{\tau,ij,gh}^{rs}$ if link ij is used before link gh in the mark array. $dq_{\tau,ij,gh}^{rs}$ is completely computed when the DFS algorithm stops for each OD pairs rs.

Finally, the algorithm of semi-dynamic SUE with flow propagation for the CNL model and the *q*-generalized logit model based on the sensitivity analysis includes the following four steps for each period τ.

Step 1: Computing static SUE by using the "STOCH3-efficient route" definition, DFS algorithm, and SRA method in period $\tau = 1$.

Step 2: Set current iteration counter $l := 1$, $\eta \in [1.5,2]$, $\gamma \in [0.01,0.5]$, $\vartheta^{(l-1)} = 0$ and convergence test value σ. Computing $\mathbf{\nabla_s x}_0$ and initial solution of eliminated flow $\mathbf{s}_{\tau}{}^{(l)}$ based on the results at the static SUE state.

Step 3: Solving the fixed-point problem of eliminated link flow:

(a) Computing the reference link flow $\{x_{\tau,ij}{}^{(l)}\}$, adjusted link flow $\{z_{\tau,ij}{}^{(l)}\}$ and link travel time $\{t_{\tau,ij}{}^{(l)}\}$ based on the current solution of eliminated flow $\{s_{\tau,ij}^{(l)}\}$,

(b) Based on $\{t_{\tau,ij}{}^{(l)}\}$, using the DFS algorithm to calculate $\{x_{\tau,ij\to gh}^{(l)}\}$ the auxiliary eliminated flow $sy_{\tau,ij}{}^{(l)}$ is given by:

$$sy_{\tau,ij}{}^{(l)} = \sum_{rs \in W_\tau} \sum_{gh \in A} \frac{x^{rs}_{\tau,gh \to ij}{}^{(l)}}{L} t_{gh}{}^{(l)}$$

(c) Compute the step length $\vartheta^{(l)}$ as follows:

$$\vartheta^{(l)} = \begin{cases} \vartheta^{(l-1)} + \eta, \text{if } \left\| \mathbf{s}_\tau{}^{(l)} - \mathbf{sy}_\tau{}^{(l)} \right\| \geq \left\| \mathbf{s}_\tau{}^{(l-1)} - \mathbf{sy}_\tau{}^{(l-1)} \right\| \\ \vartheta^{(l-1)} + \gamma, \text{if } \left\| \mathbf{s}_\tau{}^{(l)} - \mathbf{sy}_\tau{}^{(l)} \right\| < \left\| \mathbf{s}^{(l-1)} - \mathbf{sy}_\tau{}^{(l-1)} \right\| \end{cases}$$

(d) Let $rg_{\tau,ij}{}^{(l)} = sy_{\tau,ij}{}^{(l)} - s_{\tau,ij}{}^{(l)}$.

(e) Set $s_{\tau,ij}{}^{(l+1)} = s_{\tau,ij}{}^{(l)} + \frac{1}{\vartheta^{(l)}} rg_{\tau,ij}{}^{(l)}$.

(f) Stopping test: If $max_{ij}\{|rg_{\tau,ij}{}^{(l)}|\} \leq \varepsilon$, stop. $\{s_{\tau,ij}{}^{(l)}\}$,$\{x_{\tau,ij}{}^{(l)}\}$ $\{z_{\tau,ij}{}^{(l)}\}$ and $\{t_{\tau,ij}{}^{(l)}\}$ are the approximate solutions. Otherwise, come back to substep (a).

Step 4: Calculate the residual link flow for flow propagation: Using an $|N| \times |N|$ matrix, **B**, to store the value of $\{y^{rs}_{\tau,ij}\}$. Based on the equilibrium results of $\{t_{\tau,ij}{}^{(l)}\}$ at step 3, for each OD pair: Running DFS algorithm to calculate $x^{rs}_{\tau,ij}$ and calculate the residual link flow:

$$y^{rs}_{\tau,ij} = \frac{x^{rs}_{\tau,ij}}{L} t_{\tau,ij}$$

$\{y^{rs}_{\tau,ij}\}$ is added to the row j and column s of matrix **B**. After all OD pairs are regarded, **B** is completely computed. The matrix **B** is propagated to the period $\tau + 1$ and the travel demand matrix for the period $\tau + 1$ is also created by using the matrix **B** and the new travel demand in this period.

The same procedure is conducted for the next period.

5.4.2 Comparisons

A small application

We will perform some semi-DTA models including the MNL, the CNL, and the *q*-generalized logit models to the small virtual network with 5 nodes, 6 links, and 1 OD pair shown in **Figure 5.8**. **Table 5.10** depicts the link parameters of the

network and **Table 5.11** shows the travel demand in two time periods. BPR parameters also are assumed as $\alpha = 0.15$, $\beta = 4$. The logit parameter θ is 0.5, the CNL μ is 0.5 and the q-generalized logit q is 0.5.

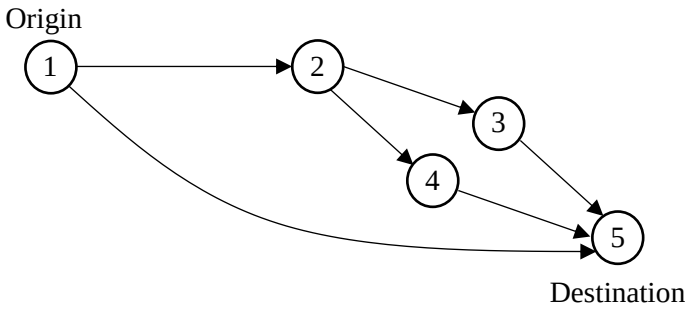


Figure 5.8 A network with 5 nodes and 6 links

Table 5.10 The link parameters of the network

No.	Link	Free-flow travel time (min)	Capacity (pcu)
1	12	20	150
2	15	30	125
3	23	5	175
4	24	5	125
5	35	5	200
6	45	5	150

Table 5.11 The travel demand of the network

No.	O-D	Travel demand in period 1(pcu)	Travel demand in period 2 (pcu)
1	1→5	200	250

The results of the application are shown in **Table 5.12**.

Table 5.12 The comparison between some semi-DTA models in the results of adjusted link traffic flows and calculation time.

Period	Results		Semi-DTA MNL model		Semi-DTA CNL model		Semi-DTA *q*-generalized logit model	
			Not approximated value	Approximated value	Not approximated value	Approximated value	Not approximated value	Approximated value
1	Adjusted flow of link (pcu)	12	125.22	125.38	119.30	119.43	130.50	130.57
		15	74.78	74.62	80.70	80.57	69.50	69.43
		23	40.25	40.15	38.61	38.46	41.64	41.61
		24	40.19	40.48	38.53	38.84	41.62	41.72
		35	35.03	34.93	33.64	33.49	36.20	36.17
		45	34.97	35.26	33.56	33.87	36.17	36.27
	Calculation time (sec)		0.17	0.02	0.17	0.03	0.33	0.02
	RMSE		0.20		0.21		0.07	
	%RMS		0.34		0.37		0.12	
2	Adjusted flow of link (pcu)	12	150.93	151.42	146.39	146.80	159.54	159.86
		15	99.07	98.58	103.61	103.20	90.46	90.14
		23	69.30	68.82	67.10	66.52	71.88	71.55
		24	68.37	69.57	66.01	67.21	71.51	72.14
		35	66.28	65.83	64.13	63.57	68.65	68.32
		45	65.38	66.62	63.10	64.33	68.22	68.85
	Calculation time (sec)		0.56	0.03	0.75	0.05	2.33	0.03
	RMSE		0.81		0.81		0.45	
	%RMS		0.93		0.95		0.51	

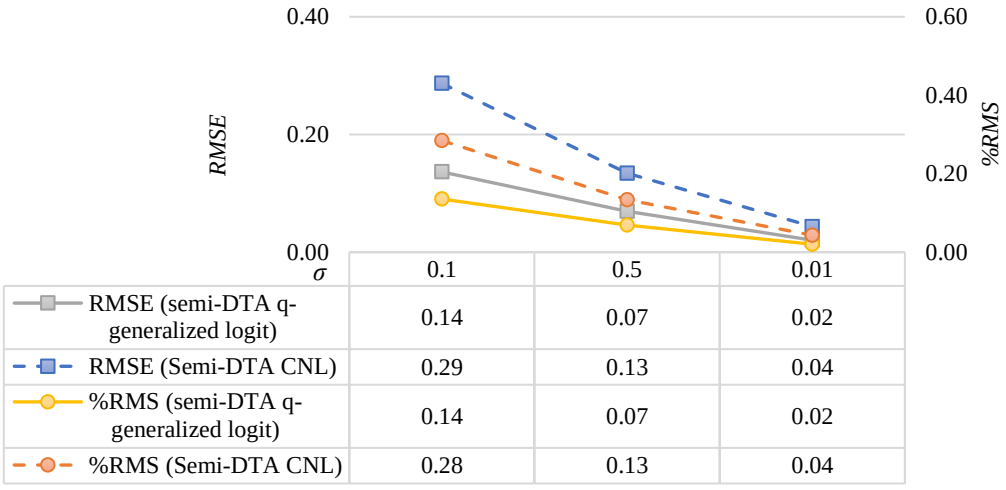
As can be seen, the sensitivity analysis method brings good approximated results in all three proposed semi-DTA models. The differences between the approximated results of sensitivity analysis methods and the results of the original models are very small with the small value of $RMSE$ and $\%RMS$ indicators. The double-looped fixed-point problem with the link travel times and the reference route flows in the original model could be solved by using the algorithm proposed in Chapter 4, but the calculation time will be a big problem in the application to large traffic network. Moreover, the calculation time of the original models will be different because the number of loops required for the convergence solution depends on the initial selection solution. The need to solve up to 2 calculation loops for each fixed-point problem will make the calculation time of the original models will be very different. In this application, the original models use the same initial

solutions in which the reference route flows are equal in all routes and the link travel times are the link free-flow travel times. In this case, the computational time of the original semi-DTA q-generalized logit model is higher than the other original semi-DTA models because the number of loops required to converge to the reference route flows equilibrium of it is two to three times the number of loops in the others. Meanwhile, the approximate models using the sensitivity analysis have negligible computation time because they do not need to solve the double-loop fixed-point problem and do not need to use the route-based variables. Thus, the use of sensitivity analysis methods can reduce computation time. In the case of the small traffic network in **Figure 5.8**, the calculation time is saved at least 88%. **Table 5.12** also shows the differences in adjusted link flow results of different models. For example, the difference between the MNL and the *q*-generalized logit models is due to the q parameter and the CNL model has solved the problem of overlapping when the traffic flow in link 12 that included in two routes has not been calculated too much compared to the remaining models. To draw the optimal application of the models to the actual road network, the models will then be applied to the Kanazawa road network.

Kanazawa road network application

Three proposed semi-DTA models are applied to Kanazawa road network with the same network and input OD data shown in Section 4.5.2 of Chapter 4. The BPR parameters are assumed as $\alpha = 1$ and $\beta = 2$, the logit parameter $\theta = 0.2$, $q = 0.5$, $\mu = 0.5$ and $h_{ij}^{r} = 1.5$. There are three time periods: 1 (6:00-7:00 AM), 2 (7:00-8:00 AM), 3 (8:00-9:00 AM). As stated in Chapter 4, because the original model cannot run in the periods 2 and 3, we also conduct tests in period 1 (with convergence error $\sigma \in [10^{-1}, 10^{-2}]$). **Figure 4.9** depicts the $RMSE$ and $\%RMS$ indicators between the adjusted link flows of the original approaches and the approximated approaches of the semi-DTA CNL and semi-DTA q-generalized logit models. The small values of two indicators $RMSE$ and $\%RMS$ show the accuracy of the approximated approaches. Besides, the smaller the convergence error of the original model is, the closer approximation to the original model is. With σ set to 0.1, the calculation time is 17.6 minutes and 15 minutes in the original semi-DTA CNL model and q-generalized logit model, respectively.

Meanwhile, the calculation times of the approximate models are only 48 seconds and 26 seconds. Thus, the proposed method will save at least 97% in the calculation time.



σ	0.1	0.5	0.01
RMSE (semi-DTA q-generalized logit)	0.14	0.07	0.02
RMSE (Semi-DTA CNL)	0.29	0.13	0.04
%RMS (semi-DTA q-generalized logit)	0.14	0.07	0.02
%RMS (Semi-DTA CNL)	0.28	0.13	0.04

Figure 5.9 *RMSE* and *%RMS* indicators between the original models and the models using the sensitivity analysis method with Kanazawa road network in period 1.

The next is the application approximated models for all three time periods. While the results of the semi-DTA using MNL were shown in Section 4.5.2, the comparison between observed link traffic flows, calculated link flows of the static SUE models, and calculated link flows of the semi-DTA models using q-generalized logit and CNL are shown in **Figures 5.10, 5.11 and 5.12**.

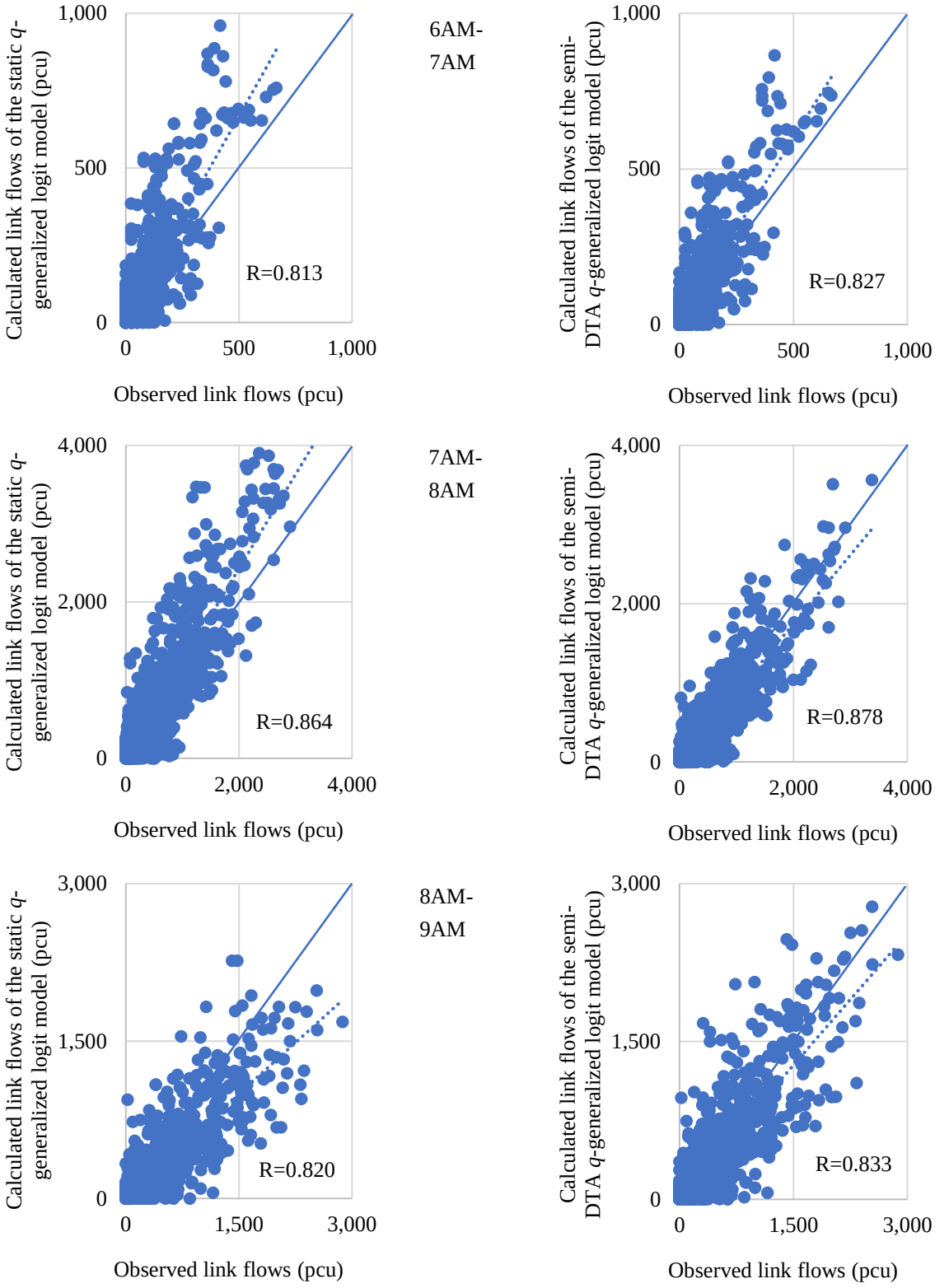


Figure 5.10 Scatter plots between the observed and calculated link flows using the static SUE and the semi-DTA SUE models based on *q*-generalized logit in each period of Kanazawa road network.

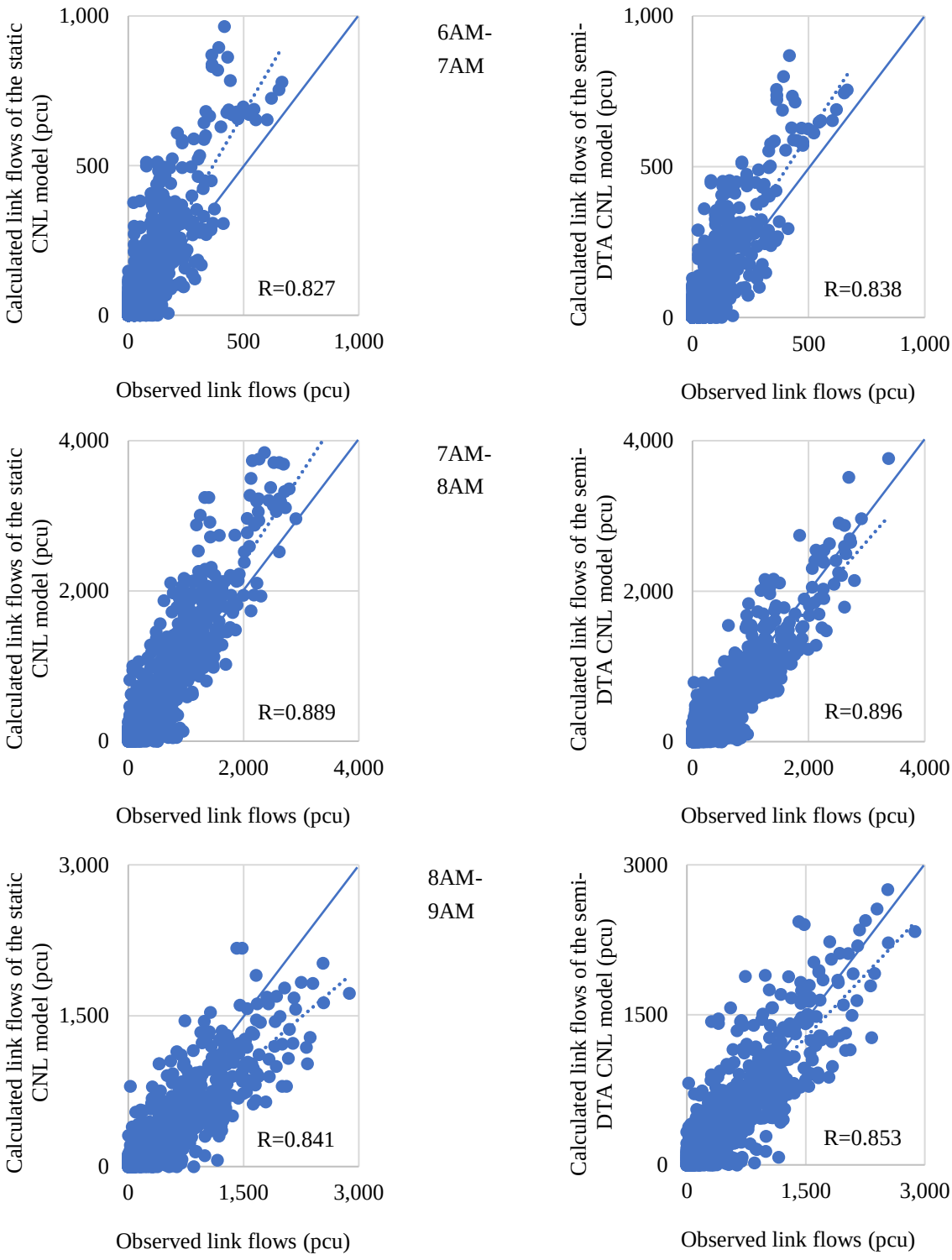


Figure 5.11 Scatter plots between the observed and calculated link flows using the static SUE and the semi-DTA SUE models based on CNL in each period of Kanazawa road network.

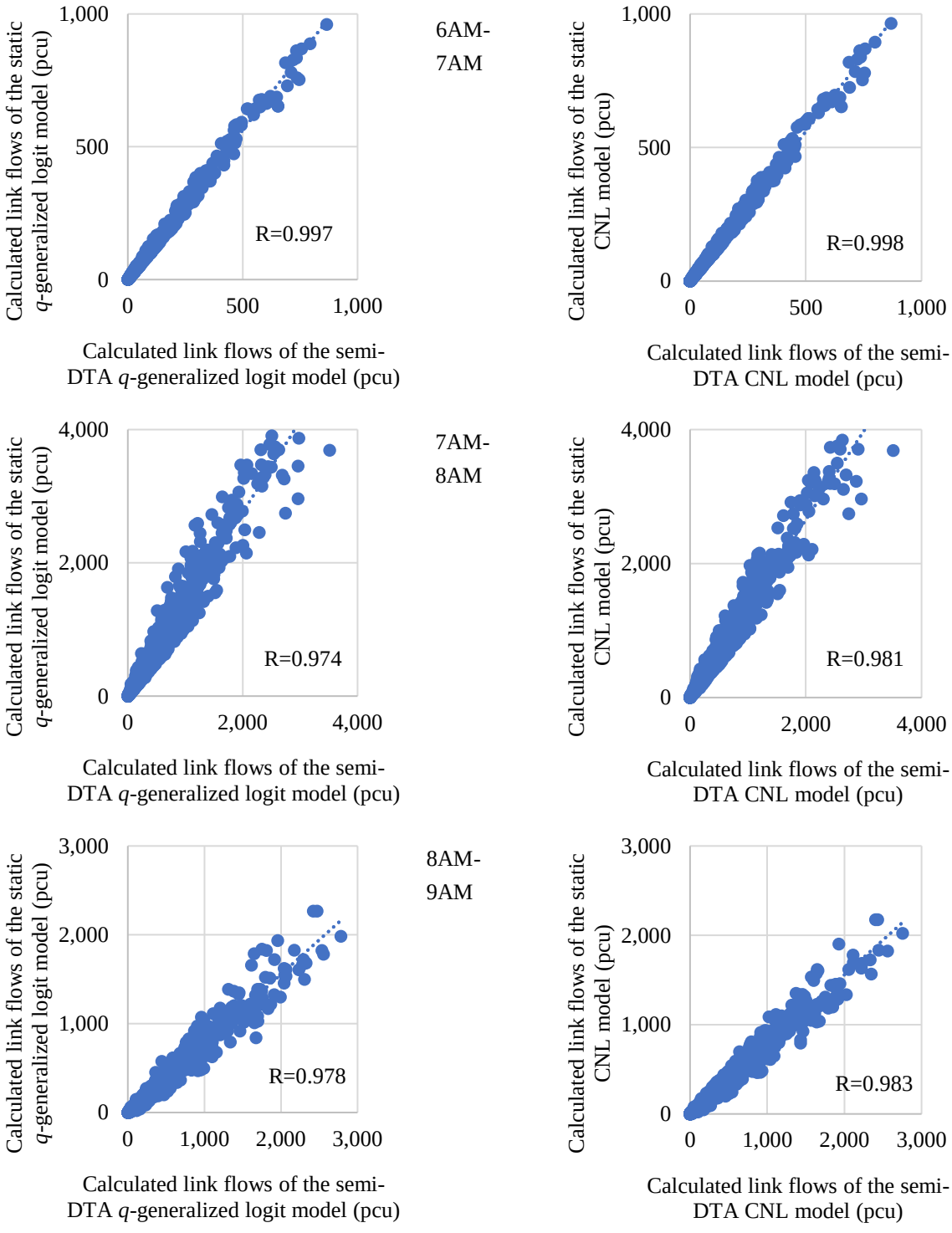


Figure 5.12 Scatter plots between the calculated link flows using the static SUE and the calculated link flows using the semi-DTA SUE in each period of Kanazawa road network.

In **Figures 5.10 and 5.11**, the results obtained from the static SUE models and the semi-DTA models are compared with the observed data. All the semi-DTA models yield better results than the static SUE ones in the correlation coefficient. Moreover, the trend of underestimating or overestimating in the semi-DTA models has decreased compared to the static models. The reason for this has been mentioned in Chapter 4 with the influence of flow propagation between periods. Furthermore, the results of the semi-DTA model are strongly correlated with the static models which are shown in **Figure 5.12**.

Also, the comparison of the computational time of the three models using the same method as shown in **Figure 5.13**.

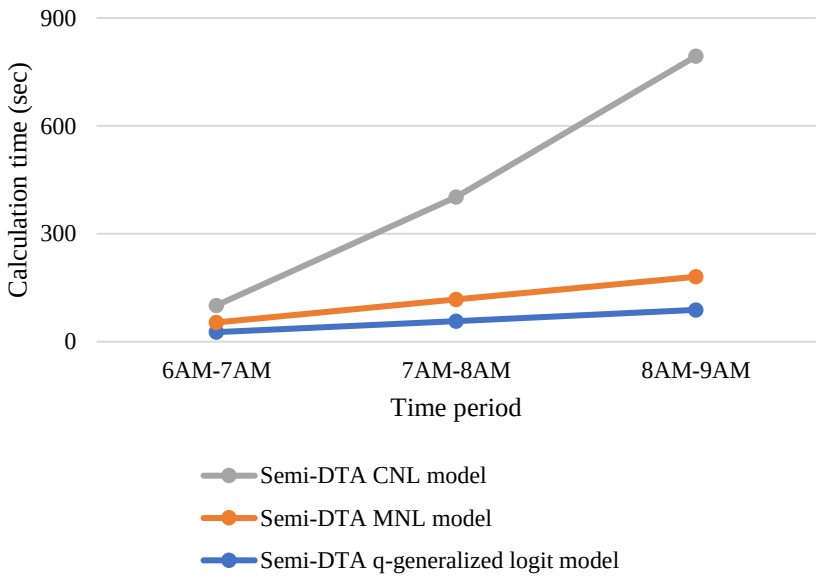


Figure 5.13 Calculational time comparison between three proposed models in each period of applying to Kanazawa road network

As can be seen, the semi-DTA *q*-generalized logit model has the lowest computation time but it is not much less than the MNL model. The difference in computational time of the two semi-DTA models with *q*-generalized logit and MNL mainly lies in solving the fixed point problem with eliminated link flows. In all three periods, the *q*-generalized logit model requires fewer loops to solve this problem. Both models choose the same SRA method to solve the fixed-point

problem, however, because the equilibrium of the two methods is different, the number of iterations in the two models will be different. Moreover, the semi-DTA CNL model takes a lot of time to calculate comparing with the other two models. The reason for this is because the CNL model requires a complex computational structure with twice runs of the DFS algorithm for each OD pair. However, it is only high when compared with the other two models. The total calculation time of the semi-DTA CNL for all three time periods is about 15.75 minutes which still reflects the computation time quite well for practical application.

Consequently, because the results of the semi-DTA q-generalized logit model depend on the parameter q and the semi-DTA CNL model depend on the parameter μ, we set the parameters q and μ from 0.1 to 0.9 and run the models to have the results of correlation coefficients that are shown in **Figure 5.14**.

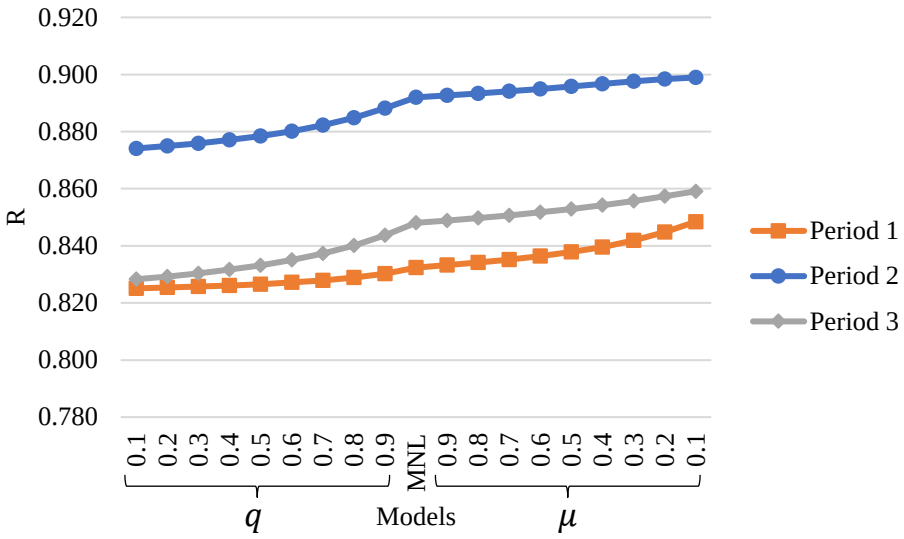


Figure 5.14 Correlation coefficients of different cases in the application to Kanazawa road network in three time periods

All three models have the results of ensuring reliability with a high correlation coefficient from 0.825 to 0.899. Interestingly, all three periods experience the same trend of the best-applied model in which the CNL model gives the best results in the correlation coefficient indicator. In these applications, all three models use the DFS algorithm with the same STOCH3-efficient routes definition. Moreover, the semi-DTA q-generalized logit model is closer to the semi-DTA MNL model when the q parameter is closer to 1 and the semi-DTA

CNL model is closer to the semi-DTA MNL model when the μ is closer to 1.

Based on two criteria of correlation coefficients and calculation time, all three semi-DTA models have high applicability. The semi-DTA MNL model still shows the simplicity of the calculation and the variety of applications of calculation methods such as the STOCH3 algorithm and the DFS algorithm. The computation time and correlation coefficient of the semi-DTA MNL model are still very good when compared with the extended approaches with two typical models: q-generalized logit and CNL models. In the semi-DTA q-generalized logit model and the semi-DTA CNL model, the estimation of optimal parameters for application is still a problem that needs to be considered in the future.

References

1. F. M. Leurent, “Curbing the Computational Difficulty of the Logit Equilibrium Assignment Model,” *Transp. Res. Part B*, vol. 31, no. 4, pp. 315–326, 1997.
2. Y. Sheffi, *Urban Transportation Networks: Equilibrium Analysis with Mathematical Programming Methods*. 1985.
3. McFadden, *Modelling the Choice of Residential Location*. 1978.
4. P. Vovsha and S. Bekhor, “Link-Nested Logit Model of Route Choice: Overcoming Route Overlapping Problem,” *Transp. Res. Rec. J. Transp. Res. Board*, vol. 1645, no. 1, pp. 133–142, Jan. 1998.
5. A. Papola, “Some Developments on the Cross-nested Logit Model,” *Transp. Res. Part B Methodol.*, vol. 38, pp. 833–851, 2004.
6. S. Bekhor and J. N. Prashker, “Stochastic User Equilibrium Formulation for Generalized Nested Logit Model,” *Transp. Res. Rec. J. Transp. Res. Board*, vol. 1752, pp. 84–90, 2001.
7. J. N. Prashker and S. Bekhor, “Congestion, Stochastic, and Similarity Effects in Stochastic: User-Equilibrium Models,” *Transp. Res. Rec. J. Transp. Res. Board*, vol. 1733, pp. 80–87, 2000.
8. S. Nakayama, “*q*-Generalized Logit Route Choice and Network Equilibrium Model,” *Present. 20th Int. Symp. Transp. Traffic Theory*, 2013.
9. S. Nakayama and M. Chikaraishi, “Unified Closed-form Expression of Logit and Weibit and its Application to a Transportation Network Equilibrium Assignment,” *Transp. Res. Part B*, vol. 81, pp. 672–685, 2015.
10. J. N. Prashker and S. Bekhor, “Stochastic User-Equilibrium Formulations for Extended-Logit Assignment Models,” *Transp. Res. Rec. J. Transp. Res. Board*, vol. 1676, pp. 145–152, 1999.
11. H. X. Liu, X. He, and B. He, “Method of Successive Weighted Averages (MSWA) and Self-regulated Averaging Schemes for Solving Stochastic User Equilibrium Problem,” *Networks Spat. Econ.*, vol. 9, pp. 485–503, 2009.

12. C. Tsallis, “What are the Numbers that Experiments Provide?” Q*uimica Nova,* 1994. 17: 468–471.

13. T. T. Bui, S. Nakayama, and H. Yamaguchi, “Sensitivity-analysis Method for *q*-generalized Logit Traffic Assignment,” *Paper presented at 99th Annual Meeting of the Transportation Research Board, Washington, D.C.*, 2020.

Chapter 6

Conclusions

This Chapter will, firstly, summarize the findings obtained in this study. Furthermore, the issues that need to be resolved in the future will be discussed.

6.1 Summary of this study

This study aims at three main objectives: Developing sensitivity analysis method for static logit-based traffic assignment SUE model to achieve high accuracy approximation of traffic assignment results, as well as reduce the calculation time of traffic assignment model; Proposing the semi-DTA model to replace STA model in which the application of sensitivity analysis method is the main method to achieve the unique results of the proposed model and increase the applicability of the proposed model with link-based and node-based approaches; Expanding the sensitivity analysis method and the semi-DTA models for the CNL and the *q*-generalized logit. The proposed method and models were applied to virtual networks and Kanazawa metropolitan area network to analyze the characteristics and applicability. The research results of the study are summarized as follows:

6.1.1 Sensitivity analysis method for logit-based stochastic user equilibrium traffic assignment

As a result of SUE did not occur when applying Dial's algorithm into Kanazawa road network, the proposed sensitivity analysis method for logit-based SUE traffic assignment based on Dial's algorithm has been difficult to apply in practice. To achieve the SUE solution, the link-based STOCH3 algorithm was proven effective. Thus, an effective alternative sensitivity analysis method for the logit-based SUE traffic assignment problem was proposed based on STOCH3 and DFS algorithms. As a source for developing a sensitivity analysis method, the DFS algorithm has been considered. And, DFS procedure loading with the MSA method has been developed as an effective alternative to the STOCH3 algorithm. From the result of the SUE solution, the derivatives of link flows with respect to some parameters were calculated. Such derivative information could be utilized for transport design and management policies. Since an equilibrium solution has been calculated, the proposed method allowed estimate acceptable approximate

solutions for any combination of changes in the parameters. Especially, the proposed method kept the computational accuracy and reduced the computational cost. The DFS-based algorithm has been developed and proved the computational time is small in application to the Kanazawa road network. However, having to calculate the number of calculations by the number of routes in implicit route-set makes this algorithm can increase the calculation time when applied to more complex road networks. To ensure a more effective, more widespread application, STOCH3 with only link-based and node-based approaches algorithm was also used to solve the sensitivity analysis procedure for saving memory and computational time. The computation time of all proposals is exceedingly small which proves applicability. The results of the numerical and real traffic networks have presented the efficiency of the proposed solution method.

6.1.2 Semi-dynamic stochastic user equilibrium traffic assignment with flow propagation based on the sensitivity analysis method

For the reason that transportation conditions are varied at separate times of the day, STA models could not fully express traffic flow, especially the time-varying congestion phenomenon. Besides, DTA models needed detailed and exact dynamic OD data, and a huge amount of the computational cost. The time-based equilibrium assignment model, called semi-DTA, keeps the characteristics of static distribution, such as the small computational load and uniqueness of the solution, while handling the traffic flow and the transition of the traffic flow between the periods. As a method for easily grasping the traffic state for each period, the day is divided into periods, and the static model is used for each period. Semi-DTA seems to be applicable in situations where travel demand management measures such as road pricing during peak hours, time-shifted work introduction, and introduction of reversible lanes are evaluated for current or near-future time points. Although it can be calculated throughout the day, it may be difficult to obtain a 24-hour OD table. However, it may be possible to give using only OD data in a limited period such as peak-period. Also, since it will be possible to estimate future traffic volume only during peak hours, it is possible to perform assignment calculations only during peak hours. In this study, the semi-DTA model with propagation was formulated as a logit-based model for applicability reasons. To solve the traffic balance problem in the proposed model, an approximation

algorithm was used to save computation time. Instead of finding a solution to a double-looped fixed-point problem, the sensitivity analysis methods have been developed. If computation time and applicability are taken into consideration, the link-based algorithm outperforms the route-based algorithm. The fact that sensitivity analysis using the link-based STOCH3 algorithm does not have to spend huge memory on route variables and the calculation time is almost independent of the number of routes in the route-set makes the algorithm applicable. There have been two proposed link-based sensitivity analysis algorithms. The second algorithm differs from the first algorithm by having to solve one more problem to find a fixed-point. In the application of the virtual road network and the actual road network, the second algorithm has produced better results than the first. The calculation time of the second algorithm is larger than the first, but it is still ridiculously small. These two approaches can be solved based on either the DFS algorithm or the STOCH3 algorithm. To express the model could represent the time-space movement of traffic congestion within-day traffic dynamics and performed computational cost efficiency, we applied the proposed model to small network and Kanazawa road network with three time periods: 6:00-7:00 AM, 7:00-8:00 AM, and 8:00-9:00 AM. A small application showed the effectiveness of the proposed method. And, the applicability of the model to the large-scale network was depicted through the application to Kanazawa road network. The results showed that the validity of the model with the good value of correlation coefficient between observed and calculated link flows, and the computational time was about 3 min in the second algorithm that represented the applicability to a large network, at least a medium-sized road network such as Kanazawa city. However, the application of the model to the Kanazawa road network points out some problems of the model that need to be further addressed in future studies. A tendency of slight underestimation in periods 2 and 3 still exists though the second approach is smaller than the first approach. The assumption that the residual flow is a continuous and increasing function of the inflow on a link is considered the reason. Thus, it is necessary to calibrate residual flows with more assumptions in future research. Besides, assigning the demand to the departure period without the consideration of the destination time in described assumption may be another reason. In future research, the issue of how to distribute travel demand into periods and other assumptions to reduce the disparity should be

further concerned.

6.1.3 Sensitivity analysis method and semi-dynamic stochastic user equilibrium traffic assignment model for cross-nested logit and *q*-generalized logit models

The weaknesses of the MNL with independent distribution and identical variance assumptions have been solved through the expansion of sensitivity analysis algorithm and semi-DTA model applied to the CNL and the *q*-generalized logit models. The results of the application to the numerical and the Kanazawa traffic networks demonstrated the efficiency of the extended proposed sensitivity analysis method, which maintained computational accuracy while reducing the computational cost. Semi-DTA with sensitivity analysis models applied to the Kanazawa road network with all three approaches (MNL, CNL, and *q*-generalized) has very good computation time in which the semi-DTA *q*-generalized logit model gives the best results on the calculation time and the semi-DTA CNL model for showing the best correlation coefficient. However, estimating some parameters such as q in the q-generalized logit model and μ in the CNL model and an integrated model considering both the overlapping and identical variance problems needs to be presented in future research.

6.2 Future work

Section 6.1 showed that the research objectives of proposing a sensitivity analysis algorithm, a semi-DTA traffic assignment model based on the sensitivity analysis algorithm, and extending the model for the various logit-based approaches reached. However, the results of the application also open some issues that need to be further addressed in future studies. The issues are summarized as follows:

- Applying the proposed sensitivity analysis method and semi-DTA model based on the sensitivity analysis to more complex networks are indispensable to fully explore the properties of the proposed method and model.

- It is also absolutely mandatory to consider more about the assumption of residual flow

- The construction of a more suitable OD matrix for the semi-DTA model is another problem.

- Some parameters need to be estimated such as q in the q-generalized logit model and μ in the CNL model.

- An integrated model needs to be proposed in which both the overlapping and identical variance problems are resolved. Furthermore, it is also crucial to develop the semi-DTA model with a combination of mode choice and route choice model

Acknowledgments

It has been nearly three years since I first came to Japan and entered Kanazawa University in the hope of pursuing further studies here. There are many valuable things I have learned here, about this beautiful country and the wonderful people. I have learned many things from life, especially, the research environment in the Laboratory of Transportation and Urban Engineering, Division of Environmental Design, Graduate school of natural science and Technology, Kanazawa University. With this acknowledgment, I would like to express the deepest appreciation and honest gratitude to my academic supervisor, Professor Shoichiro Nakayama, for his guidance, unconditional help, and visions. The lessons and help from Professor Nakayama will follow me throughout my life and my future career. I would also like to sincerely thank Dr. Hiromichi Yamaguchi, Associate Professor Yuki Takayama, Dr. Junichi Ninomiya for giving me valuable advice when I have difficulties in research and life. I am also grateful to Professor Junichi Takayama, Associate Professor Makoto Fujiu, Professor Yasuo Chikata, Professor Shen Zhen-jiang, Professor Hiroshi Masuya, Associate Professor Shunichi Kobayashi and the teachers in the environmental design division who have given me valuable lessons.

Honestly, I take this opportunity to express my sincere thanks to the kind supports of Japanese friends, Vietnamese friends, and the Kanazawa University staff since I started a new life in Japan. I also express my sincere thanks to my Laboratory mates including Mr. Koike, Mr. Nakaminami, Mr. Vu, Mr. Jun, Mr. Ali, Mr. Yuta, and many others. Moreover, I would like to express my sincere gratitude to Vietnamese Governments, Ministry of education and training, International Cooperation Department for providing financial supports in my Ph.D. course. I am also very grateful to the University of Transport and Communication where I work for enabling me to continue my studies.

Last but not least, I would like to express my sincere respect and thanks to my parents, my sisters, my brother, my beloved wife, my lovely and tiny son and daughter who have given me moral supports, unconditional love, patience, encouragements, and inspirations.

Appendixes

Appendix A. Fortran coding for using DFS algorithm to search the efficient routes

In Chapter 3, the use of a DFS-based algorithm with a recursive technique to find efficient routes is considered. The following will detail the Fortran coding of this algorithm.

A.1 DFS-based algorithm for finding STOCH3-efficient routes.

```
!c ***FINDING ALL IMPLICIT STOCH3-EFFICIENT ROUTES*******
      Integer i, j, m, ioi, count
      Integer, Parameter :: node = 7 !number of node
      Integer, Parameter :: link = 8 !number of link
      Integer, Parameter :: nod = 1 !number of OD pairs
      Integer s(link), e(link) !start and end node and link number
      Integer ori(nod), des(nod) !origin, destination of od pairs
      Integer head(0:node), alink(link) ! adjacent links
      Integer omega(nod, link)
!omega(nod,link): marking efficient links of each od pair
      Integer passlink(6), npath(nod)
!*********************************************************
      Open (1, File='1.network.txt', Action='read')
      Open (3, File='3.demand.txt', Action='read')
      Open (5, File='5.route-set.txt')

      Do i = 1, link
        Read (1, *) s(i), e(i) !s(i):start node, e(i):end node
      End Do
      Do i = 1, nod
        Read (3, *) ori(i), des(i)
      End Do
! create adjacent links
      head(0) = 0
      count = 0
      Do i = 1, node
        Do j = 1, link
          If (i==s(j)) Then
            count = count + 1
```

```
30:              alink(count) = j
31:            End If
32:          End Do
33:          head(i) = count
34:        End Do
35:        omega = 1
36:        npath = 0
37:    !***** Finding efficient paths ******************************
38:        Do ioi = 1, nod
39:          j = ori(ioi) !current node
40:          m = 0
41:          passlink = 0
42:          Call creaters(1)
43:        End Do
44:    !*****SUBROUTINE************************************************************
45:      Contains
46:    !c     Creating route-set with DFS algorithm
47:        Recursive Subroutine creaters(dfs)
48:          Integer dfs, iii
49:    !dfs: used to specify the order of node to be browsed from the origin
50:    !iii:used to point out the position of adjacent links of current node
51:
52:          If (j==des(ioi)) Then !Stoppping condition
53:            npath(ioi) = npath(ioi) + 1
54:            Write (5, *) passlink
55:          Else
56:            Do iii = head(j-1) + 1, head(j) !position of adjacent links of current node
57:              If (omega(ioi,alink(iii))/=0) Then !check efficient link condition
58:                m = m + 1
59:                j = e(alink(iii)) !next node
60:                passlink(m) = alink(iii) !mark traversed link
61:                Call creaters(dfs+1) !recursive technique
62:                passlink(m) = 0 !delete backtracked link
63:                m = m - 1
64:              End If
65:            End Do
66:          End If
67:        End Subroutine
68:    !*****FINISH SUBROUTINE**************************************
69:      End Program
```

A.2 DFS-based algorithm for finding modified STOCH3-efficient routes.

```
1:    !c *** FINDING ALL MODIFIED IMPLICIT STOCH3-EFFICIENT ROUTES*******
2:          Integer i, j, m, ioi, count
3:          Integer, Parameter :: node = 7 !number of node
4:          Integer, Parameter :: link = 9 !number of link
5:          Integer, Parameter :: nod = 1 !number of OD pairs
6:          Integer s(link), e(link) !start and end node and link number
7:          Integer ori(nod), des(nod) !origin, destination of od pairs
8:          Integer head(0:node), alink(link) ! adjacent links
9:          Integer omega(nod, link)
10:   !omega(nod,link): marking efficient links of each od pair
11:         Integer passlink(6), npath(nod), mark(node)
12:   !*************************************************
13:         Open (1, File='1.network.txt', Action='read')
14:         Open (3, File='3.demand.txt', Action='read')
15:         Open (5, File='5.route-set.txt')
16:
17:         Do i = 1, link
18:           Read (1, *) s(i), e(i) !s(i):start node, e(i):end node
19:         End Do
20:         Do i = 1, nod
21:           Read (3, *) ori(i), des(i)
22:         End Do
23:
24:   ! create adjacent links
25:         head(0) = 0
26:         count = 0
27:         Do i = 1, node
28:           Do j = 1, link
29:             If (i==s(j)) Then
30:               count = count + 1
31:               alink(count) = j
32:             End If
33:           End Do
34:           head(i) = count
35:         End Do
36:         omega = 1
37:         npath = 0
38:   !***** Finding efficient paths ******************************
39:         Do ioi = 1, nod
40:           j = ori(ioi) !current node
41:           m = 0
42:           passlink = 0
43:           mark = 0
44:           Call creaters(1)
45:         End Do
46:
47:   !*****SUBROUTINE*************************************************************
48:       Contains
49:   !c     Creating route-set with DFS algorithm
50:         Recursive Subroutine creaters(dfs)
51:           Integer dfs, iii
```

```
52:   !dfs: used to specify the order of node to be browsed from the origin
53:   !iii:used to point out the position of adjacent links of current node
54:
55:         If (j==des(ioi)) Then !Stoppping condition
56:           npath(ioi) = npath(ioi) + 1
57:           Write (5, *) passlink
58:         Else
59:           Do iii = head(j-1) + 1, head(j) !position of adjacent links of current node
60:             If (omega(ioi,alink(iii))/=0 .And. mark(e(alink(iii)))==0) Then !check efficient link
      condition
61:               m = m + 1
62:               mark(s(alink(iii))) = 1
63:               j = e(alink(iii)) !next node
64:               passlink(m) = alink(iii) !mark traversed link
65:               Call creaters(dfs+1) !recursive technique
66:               mark(s(passlink(m))) = 0
67:               passlink(m) = 0 !delete backtracked link
68:               m = m - 1
69:             End If
70:           End Do
71:         End If
72:       End Subroutine
73:   !*****FINISH SUBROUTINE**********************************
74:     End Program
```

Appendix B. STOCH3 algorithm for each OD pair

Leurent proposed the STOCH3 algorithm running for each origin node r with the MSA method. This algorithm can be also run for each OD pair rs as follows:

Program variables:

Ω_{ij}^{r}: Indicator variable ($\Omega_{ij}^{r} \coloneqq 1$ if link ij is efficient from origin node r and $\Omega_{ij}^{r} \coloneqq 0$ otherwise).

O_{i}^{r}: The ith node in the order of increasing access time C_{ri}^{0} from origin node r.

a_{ij}: Likelihood of link ij ($a_{ij} = \exp(-\theta t_{ij})$).

wl_{ij}^{rs}: Link weight that presents the importance of link ij in contributing to efficient routes from origin node r to destination node s.

wn_{n}^{rs}: Node weight that presents the importance of node n in contributing to efficient routes from origin node r to destination node s.

xn_{n}^{rs}: Flow passing through node n from origin node r to destination node s.

xl_{ij}^{rs}: The flow of link ij from origin node r to destination node s.

Index rs could be removed on the variables wl_{ij}^{rs}, wn_{n}^{rs}, xn_{n}^{rs} and xl_{ij}^{rs} because they are reused after dealing with OD pair rs and do not need to store.

The STOCH3 procedure with MSA method:

Step 0: Preliminaries

(a) Set iteration counter $l \coloneqq 1$ and maximum value of the change in link flow from the previous iteration $l-1$ to the current one l ($\sigma \coloneqq 10^{-3}$)

(b) Based on the free-flow travel time $\{t_{ij}^{0}\}$, for each origin node r, calculate the minimum travel time to all other nodes. Determine C_{rn}^{0} for each node n and the order of increasing access time from origin node r.

(c) For each link ij, set $\Omega_{ij}^{r} \coloneqq 1$ if $(1 + h_{ij}^{r})(C_{rj}^{0} - C_{ri}^{0}) \geq t_{ij}^{0}$,

otherwise $\Omega_{ij}^{r} \coloneqq 0$ and set $t_{ij} = t_{ij}^{0}$ and $x_{ij}^{(l)} = 0$.

Step 1: Set the link likelihood

For each link ij, set $a_{ij} \coloneqq \exp(-\theta\, t_{ij})$

Steps 2, 3, and 4 are to be run for each OD pair.

Step 2: Forward pass

(a) Set all wl_{ij}^{rs} and wn_{n}^{rs} to 0. Set wn_{r}^{rs}:=1

(b) For each link ij taken in the order of increasing reference access time from r, if $\Omega_{ij}^{r} = 1$ then compute $wl_{ij}^{rs} \coloneqq a_{ij} wn_{i}^{rs}$ and add wl_{ij}^{rs} to wn_{j}^{rs}, otherwise do nothing

Step 3: Backward pass

(a) Set $xn_{s}^{rs} \coloneqq Q_{0}^{rs}$ for the destination node.

(b) For each link ij taken in the order of decreasing reference access time from r, if $\Omega_{ij}^{r} = 1$ then compute $xl_{ij}^{rs} \coloneqq xn_{j}^{rs} wl_{ij}^{rs} / wn_{j}^{rs}$ and add xl_{ij}^{rs} to xn_{i}^{rs}, otherwise do nothing.

Step 4: Contribution to total link-flows

Calculate $x_{ij}^{(l)} := x_{ij}^{(l)} + xl_{ij}^{rs}$.

Step 5: Link travel time and link impedances update

Set $t_{ij}^{(l)} \coloneqq t_{ij}(x_{ij}^{(l)})$ and $a_{ij} \coloneqq \exp(-\theta\, t_{ij}^{(l)})$.

Step 6: Direction finding

Based on link travel time $\{t_{ij}^{(l)}\}$ and a_{ij}, steps 6.1, 6.2, 6.3 are the same as steps 2, 3, and 4 for each OD pair, respectively. Yielding an auxiliary link flow pattern $\{yx_{ij}^{(l)}\}$.

Step 7: Link flow update

(a) Choose a sequence of real numbers such that $(0 < \lambda^{(l)} < 1)$.

(b) Let $rg_{ij}^{(l)} = yx_{ij}^{(l)} - x_{ij}^{(l)}$.

(c) Set $x_{ij}^{(l+1)} = x_{ij}^{(l)} + \lambda^{(l)} rg_{ij}^{(l)}$.

Step 8: Stopping the test

If $max_{ij}\left\{rg_{ij}^{(l)}\right\} \leq \sigma$, stop. The solution is $\{x_{ij}^{(l)}\}$. Otherwise, set $l := l + 1$ and go to step 5.

The next proves that the STOCH3 algorithm can run on each OD pair without changing the result of the traffic assignment on each efficient link ij.

We denote as $K_r^{u,v}$ the set of efficient routes from node u to node v with respect to origin node r, which are the routes from node u to node v that only include links ij such that $\Omega_{ij}^r = 1$. Thus, K_r^{rs} is the set of the STOCH3-efficient routes from origin node r to destination node s.

Lemma 1. $\forall ij \in A$, it is clear that:

$$\sum_{k \in K_r^{uv}} \delta_{ij,k}^{uv} \exp(-\theta c_k^{uv}) = \sum_{k \in K_r^{ui}} \exp\left(-\theta c_k^{ui}\right) \sum_{k \in K_r^{iv}} \delta_{ij,k}^{iv} \exp\left(-\theta c_k^{iv}\right) \quad \text{(B.1)}$$

$$\sum_{k \in K_r^{uv}} \delta_{ij,k}^{uv} \exp(-\theta c_k^{uv}) = \sum_{k \in K_r^{uj}} \delta_{ij,k}^{uj} \exp\left(-\theta c_k^{uj}\right) \sum_{k \in K_r^{jv}} \exp\left(-\theta c_k^{jv}\right) \quad \text{(B.2)}$$

Proof.

In the case $\Omega_{ij}^r = 0$, $\delta_{ij,k}^{uv} = 0 \; \forall \, k \in K_r^{uv}$, $\delta_{ij,k}^{uj} = 0 \; \forall \, k \in K_r^{uj}$ and $\delta_{ij,k}^{iv} = 0 \; \forall \, k \in K_r^{iv}$. Thus, **(B.1)** and **(B.2)** hold.

In the case $\Omega_{ij}^r = 1$:

- If $\delta_{ij,k}^{uv} = 0 \; \forall \, k \in K_r^{uv}$, the following will occur:

  - If $K_r^{ui} = \emptyset$, $\sum_{k \in K_r^{ui}} \exp\left(-\theta c_k^{ui}\right)$ and **(B.1)** holds.
  - If $K_r^{ui} \neq \emptyset$ and $K_r^{iv} = \emptyset$, $\sum_{k \in K_r^{iv}} \exp\left(-\theta c_k^{iv}\right) = 0$ and **(B.1)** holds.
  - If $K_r^{ui} \neq \emptyset$ and $K_r^{iv} = \emptyset$, $\sum_{k \in K_r^{iv}} \exp\left(-\theta c_k^{iv}\right) = 0$ and **(B.1)** holds.
  - If $K_r^{ui} \neq \emptyset$ and $K_r^{iv} \neq \emptyset$, $K_r^{uv} \neq \emptyset$. Without loss of generality, we can assume that there is at least one efficient route k from node i to node v with respect to origin node r including link ij ($\delta_{ij,k}^{iv} \neq 0$). Because $K_r^{ui} \neq \emptyset$, $K_r^{iv} \neq \emptyset$ and $K_r^{uv} \neq \emptyset$, there is also at least one efficient route from node u to node v with respect to origin node r

including link ij $(\delta_{ij,k}^{uv} \neq 0)$. This is contrary to the original hypothesis $(\delta_{ij,k}^{uv} = 0 \; \forall \; k \in K_r^{uv})$, deducing $\delta_{ij,k}^{iv} = 0 \; \forall \; k \in K_r^{iv}$ and **(B.1)** holds.

- If $K_r^{uj} = \emptyset$, $\sum_{k \in K_r^{uj}} \exp\left(-\theta c_k^{uj}\right)$ and **(B.2)** holds.
- If $K_r^{uj} \neq \emptyset$ and $K_r^{jv} = \emptyset$, $\sum_{k \in K_r^{jv}} \exp\left(-\theta c_k^{jv}\right) = 0$ and **(B.2)** holds.
- If $K_r^{uj} \neq \emptyset$ and $K_r^{jv} = \emptyset$, $\sum_{k \in K_r^{jv}} \exp\left(-\theta c_k^{jv}\right) = 0$ and **(B.2)** holds.
- If $K_r^{uj} \neq \emptyset$ and $K_r^{jv} \neq \emptyset$, $K_r^{uv} \neq \emptyset$. Without loss of generality, we can assume that there is at least one efficient route k from node u to node j with respect to origin node r including link ij $(\delta_{ij,k}^{uj} \neq 0)$. Because $K_r^{uj} \neq \emptyset$, $K_r^{jv} \neq \emptyset$ and $K_r^{uv} \neq \emptyset$, there is also at least one efficient route from node u to node v with respect to origin node r including link ij $(\delta_{ij,k}^{uv} \neq 0)$. This is contrary to the assumption $(\delta_{ij,k}^{uv} = 0 \; \forall \; k \in K_r^{uv})$, deducing $\delta_{ij,k}^{uj} = 0 \; \forall \; k \in K_r^{uj}$ and **(B.2)** holds.

- If there is at least one efficient route k from node u to node v with respect to origin node r including link ij $(\delta_{ij,k}^{uv} \neq 0)$, route k could be decomposed into some subroutes:

route k= route k_1+ $\{ij\}$ + route k_3 = route k_2+ route k_3 = route k_1+ route k_4

where route k_1 includes the links of route k from node u to node i, route k_2 includes the links of route k from node u to node j, route k_3 includes the links of route k from node j to node v and route k_4 includes the links of route k from node i to node v. Therefore, $\delta_{ij,k_2}^{uj} \neq 0$, $\delta_{ij,k_4}^{iv} \neq 0$, and the travel time on the route k could be represented as:

$$c_k^{uv} = c_{k_2}^{uj} + c_{k_3}^{jv} = c_{k_1}^{ui} + c_{k_4}^{iv}$$

Consequently,

$$\begin{aligned} \delta_{ij,k}^{uv} \exp(-\theta c_k^{uv}) &= \delta_{ij,k_2}^{uj} \exp\left(-\theta c_{k_2}^{uj}\right) \exp\left(-\theta c_{k_3}^{jv}\right) \\ &= \exp\left(-\theta c_{k_2}^{ui}\right) \left[\delta_{ij,k_1}^{iv} \exp\left(-\theta c_{k_4}^{iv}\right)\right] \end{aligned}$$

Summing over indices k_1, k_4 on the right-hand side and k on the left-hand

side leads to **(B.1)** and summing over indices k_2, k_3 on the right-hand side and k on the left-hand side leads to **(B.2)**.

Lemma 2.

$$wl_{ij}^{rs} = \sum_{k \in K_r^{rj}} \delta_{ij,k}^{rj} \exp(-\theta c_k^{rj}), \tag{B.3}$$

$$wn_j^{rs} = \sum_{k \in K_r^{rj}} \exp(-\theta c_k^{rj}) \text{ if } j \neq r \tag{B.4}$$

Proof.

If $j \neq r$, wn_j^{rs} is calculated in the forward pass:

$$wn_j^{rs} = \sum_{i \in N_j^{in}} wl_{ij}^{rs}$$

If **(B.3)** is true, because all links connected to node j will be taken into consideration,

$$wn_j^{rs} = \sum_{i \in N_j^{in}} \sum_{k \in K_r^{rj}} \delta_{ij,k}^{rj} \exp(-\theta c_k^{rj}) = \sum_{k \in K_r^{rj}} \exp(-\theta c_k^{rj}).$$

Therefore, **(B.4)** is proved if **(B.3)** is proved. **(B.3)** will be proved recursively as follows:

If $\Omega_{ij}^r = 0$, $\delta_{ij,k}^{rj} = 0 \; \forall \; k \in K_r^{rj}$ and $wl_{ij}^{rs} = 0$. Thus, **(B.3)** holds.

For link ij such that $\Omega_{ij}^r = 1$: In the case where $i = r$, $\delta_{rj,k}^{rj} = 1$, $\sum_{k \in K_r^{rj}} \delta_{rj,k}^{rj} \exp(-\theta c_k^{rj}) = \exp(-\theta t_{rj})$ and $wl_{rj}^{rs} = \exp(-\theta t_{rj}) wn_r^{rs} = \exp(-\theta t_{rj})$. Hence, **(B.3)** holds. Consequently, **(B.4)** holds for all nodes j connecting from r. By recursion, we can assume that **(B.3)** holds for all links gh such that $C_{rg}^0 < C_{ri}^0$. since **(B.3)** holds for those links gh that are considered before link ij , **(B.4)** holds for node i and the forward pass calculates $wl_{ij}^{rs} = \exp(-\theta t_{ij}) wn_i^{rs} = \exp(-\theta t_{ij}) \sum_{k \in K_r^{ri}} \exp(-\theta c_k^{ri})$. Because $\exp(-\theta t_{ij}) = \sum_{k \in K_r^{ij}} \delta_{ij,k}^{ij} \exp(-\theta c_k^{ij})$ if $\Omega_{ij}^r = 1$, using **(B.1)** gives

$$wl_{ij}^{rs} = \sum_{k \in K_r^{ij}} \delta_{ij,k}^{ij} \exp(-\theta c_k^{ij}) \sum_{k \in K_r^{ri}} \exp(-\theta c_k^{ri}) = \sum_{k \in K_r^{rj}} \delta_{ij,k}^{rj} \exp(-\theta c_k^{rj})$$

Therefore, **(B.3)** holds.

Lemma 3.

$$xl_{ij}^{rs} = Q_0^{rs} \frac{\sum_{k \in K_r^{rs}} \delta_{ij,k}^{rs} \exp(-\theta c_k^{rs})}{\sum_{k \in K_r^{rs}} \exp(-\theta c_k^{rs})} \tag{B.5}$$

Proof.

Recursive proof of **(B.5)** will be given regarding the links ij in the order of decreasing costs C_{rj}^0.

Firstly, if $\Omega_{ij}^r = 0$, $wl_{ij}^{rs} = 0$ and $xl_{ij}^{rs} = 0$. In that case, because there is no efficient route from r to s including link ij, the right-hand side of **(B.5)** also equals 0. Hence, **(B.5)** holds.

Secondly, if $j = s$. We have:

$$xl_{is}^{rs} = \frac{wl_{is}^{rs}}{wn_s^{rs}} xn_s^{rs} = \frac{wl_{is}^{rs}}{wn_s^{rs}} Q_0^{rs}$$

Using **(B.3)**, **(B.4)** and the backward pass yields

$$xl_{is}^{rs} = Q_0^{rs} \frac{\sum_{k \in K_r^{rs}} \delta_{ij,k}^{rs} \exp(-\theta c_k^{rs})}{\sum_{k \in K_r^{rs}} \exp(-\theta c_k^{rs})}$$

Finally, assuming that **(B.5)** holds $\forall gh$ such that $C_{rh}^0 > C_{rj}^0$, we calculate **(B.5)** for link ij. This assumption also indicates that **(B.5)** holds for every links jm with $m \in N_j^{out}$. Because if $C_{rm}^0 > C_{rj}^0$, **(B.5)** holds from the assumption, and if $C_{rm}^0 \leq C_{rj}^0$, $\Omega_{rj}^r = 0$ and **(B.5)** also holds from the first case of this proof. Hence, using **(B.3)**, **(B.4)** and the backward pass yields

$$xl_{ij}^{rs} = \frac{wl_{ij}^{rs}}{wn_j^{rs}} xn_j^{rs} = \frac{wl_{ij}^{rs}}{wn_j^{rs}} \sum_{m \in N_j^{out}} xl_{jm}^{rs}$$

$$= \frac{\sum_{k \in K_r^{rj}} \delta_{ij,k}^{rj} \exp(-\theta c_k^{rj})}{\sum_{k \in K_r^{rj}} \exp(-\theta c_k^{rj})} \sum_{m \in N_j^{out}} Q_0^{rs} \frac{\sum_{k \in K_r^{rs}} \delta_{jm,k}^{rs} \exp(-\theta c_k^{rs})}{\sum_{k \in K_r^{rs}} \exp(-\theta c_k^{rs})}$$

$$= Q_0^{rs} \frac{\sum_{k \in K_r^{rj}} \delta_{ij,k}^{rj} \exp(-\theta c_k^{rj})}{\sum_{k \in K_r^{rj}} \exp(-\theta c_k^{rj})} \frac{\sum_{m \in N_j^{out}} \sum_{k \in K_r^{rs}} \delta_{jm,k}^{rs} \exp(-\theta c_k^{rs})}{\sum_{k \in K_r^{rs}} \exp(-\theta c_k^{rs})}$$

From **(B.1)**,

$$\sum_{k \in K_r^{rs}} \delta_{jm,k}^{rs} \exp(-\theta c_k^{rs}) = \sum_{k \in K_r^{rj}} \exp\left(-\theta c_k^{rj}\right) \sum_{k \in K_r^{js}} \delta_{jm,k}^{js} \exp\left(-\theta c_k^{js}\right)$$

Thus,

$$\sum_{m \in N_j^{out}} \sum_{k \in K_r^{rs}} \delta_{jm,k}^{rs} \exp(-\theta c_k^{rs})$$

$$= \sum_{m \in N_j^{out}} \sum_{k \in K_r^{rj}} \exp\left(-\theta c_k^{rj}\right) \sum_{k \in K_r^{js}} \delta_{jm,k}^{js} \exp\left(-\theta c_k^{js}\right)$$

$$= \sum_{k \in K_r^{rj}} \exp\left(-\theta c_k^{rj}\right) \sum_{m \in N_j^{out}} \sum_{k \in K_r^{js}} \delta_{jm,k}^{js} \exp\left(-\theta c_k^{js}\right) \quad \text{(B.6)}$$

$$= \sum_{k \in K_r^{rj}} \exp\left(-\theta c_k^{rj}\right) \sum_{m \in N_j^{out}} \sum_{k \in K_r^{js}} \delta_{jm,k}^{js} \exp\left(-\theta c_k^{js}\right)$$

$$= \sum_{k \in K_r^{rj}} \exp\left(-\theta c_k^{rj}\right) \sum_{k \in K_r^{js}} \exp\left(-\theta c_k^{js}\right)$$

The calculation of xl_{ij}^{rs} becomes

$$xl_{ij}^{rs} = Q_0^{rs}\frac{\sum_{k\in K_r^{rj}} \delta_{ij,k}^{rj} \exp\left(-\theta c_k^{rj}\right)}{\sum_{k\in K_r^{rj}} \exp\left(-\theta c_k^{rj}\right)} \frac{\sum_{k\in K_r^{rj}} \exp\left(-\theta c_k^{rj}\right) \sum_{k\in K_r^{js}} \exp\left(-\theta c_k^{js}\right)}{\sum_{k\in K_r^{rs}} \exp(-\theta c_k^{rs})}$$

$$= Q_0^{rs}\frac{\sum_{k\in K_r^{rj}} \delta_{ij,k}^{rj} \exp(-\theta c_k^{rj}) \sum_{k\in K_r^{js}} \exp\left(-\theta c_k^{js}\right)}{\sum_{k\in K_r^{rs}} \exp(-\theta c_k^{rs})}$$

Using **(B.2)** gives

$$xl_{ij}^{rs} = Q_0^{rs}\frac{\sum_{k\epsilon K_r^{rs}} \delta_{ij,k}^{rs} \exp(-\theta c_k^{rs})}{\sum_{k\epsilon K_r^{rs}} \exp(-\theta c_k^{rs})}$$

Thus, running STOCH3 algorithm for each OD pair, the link traffic flow of each efficient link ij, x_{ij}^{rs}, can be calculated as follows:

$$x_{ij}^{rs} = \frac{wl_{ij}^{rs}}{wn_j^{rs}} xn_j^{rs} \tag{B.7}$$

Consequently, let we transform $x_{ij\to gh}^{rs}$ and show how to use x_{ij}^{rs} and the STOCH3 algorithm to calculate this variable. If we use STOCH3-efficient route definition,

$$x_{ij\to gh}^{rs} = Q_0^{rs}\frac{\sum_{k\epsilon K_r^{rs}} \delta_{ij\to gh,k}^{rs} \exp(-\theta c_k^{rs})}{\sum_{k\epsilon K_r^{rs}} \exp(-\theta c_k^{rs})} \tag{B.8}$$

where $\delta_{ij\to gh,k}^{rs}$ denotes the double link-route incidence variable. If the route k includes both links ij and gh in such a way that link ij is used before link gh then

$\delta^{rs}_{ij\to gh,k} = 1$, otherwise$\delta^{rs}_{ij\to gh,k} = 0$.

Note that $x^{rs}_{ij\to gh}$ just needs to be calculated when $x^{rs}_{ij} \neq 0$ and $j \neq s$, which also means that there exists at least one efficient route from node j to node s with respect to origin node r. Without loss of generality, we can multiply both the numerator and the denominator of **Equation (B.8)** by $\sum_{k\in K_r^{js}} \exp\left(-\theta c_k^{js}\right)$, we have:

$$x^{rs}_{ij\to gh} = Q^{rs}_0 \frac{\sum_{k\in K_r^{rs}} \delta^{rs}_{ij\to gh,k} \exp(-\theta c_k^{rs}) \sum_{k\in K_r^{js}} \exp\left(-\theta c_k^{js}\right)}{\sum_{k\in K_r^{rs}} \exp(-\theta c_k^{rs}) \sum_{k\in K^{rs}} \exp(-\theta c_k^{rs})} \tag{B.9}$$

In addition, we can use the following expression

$$\sum_{k\in K_r^{rs}} \exp(-\theta c_k^{rs})\, \delta^{rs}_{ij\to gh,k} = \sum_{k\in K_r^{rj}} \delta^{rj}_{ij,k} \exp(-\theta c_k^{rj}) \sum_{k\in K_r^{js}} \delta^{js}_{gh,k} \exp(-\theta c_k^{js}) \tag{B.10}$$

Proof.

If $\delta^{rs}_{ij\to gh,k} = 0\ \forall k \in K_r^{rs}, \sum_{k\in K_r^{rs}} \exp(-\theta c_k^{rs})\, \delta^{rs}_{ij\to gh,k} = 0$. Either $\delta^{rj}_{ij,k} = 0\ \forall k \in k \in K_r^{rj}$ or $\delta^{js}_{gh,k}\ \forall k \in k \in K_r^{js}$ give **(B.10)**. We can assume that there is at least one efficient route k_1 from node r to node j with respect to origin node r including link ij ($\delta^{rj}_{ij,k} \neq 0$) and at least one efficient route k_2 from node j to node s with respect to origin node r including link gh ($\delta^{js}_{gh,k} \neq 0$). It implies there is at least one efficient route k from r to s containing both links ij and gh that satisfy link ij used before link gh (it is contrary to the hypothesis $\delta^{rs}_{ij\to gh,k} = 0\ \forall k \in K_r^{rs}$). Thus, the right-hand side of **(B.10)** equals to 0 in this case and (**B.10)** holds.

If there is at least one efficient route k satisfying $\delta^{rs}_{ij\to gh,k} = 1$. Because the route k from r to s containing both links ij and gh that satisfy link ij used before link gh can be divided into two subroutes: route k_1 from r to j containing link ij and route k_2 from j to s containing link gh, $\delta^{rj}_{ij,k_1} \neq 0$, $\delta^{js}_{gh,k_2} \neq 0$, and the travel time on the route k could be represented as:

$$c_k^{rs} = c_{k_1}^{rj} + c_{k_2}^{js}$$

Consequently,

$$\delta_{ij\to gh,k}^{rs}\exp(-\theta c_k^{uv}) = \delta_{ij,k_1}^{rj}\exp\left(-\theta c_{k_1}^{rj}\right)\delta_{gh,k_2}^{js}\exp\left(-\theta c_{k_2}^{js}\right)$$

Summing over indices k_1, k_2 on the right-hand side and k on the left-hand side leads to **(B.10).**

Using **(B.10), (B.9)** becomes

$$x_{ij\to gh}^{rs} = Q_0^{rs}\frac{\sum_{k\in K_r^{rj}}\delta_{ij,k}^{rj}\exp\left(-\theta c_k^{rj}\right)\sum_{k\in K_r^{js}}\delta_{gh,k}^{js}\exp\left(-\theta c_k^{js}\right)}{\sum_{k\in K_r^{rs}}\exp(-\theta c_k^{rs})}\frac{\sum_{k\in K_r^{js}}\exp\left(-\theta c_k^{js}\right)}{\sum_{k\in K^{rs}}\exp(-\theta c_k^{rs})} \quad \text{(B.11)}$$

$$= Q_0^{rs}\frac{\sum_{k\in K_r^{js}}\delta_{gh,k}^{js}\exp\left(-\theta c_k^{js}\right)}{\sum_{k\in K_r^{rs}}\exp(-\theta c_k^{rs})}\frac{\sum_{k\in K_r^{rj}}\delta_{ij,k}^{rj}\exp\left(-\theta c_k^{rj}\right)\sum_{k\in K_r^{js}}\exp\left(-\theta c_k^{js}\right)}{\sum_{k\in K^{rs}}\exp(-\theta c_k^{rs})}$$

According to **(B.2)**,

$$\sum_{k\in K_r^{rs}}\delta_{ij,k}^{rs}\exp(-\theta c_k^{rs}) = \sum_{k\in K_r^{rj}}\delta_{ij,k}^{rj}\exp\left(-\theta c_k^{rj}\right)\sum_{k\in K_r^{js}}\exp\left(-\theta c_k^{js}\right)$$

Thus, **(B.11)** is equivalent to

$$x_{ij\to gh}^{rs} = Q_0^{rs}\frac{\sum_{k\in K_r^{js}}\delta_{gh,k}^{js}\exp\left(-\theta c_k^{js}\right)}{\sum_{k\in K_r^{js}}\exp\left(-\theta c_k^{js}\right)}\frac{\sum_{k\in K_r^{rs}}\delta_{ij,k}^{rs}\exp(-\theta c_k^{rs})}{\sum_{k\in K^{rs}}\exp(-\theta c_k^{rs})} \quad \text{(B.12)}$$

$$= \frac{\sum_{k\in K_r^{js}}\delta_{gh,k}^{js}\exp\left(-\theta c_k^{js}\right)}{\sum_{k\in K_r^{js}}\exp\left(-\theta c_k^{js}\right)}\frac{\sum_{k\in K_r^{rs}}\delta_{ij,k}^{rs}\exp(-\theta c_k^{rs})}{\sum_{k\in K^{rs}}\exp(-\theta c_k^{rs})}Q_0^{rs}$$

$$= \frac{\sum_{k \in K_r^{js}} \delta_{gh,k}^{js} \exp\left(-\theta c_k^{js}\right)}{\sum_{k \in K_r^{js}} \exp\left(-\theta c_k^{js}\right)} x_{ij}^{rs}$$

The above formula is equivalent to the STOCH3 algorithm in **(B.5)** with x_{ij}^{rs} corresponding to Q^{rs} and j corresponding to r. Therefore, after having the results of x_{ij}^{rs}, we can compute $x_{ij\to gh}^{rs}$ by running STOCH3 algorithm once from j to s for all links gh by setting all wn_n^{rs} to 0 and wn_j^{rs}:=1 in the forward pass and $xn_s^{rs} := x_{ij}^{rs}$ in the backward pass. Note that the order of reference shortest generalized travel time from origin r could be reused for saving calculational time. The calculation process is presented for each OD pair rs and after having the results of x_{ij}^{rs} as follows:

Step 1: Forward pass

(a) Set all wl_{gh}^{rs} and wn_n^{rs} to 0. Set wn_j^{rs}:=1

(b) For each link gh taken in the order of increasing reference access time from r, if $\Omega_{gh}^r = 1$ then compute $wl_{gh}^{rs} := a_{gh} wn_g^{rs}$ and add wl_{gh}^{rs} to wn_h^{rs}, otherwise do nothing

Step 2: Backward pass

(a) Set $xn_s^{rs} := x_{ij}^{rs}$ for the destination node.

(b) For each link gh taken in the order of decreasing reference access time from r, if $\Omega_{gh}^r = 1$ then compute $xl_{gh}^{rs} := xn_h^{rs} wl_{gh}^{rs} / wn_h^{rs}$ and add xl_{gh}^{rs} to xn_g^{rs}, otherwise do nothing.

Step 3: Calculating $x_{ij\to gh}^{rs}$

$$x_{ij\to gh}^{rs} := xl_{gh}^{rs}$$

Appendix C. Final form of the derivative of a quotient

To agree on how to use the final form of derivative of a quotient in the chapters, we will use the following unified transformation:

Suppose that function C is expressed

$$C = \frac{A}{B} = AB^{-1}$$

where both A and B are the functions of x. The partial derivative of C with respect to x is represented as follows:

$$\frac{\partial C}{\partial x} = \frac{\partial (AB^{-1})}{\partial x} = \frac{\partial (A)}{\partial x} B^{-1} + \frac{\partial (B^{-1})}{\partial x} A = \frac{\partial (A)}{\partial x} B^{-1} + \frac{\partial (B)}{\partial x} AB^{-2} = \\ = \frac{\frac{\partial (A)}{\partial x}}{B} + \frac{C \frac{\partial (B)}{\partial x}}{B}$$

So, the final form of $\frac{\partial C}{\partial x}$ is

$$\frac{\partial C}{\partial x} = \frac{1}{B} \frac{\partial (A)}{\partial x} + \frac{C}{B} \frac{\partial (B)}{\partial x}$$

Appendix D. Details of links and nodes of the Kanazawa road network

Kanazawa road network includes 272 nodes and 964 links. **Table D.1** and **Table D.2** show the intersection names and position coordinates of the nodes. **Tables D.3-D.9** show the links including start and end node numbers, distance, traffic capacity, and free-flow travel time.

Table D.1 Node name and coordinates from node 1 to node 136

No.	Node name	Longitude	Latitude	No.	Node name	Longitude	Latitude
1	Katamachi	136.6521	36.55967	69	Kamiyachi	136.6807	36.59906
2	Korinbo	136.6536	36.56219	70	Izuminodemachiminami	136.6475	36.53482
3	Minamicho	136.6542	36.56793	71	Fushimidai1	136.6391	36.53463
4	Musashi	136.6559	36.57209	72	Hisayasu2	136.6328	36.53447
5	Hashiba	136.6639	36.57098	73	Yokogawaminami	136.6242	36.54148
6	Kenrokuenshita	136.6633	36.56564	74	Otsuka	136.6198	36.54541
7	Hirosaka Kita	136.6598	36.56215	75	Oshino	136.6183	36.54384
8	Gakuinmae	136.6667	36.55876	76	Nishikanazawa	136.6142	36.55231
9	Taimachi	136.6776	36.5608	77	Matsujimaminami	136.6065	36.561
10	Ishibiki 1	136.6736	36.55268	78	Nogyokaikanmaehigashi	136.6068	36.5658
11	Kasamai 3	136.6646	36.55223	79	Wakamiyamachi	136.6327	36.58133
12	Urokomachi	136.6578	36.55588	80	Ekinishimotomachi3	136.6372	36.58541
13	Nagamachi	136.6501	36.5619	81	Sainenmachi	136.6413	36.58823
14	Motoguruma	136.6449	36.56565	82	Kitayasuekita	136.6467	36.58962
15	Sanja	136.6458	36.57023	83	Chuooroshishijonishi	136.6301	36.58674
16	Rokumai	136.6473	36.57388	84	Nishishomae	136.638	36.59212
17	Shirogane	136.6511	36.57366	85	Kubomachi	136.6456	36.52765
18	Ekihigashiguchi	136.6498	36.57749	86	Kubo7chome	136.6383	36.52946
19	Nanatsuyamachi	136.6517	36.58289	87	Minma2chome	136.6336	36.5295
20	Horikawamachikita	136.6527	36.58163	88	Minma3chome	136.6304	36.52951
21	Nakajima Ohashi	136.6554	36.57969	89	Minama1chomenishi	136.6295	36.52705
22	Hikosomachi	136.66	36.57604	90	Yokomiya	136.6169	36.53619
23	Tenjinbashi	136.6687	36.56954	91	Tachinoshogakkomae	136.6157	36.54109
24	Tokiwabashi	136.6734	36.56601	92	Yokaichiminami	136.6097	36.54201
25	Kasamai	136.6666	36.55131	93	Takao1chome	136.641	36.52304
26	Heiwamachi	136.6629	36.54251	94	Takaodai1chome	136.6394	36.52423
27	Teramachi 1	136.6593	36.54615	95	Takaodaichumae	136.6333	36.52369
28	Teramachi 5	136.6526	36.55352	96	Sugahara	136.6146	36.52736
29	Nomachi Hirokoji	136.6489	36.55692	97	Nonoichishogakkomae	136.6101	36.533
30	Shiragikucho	136.6471	36.55776	98	Taiheiji	136.6068	36.5327
31	Nakamuramachi	136.6418	36.56242	99	Horiuchikita	136.6007	36.53229
32	Masuizumi	136.6378	36.55914	100	Mikkaichi	136.5954	36.53231
33	Shinkanda	136.6292	36.56345	101	Futsukaichi	136.6004	36.53932
34	Irie	136.6334	36.56732	102	Okyozuka	136.6002	36.547
35	Mamedabashi	136.6362	36.56981	103	Okyozukakita	136.5994	36.55411
36	Minamihirookamachi	136.639	36.57221	104	Morito1chome	136.5985	36.56096
37	Nakabashi	136.6435	36.57594	105	Matsujimakita	136.6026	36.56914
38	Hirooka	136.6456	36.57923	106	Kobu	136.6066	36.57236
39	Hirooka 1	136.6474	36.58187	107	Shimenonakamachi	136.6171	36.57915
40	Otomarumachi	136.6632	36.58594	108	Fujie	136.624	36.58559
41	Asano Motomachi	136.6634	36.5826	109	Sainen	136.6309	36.59148
42	Moriyamakita	136.6645	36.57822	110	Minamishimbo	136.64	36.59622
43	Higashiyama	136.6662	36.57439	111	Moroe	136.6447	36.59784
44	Yamanoue	136.6704	36.57983	112	Moroemachinakacho	136.6479	36.59867
45	Naruwa	136.6736	36.58464	113	Tanaka	136.6594	36.60173
46	Izumino	136.6488	36.54378	114	Hikidamachi	136.67	36.60452
47	Izumi	136.6442	36.54986	115	Fukuhisa	136.6815	36.61557
48	Izumimotocho	136.6336	36.55136	116	Wakahino	136.5978	36.57741
49	Nagasaka	136.6562	36.53853	117	Matsumuramachi	136.6163	36.58938
50	Izuminodemachi	136.6482	36.53953	118	Unedamachi	136.6105	36.5923
51	Uearimatsu	136.6395	36.54004	119	Chuobyoinkitaguchi	136.6259	36.59634
52	Arimatsu	136.6413	36.54443	120	Monyamachinishiguchi	136.6347	36.60593
53	Hisayasu	136.6331	36.54436	121	Toiyamachi	136.6438	36.60502
54	Yokogawa	136.6278	36.54408	122	Toiyadanchihigashiguchi	136.6492	36.60551
55	Nishikanazawaeki	136.6216	36.55197	123	Takaominami	136.634	36.51787
56	Hokomachi	136.6192	36.5556	124	Takao	136.6269	36.51948
57	Hoko 3	136.6163	36.55854	125	Nukashimbonishi	136.6171	36.52147
58	Irie 3	136.623	36.56614	126	Yahagikita	136.6133	36.5224
59	Tamaboko	136.6277	36.57058	127	Toheida	136.6091	36.52307
60	Wakamiyaohashi	136.6303	36.57312	128	Shimobayashi	136.6042	36.52095
61	Wakamiyajinjamae	136.6326	36.57529	129	Shimobayashinishi	136.6	36.5211
62	Futakuchimachi	136.6367	36.57926	130	Nagatakehigashi	136.5884	36.52108
63	Ekinishimotomachi1	136.6404	36.58264	131	Inuimachi	136.5806	36.52167
64	Sainemachihigashi	136.644	36.58519	132	Nunoichi	136.571	36.52257
65	Kitayasuemachi	136.6482	36.5868	133	Mattokeisatsushomae	136.5706	36.52584
66	Kitayasuehigashi	136.6502	36.58743	134	Higashisanbancho	136.5685	36.5256
67	Takayanagi	136.663	36.59437	135	Miyanaga	136.5731	36.54626
68	Kosakamachi	136.6769	36.59074	136	Midoridanchiguchi	136.582	36.56941

Table D.2 Node name and coordinates from node 137 to node 272

No.	Node name	Longitude	Latitude	No.	Node name	Longitude	Latitude
137	Nishikeisatsushomae	136.5984	36.59864	205	Tekkodanchiguchi	136.5839	36.54647
138	Katsuramachiminami	136.607	36.60539	206	Yokoekogyodanchinokosaten	136.5845	36.54266
139	Kanazawako	136.6182	36.60722	207	Banjo	136.5831	36.53052
140	Chikaokamachi	136.6306	36.6162	208	Yokaichi5	136.6092	36.55009
141	Kitamamachi	136.636	36.62444	209	Sumiyoshinaka	136.6187	36.52661
142	Suzakimachi	136.6405	36.62849	210	Shisogotaiikukanmae	136.651	36.53942
143	Minato2chome	136.6435	36.63191	211	Izumino1	136.6526	36.5443
144	Kigoshikita	136.659	36.62467	212	Teramachi1	136.6593	36.54615
145	Obamachinishi	136.6744	36.62482	213	Fukumasumachinainokosaten	136.5778	36.54301
146	Nukadanimachi	136.6291	36.51138	214	Kamiyasuhara	136.5883	36.56149
147	Nukuchugakkomae	136.6236	36.51034	215	Fukurobatake	136.6065	36.57778
148	Awada	136.6126	36.51138	216	Nodamachi	136.6678	36.53621
149	Awadanishi	136.6089	36.5116	217	Anyojikita	136.5963	36.49935
150	Tomioku	136.6052	36.51066	218	Shinjo	136.6068	36.50173
151	Nakabayashi	136.5993	36.5079	219	Awadahigashi	136.6137	36.51129
152	Suematsu2chome	136.5914	36.50907	220	Nogyotandaiguchi	136.5954	36.50811
153	Miuramachi	136.5827	36.50501	221	Senpukuji	136.5876	36.52698
154	Tokumarumachi	136.5774	36.5181	222	Inari	136.6039	36.53247
155	Shijimahigashi	136.623	36.50216	223	Noshiromachi	136.6027	36.53917
156	Chushijima	136.6199	36.50168	224	Okyozuka	136.6002	36.54701
157	Shinjo	136.6112	36.50253	225	Yokaichi1	136.609	36.54463
158	Kamibayashi	136.5999	36.50025	226	Oshinonishi	136.6119	36.5437
159	Kozu	136.589	36.49449	227	Honmachi	136.6145	36.53202
160	Honmachi2chomeminami	136.6194	36.53132	228	Takaodai2	136.6285	36.52407
161	Nonoichiekikita	136.5968	36.542	229	Minma1	136.6335	36.52696
162	Okyozukanishi	136.5954	36.54677	230	Fushimidaishomae	136.6385	36.52775
163	Miyanagaminami	136.5714	36.54313	231	Fushimishinmachi	136.6499	36.5348
164	Fukumasumachi	136.5807	36.56442	232	Wakakusamachi	136.6599	36.54075
165	Kaihinkoenguchi	136.589	36.58558	233	Izumino3	136.6492	36.54788
166	Saigawabashi	136.5896	36.58716	234	Nishiizumi5	136.6292	36.55174
167	Futatsuderabashi	136.6021	36.58544	235	Morito5	136.5988	36.55852
168	Shimenobashi	136.6091	36.57773	236	Fukumasumachi	136.5777	36.54628
169	Saigawabashi	136.6154	36.57799	237	Kamiyasuharanainokosaten	136.5841	36.56406
170	Wakamiyaohashi	136.6303	36.57312	238	Yasuharakogyodanchinainokosaten	136.5816	36.56826
171	Mamedaohashi	136.6362	36.5698	239	Minamizukamachinainokosaten	136.5949	36.56601
172	Mikageohashi	136.6432	36.5636	240	Kitazuka	136.5956	36.56911
173	Shinbashi	136.648	36.56077	241	Senkoji	136.5872	36.5789
174	Saigawaohashi	136.6504	36.55853	242	Shigarakujifukinnokosaten	136.6108	36.58688
175	Sakurabashi	136.6554	36.55338	243	Sakuradamachi	136.6192	36.57749
176	Shimokikubashi	136.6602	36.54956	244	Izumochokoen	136.6247	36.57612
177	Kamikikubashi	136.6622	36.54808	245	Toitashomae	136.6245	36.57934
178	Okuwabashi	136.6781	36.5385	246	Moroemachicho	136.6451	36.59242
179	Kagatsumeohashi	136.6472	36.62961	247	Moroemachichonishi	136.6424	36.59439
180	Kurafubashi	136.6454	36.62564	248	Minamishimbominami	136.6369	36.59487
181	Kitaderabashi	136.646	36.61134	249	Minamishimbonainokosaten	136.6356	36.59736
182	Matsuderabashi	136.6501	36.60295	250	Kuratsuki	136.6369	36.60053
183	Isobebashi	136.6495	36.59505	251	Fujiebashi	136.619	36.59075
184	Hikosanobashi	136.6613	36.57707	252	Tomizuminami	136.6192	36.60382
185	Asanogawaohashi	136.6645	36.57206	253	Waridashi	136.6444	36.60205
186	Suzumibashi	136.679	36.56202	254	Mitsuyamachi	136.6435	36.6113
187	Ouramachinainokosaten	136.6518	36.61313	255	Matsuderamachinainokosaten	136.6513	36.60953
188	Nogyokyosankaikanmaenokosaten	136.6594	36.60813	256	Omiyabashi	136.6485	36.62503
189	Jingujikita	136.669	36.58879	257	Kigoshinishi	136.6593	36.61927
190	Ohimachi	136.6751	36.5869	258	Fukuhisamachi	136.6742	36.61775
191	Jinguji	136.6684	36.58365	259	Hikidahigashi	136.6746	36.60163
192	Musashinishi	136.6548	36.57339	260	Matsuderamachi	136.6594	36.60475
193	Shimotsutsumicho	136.655	36.57093	261	Isobemachi	136.6567	36.5953
194	Fujitabyoin	136.6571	36.56932	262	Otomarurikkyokitazume	136.6634	36.59164
195	Shokokaigishomae	136.6567	36.56751	263	Bakuromachi	136.6587	36.57155
196	Motogikucho	136.6417	36.57436	264	Yokoyamacho	136.6707	36.56339
197	Sainenmachinishi	136.64	36.58744	265	Okuwashinmachi	136.6789	36.53906
198	Sainennaka	136.6333	36.58913	266	Kitadera	136.646	36.61134
199	Minamishinbonaka	136.6347	36.60101	267	Goboichimachi	136.5773	36.52887
200	Nishiko	136.6187	36.59568	268	Akatsuchimachi	136.5935	36.5829
201	Izumomachi	136.6246	36.57574	269	Yoneizumi3	136.6219	36.54464
202	Kamino	136.5947	36.56523	270	Hashibacho	136.6639	36.57098
203	Utsigimachinainokousaten	136.5669	36.56833	271	Otemachi	136.6643	36.56808
204	Kamiyasuharaminami	136.5928	36.55863	272	Kitamachi	136.6287	36.58325

Table D.3 Link characteristics from link 1 to link 140

No.	Start node	End node	Length (m)	Traffic capacity (pcu)	Free-flow travel time (Min)	No.	Start node	End node	Length (m)	Traffic capacity (pcu)	Free-flow travel time (Min)
1	1	2	330	813	0.495	71	264	24	420	100	0.84
2	2	1	330	813	0.495	72	24	264	420	100	0.84
3	2	3	640	813	0.96	73	24	23	750	100	1.5
4	3	2	640	813	0.96	74	23	24	750	100	1.5
5	3	193	210	813	0.315	75	23	271	410	350	0.82
6	193	3	210	813	0.315	76	271	23	410	350	0.82
7	193	4	300	813	0.45	77	271	5	350	350	0.525
8	4	193	300	813	0.45	78	5	271	350	350	0.525
9	4	192	160	1460	0.24	79	23	270	460	100	0.92
10	192	4	160	1460	0.24	80	270	23	460	100	0.92
11	2	192	1550	250	4.65	81	5	263	450	566	0.675
12	192	2	1550	250	4.65	82	263	5	450	566	0.675
13	192	17	330	1460	0.495	83	5	270	100	903	0.15
14	17	192	330	1460	0.495	84	270	5	100	903	0.15
15	17	16	330	2156	0.495	85	270	185	120	903	0.18
16	16	17	330	2156	0.495	86	185	270	120	903	0.18
17	16	15	440	1623	0.66	87	185	43	290	903	0.435
18	15	16	440	1623	0.66	88	43	185	290	903	0.435
19	15	14	530	1623	0.795	89	43	44	700	903	1.05
20	14	15	530	1623	0.795	90	44	43	700	903	1.05
21	14	13	640	1000	0.96	91	44	45	620	903	0.93
22	13	14	640	1000	0.96	92	45	44	620	903	0.93
23	13	1	290	1000	0.435	93	45	190	280	903	0.42
24	1	13	290	1000	0.435	94	190	45	280	903	0.42
25	1	12	700	1000	1.05	95	190	68	450	903	0.675
26	12	1	700	1000	1.05	96	68	190	450	903	0.675
27	12	11	680	1000	1.02	97	68	69	910	580	1.365
28	11	12	680	1000	1.02	98	69	68	910	580	1.365
29	11	25	240	438	0.36	99	69	259	660	1000	0.792
30	25	11	240	438	0.36	100	259	69	660	1000	0.792
31	25	178	2100	438	4.2	101	259	114	660	1000	0.792
32	178	25	2100	438	4.2	102	114	259	660	1000	0.792
33	10	25	670	100	1.34	103	114	115	1880	2264	1.88
34	25	10	670	100	1.34	104	115	114	1880	2264	1.88
35	10	11	880	438	1.32	105	114	258	1540	472	2.31
36	11	10	880	438	1.32	106	258	114	1540	472	2.31
37	10	8	920	820	1.38	107	258	145	800	350	1.6
38	8	10	920	820	1.38	108	145	258	800	350	1.6
39	8	7	880	350	1.32	109	42	44	580	500	0.87
40	7	8	880	350	1.32	110	44	42	580	500	0.87
41	7	12	720	800	1.08	111	41	191	460	893	0.69
42	12	7	720	800	1.08	112	191	41	460	893	0.69
43	7	2	520	820	0.78	113	191	45	470	893	0.705
44	2	7	520	820	0.78	114	45	191	470	893	0.705
45	7	195	800	350	1.6	115	191	189	570	350	1.14
46	195	7	800	350	1.6	116	189	191	570	350	1.14
47	195	3	230	100	0.46	117	40	189	590	350	0.885
48	3	195	230	100	0.46	118	189	40	590	350	0.885
49	195	194	210	350	0.42	119	189	190	690	350	1.035
50	194	195	210	350	0.42	120	190	189	690	350	1.035
51	193	194	440	100	0.88	121	189	68	810	757	1.215
52	194	193	440	100	0.88	122	68	189	810	757	1.215
53	4	263	270	566	0.405	123	67	259	1500	300	3
54	263	4	270	566	0.405	124	259	67	1500	300	3
55	194	263	280	300	0.56	125	113	114	1030	2179	1.03
56	263	194	280	300	0.56	126	114	113	1030	2179	1.03
57	271	194	520	350	1.04	127	257	258	1915	500	2.8725
58	194	271	520	350	1.04	128	258	257	1915	500	2.8725
59	6	271	350	350	0.525	129	144	145	1709	100	5.127
60	271	6	350	350	0.525	130	145	144	1709	100	5.127
61	7	6	510	820	0.765	131	43	42	470	1298	0.705
62	6	7	510	820	0.765	132	42	43	470	1298	0.705
63	8	6	910	820	1.365	133	41	42	500	1298	0.75
64	6	8	910	820	1.365	134	42	41	500	1298	0.75
65	264	6	690	500	1.035	135	41	40	370	1298	0.555
66	6	264	690	500	1.035	136	40	41	370	1298	0.555
67	9	264	660	500	0.99	137	40	262	640	1298	0.96
68	264	9	660	500	0.99	138	262	40	640	1298	0.96
69	10	9	1500	501	3	139	262	67	300	1298	0.36
70	9	10	1500	501	3	140	67	262	300	1298	0.36

Table D.4 Link characteristics from link 141 to link 280

No.	Start node	End node	Length (m)	Traffic capacity (pcu)	Free-flow travel time (Min)	No.	Start node	End node	Length (m)	Traffic capacity (pcu)	Free-flow travel time (Min)
141	67	113	960	1298	1.152	211	180	179	710	100	2.13
142	113	67	960	1298	1.152	212	179	180	710	100	2.13
143	113	188	680	1298	0.816	213	4	22	550	1320	0.825
144	188	113	680	1298	0.816	214	22	4	550	1320	0.825
145	188	257	1240	1298	1.488	215	192	184	420	250	0.84
146	257	188	1240	1298	1.488	216	184	192	420	250	0.84
147	257	144	600	1298	0.72	217	17	18	460	1511	0.69
148	144	257	600	1298	0.72	218	18	17	460	1511	0.69
149	9	186	270	501	0.405	219	16	18	440	1623	0.66
150	186	9	270	501	0.405	220	18	16	440	1623	0.66
151	22	42	460	500	0.69	221	18	21	300	893	0.45
152	42	22	460	500	0.69	222	21	18	300	893	0.45
153	21	41	300	893	0.45	223	18	20	550	441	0.825
154	41	21	300	893	0.45	224	20	18	550	441	0.825
155	20	40	1050	350	1.575	225	18	19	862	500	1.293
156	40	20	1050	350	1.575	226	19	18	862	500	1.293
157	19	262	1760	100	5.28	227	39	19	630	300	1.26
158	262	19	1760	100	5.28	228	19	39	630	300	1.26
159	66	67	1380	500	1.656	229	65	66	170	500	0.255
160	67	66	1380	500	1.656	230	66	65	170	500	0.255
161	183	261	450	500	0.54	231	111	112	400	2179	0.4
162	261	183	450	500	0.54	232	112	111	400	2179	0.4
163	261	67	620	500	0.744	233	253	182	870	100	2.61
164	67	261	620	500	0.744	234	182	253	870	100	2.61
165	261	260	530	500	0.795	235	121	122	600	350	1.2
166	260	261	530	500	0.795	236	122	121	600	350	1.2
167	112	260	550	2179	0.55	237	254	266	280	400	0.56
168	260	112	550	2179	0.55	238	266	254	280	400	0.56
169	260	113	450	2179	0.45	239	182	122	344	100	1.032
170	113	260	450	2179	0.45	240	122	182	344	100	1.032
171	260	182	680	100	2.04	241	122	266	760	100	2.28
172	182	260	680	100	2.04	242	266	122	760	100	2.28
173	255	188	870	100	2.61	243	266	181	64	100	0.192
174	188	255	870	100	2.61	244	181	266	64	100	0.192
175	256	144	1020	100	3.06	245	181	187	587	100	1.761
176	144	256	1020	100	3.06	246	187	181	587	100	1.761
177	179	144	478	500	0.717	247	37	38	420	1714	0.504
178	144	179	478	500	0.717	248	38	37	420	1714	0.504
179	270	22	570	100	1.71	249	38	39	350	800	0.42
180	22	270	570	100	1.71	250	39	38	350	800	0.42
181	22	184	370	100	1.11	251	39	65	530	800	0.636
182	184	22	370	100	1.11	252	65	39	530	800	0.636
183	184	21	960	100	1.92	253	65	82	340	800	0.408
184	21	184	960	100	1.92	254	82	65	340	800	0.408
185	21	20	300	250	0.9	255	82	246	360	800	0.432
186	20	21	300	250	0.9	256	246	82	360	800	0.432
187	20	19	400	441	0.6	257	246	111	590	800	0.885
188	19	20	400	441	0.6	258	111	246	590	800	0.885
189	19	66	300	441	0.45	259	111	253	440	500	0.66
190	66	19	300	441	0.45	260	253	111	440	500	0.66
191	66	183	900	441	1.35	261	253	121	340	500	0.51
192	183	66	900	441	1.35	262	121	253	340	500	0.51
193	183	112	450	441	0.675	263	121	254	700	500	1.05
194	112	183	450	441	0.675	264	254	121	700	500	1.05
195	112	182	540	100	1.62	265	254	141	1690	500	2.535
196	182	112	540	100	1.62	266	141	254	1690	500	2.535
197	182	255	550	100	1.65	267	110	111	450	2179	0.45
198	255	182	550	100	1.65	268	111	110	450	2179	0.45
199	255	187	635	100	1.905	269	120	121	1041	350	2.082
200	187	255	635	100	1.905	270	121	120	1041	350	2.082
201	187	256	1575	100	4.725	271	246	247	310	495	0.372
202	256	187	1575	100	4.725	272	247	246	310	495	0.372
203	256	179	370	100	1.11	273	247	110	300	495	0.36
204	179	256	370	100	1.11	274	110	247	300	495	0.36
205	179	143	1470	100	4.41	275	110	250	570	542	0.855
206	143	179	1470	100	4.41	276	250	110	570	542	0.855
207	256	180	550	100	1.65	277	250	120	650	542	0.975
208	180	256	550	100	1.65	278	120	250	650	542	0.975
209	180	142	1210	100	3.63	279	120	140	1235	542	1.8525
210	142	180	1210	100	3.63	280	140	120	1235	542	1.8525

Table D.5 Link characteristics from link 281 to link 420

No.	Start node	End node	Length (m)	Traffic capacity (pcu)	Free-flow travel time (Min)	No.	Start node	End node	Length (m)	Traffic capacity (pcu)	Free-flow travel time (Min)
281	39	64	480	500	0.72	351	16	37	420	2156	0.63
282	64	39	480	500	0.72	352	37	16	420	2156	0.63
283	63	64	400	1000	0.48	353	37	62	690	2156	1.035
284	64	63	400	1000	0.48	354	62	37	690	2156	1.035
285	64	65	420	1000	0.504	355	62	79	500	2156	0.75
286	65	64	420	1000	0.504	356	79	62	500	2156	0.75
287	64	81	400	500	0.6	357	79	108	880	1367	1.32
288	81	64	400	500	0.6	358	108	79	880	1367	1.32
289	80	197	340	1000	0.408	359	108	117	947	1367	1.1364
290	197	80	340	1000	0.408	360	117	108	947	1367	1.1364
291	197	81	140	1000	0.168	361	117	118	722	1367	0.8664
292	81	197	140	1000	0.168	362	118	117	722	1367	0.8664
293	81	82	530	1000	0.636	363	118	137	1520	1367	1.824
294	82	81	530	1000	0.636	364	137	118	1520	1367	1.824
295	197	84	570	350	1.14	365	118	138	1524	455	2.286
296	84	197	570	350	1.14	366	138	118	1524	455	2.286
297	198	84	530	500	0.795	367	242	118	600	100	1.8
298	84	198	530	500	0.795	368	118	242	600	100	1.8
299	84	247	460	500	0.92	369	167	137	1610	400	3.22
300	247	84	460	500	0.92	370	137	167	1610	400	3.22
301	84	248	310	350	0.465	371	117	251	765	500	1.1475
302	248	84	310	350	0.465	372	251	117	765	500	1.1475
303	248	249	177	350	0.354	373	242	117	810	100	2.43
304	249	248	177	350	0.354	374	117	242	810	100	2.43
305	249	199	555	100	1.665	375	167	242	1310	100	3.93
306	199	249	555	100	1.665	376	242	167	1310	100	3.93
307	248	110	330	2119	0.33	377	268	167	1520	100	4.56
308	110	248	330	2119	0.33	378	167	268	1520	100	4.56
309	199	250	255	400	0.51	379	165	268	1520	800	2.28
310	250	199	255	400	0.51	380	268	165	1520	800	2.28
311	252	199	1750	400	3.5	381	196	37	220	500	0.264
312	199	252	1750	400	3.5	382	37	196	220	500	0.264
313	119	249	1130	100	3.39	383	245	79	530	500	1.06
314	249	119	1130	100	3.39	384	79	245	530	500	1.06
315	109	248	660	2119	0.66	385	107	108	980	2249	0.98
316	248	109	660	2119	0.66	386	108	107	980	2249	0.98
317	38	63	610	1714	0.732	387	168	117	1940	400	3.88
318	63	38	610	1714	0.732	388	117	168	1940	400	3.88
319	63	80	570	1714	0.684	389	169	107	670	2249	0.67
320	80	63	570	1714	0.684	390	107	169	670	2249	0.67
321	80	198	540	1714	0.648	391	243	107	250	547	0.375
322	198	80	540	1714	0.648	392	107	243	250	547	0.375
323	198	109	330	1714	0.396	393	196	16	520	547	0.78
324	109	198	330	1714	0.396	394	16	196	520	547	0.78
325	109	119	632	1213	0.7584	395	245	196	970	547	1.94
326	119	109	632	1213	0.7584	396	196	245	970	547	1.94
327	200	119	1060	500	1.59	397	201	245	300	547	0.45
328	119	200	1060	500	1.59	398	245	201	300	547	0.45
329	62	63	500	1000	0.6	399	243	201	500	500	0.75
330	63	62	500	1000	0.6	400	201	243	500	500	0.75
331	79	80	650	1000	0.78	401	169	243	800	500	1.2
332	80	79	650	1000	0.78	402	243	169	800	500	1.2
333	79	83	720	350	0.864	403	168	169	385	500	0.5775
334	83	79	720	350	0.864	404	169	168	385	500	0.5775
335	83	198	390	500	0.468	405	215	168	280	400	0.56
336	198	83	390	500	0.468	406	168	215	280	400	0.56
337	83	272	430	500	0.645	407	268	215	1570	100	4.71
338	272	83	430	500	0.645	408	215	268	1570	100	4.71
339	108	272	390	2119	0.39	409	36	196	370	500	0.444
340	272	108	390	2119	0.39	410	196	36	370	500	0.444
341	272	109	510	2119	0.51	411	36	61	690	350	1.035
342	109	272	510	2119	0.51	412	61	36	690	350	1.035
343	108	251	750	500	1.125	413	61	245	300	1000	0.6
344	251	108	750	500	1.125	414	245	61	300	1000	0.6
345	251	200	535	500	0.8025	415	35	36	340	500	0.408
346	200	251	535	500	0.8025	416	36	35	340	500	0.408
347	200	252	1100	500	1.65	417	60	61	300	1000	0.36
348	252	200	1100	500	1.65	418	61	60	300	1000	0.36
349	252	139	390	500	0.585	419	244	201	360	547	0.54
350	139	252	390	500	0.585	420	201	244	360	547	0.54

Table D.6 Link characteristics from link 421 to link 560

No.	Start node	End node	Length (m)	Traffic capacity (pcu)	Free-flow travel time (Min)	No.	Start node	End node	Length (m)	Traffic capacity (pcu)	Free-flow travel time (Min)
421	14	35	980	500	1.176	491	204	202	740	400	1.48
422	35	14	980	500	1.176	492	202	204	740	400	1.48
423	35	60	620	1000	0.744	493	162	204	1350	500	2.025
424	60	35	620	1000	0.744	494	204	162	1350	500	2.025
425	60	244	570	1000	0.684	495	214	204	929	400	1.858
426	244	60	570	1000	0.684	496	204	214	929	400	1.858
427	244	243	570	547	0.684	497	204	235	375	100	1.125
428	243	244	570	547	0.684	498	235	204	375	100	1.125
429	15	36	720	350	1.08	499	205	237	2126	100	6.378
430	36	15	720	350	1.08	500	237	205	2126	100	6.378
431	116	215	1000	400	2	501	170	60	200	500	0.24
432	215	116	1000	400	2	502	60	170	200	500	0.24
433	116	268	1070	100	3.21	503	59	170	150	500	0.18
434	268	116	1070	100	3.21	504	170	59	150	500	0.18
435	116	165	1224	954	1.4688	505	58	59	640	1000	0.768
436	165	116	1224	954	1.4688	506	59	58	640	1000	0.768
437	241	116	1260	100	3.78	507	57	58	1020	1000	1.224
438	116	241	1260	100	3.78	508	58	57	1020	1000	1.224
439	106	215	625	500	0.9375	509	76	57	800	446	0.96
440	215	106	625	500	0.9375	510	57	76	800	446	0.96
441	105	116	1040	954	1.248	511	171	35	230	500	0.276
442	116	105	1040	954	1.248	512	35	171	230	500	0.276
443	106	169	400	2249	0.4	513	34	171	120	500	0.144
444	169	106	400	2249	0.4	514	171	34	120	500	0.144
445	105	106	758	2249	0.758	515	33	34	580	1000	0.696
446	106	105	758	2249	0.758	516	34	33	580	1000	0.696
447	240	105	1680	400	3.36	517	56	33	1260	1000	1.512
448	105	240	1680	400	3.36	518	33	56	1260	1000	1.512
449	136	240	680	400	1.36	519	76	56	570	1000	0.684
450	240	136	680	400	1.36	520	56	76	570	1000	0.684
451	239	240	500	100	1.5	521	34	59	660	500	0.99
452	240	239	500	100	1.5	522	59	34	660	500	0.99
453	202	105	610	100	1.83	523	33	58	640	954	0.768
454	105	202	610	100	1.83	524	58	33	640	954	0.768
455	239	202	1100	100	3.3	525	56	57	420	350	0.63
456	202	239	1100	100	3.3	526	57	56	420	350	0.63
457	214	239	280	100	0.84	527	102	76	1758	404	2.637
458	239	214	280	100	0.84	528	76	102	1758	404	2.637
459	237	214	470	400	0.94	529	162	102	561	576	0.8415
460	214	237	470	400	0.94	530	102	162	561	576	0.8415
461	164	237	485	400	0.97	531	205	162	1231	400	2.462
462	237	164	485	400	0.97	532	162	205	1231	400	2.462
463	106	59	1850	500	2.775	533	236	205	750	400	1.5
464	59	106	1850	500	2.775	534	205	236	750	400	1.5
465	78	58	1300	954	1.56	535	135	236	280	400	0.56
466	58	78	1300	954	1.56	536	236	135	280	400	0.56
467	78	57	1220	400	1.83	537	31	34	1020	500	1.53
468	57	78	1220	400	1.83	538	34	31	1020	500	1.53
469	78	106	730	260	1.095	539	32	33	940	954	1.41
470	106	78	730	260	1.095	540	33	32	940	954	1.41
471	76	78	1846	400	2.769	541	234	33	1700	600	3.4
472	78	76	1846	400	2.769	542	33	234	1700	600	3.4
473	78	105	590	954	0.708	543	76	55	570	446	0.855
474	105	78	590	954	0.708	544	55	76	570	446	0.855
475	77	78	697	800	1.0455	545	55	234	600	423	0.9
476	78	77	697	800	1.0455	546	234	55	600	423	0.9
477	76	77	1512	1000	1.8144	547	234	48	690	423	1.38
478	77	76	1512	1000	1.8144	548	48	234	690	423	1.38
479	104	77	720	500	1.08	549	48	47	990	423	1.98
480	77	104	720	500	1.08	550	47	48	990	423	1.98
481	103	76	1384	446	2.076	551	47	233	820	250	1.64
482	76	103	1384	446	2.076	552	233	47	820	250	1.64
483	104	105	1080	1826	1.08	553	172	14	170	1623	0.255
484	105	104	1080	1826	1.08	554	14	172	170	1623	0.255
485	235	104	270	1826	0.27	555	31	172	330	1623	0.66
486	104	235	270	1826	0.27	556	172	31	330	1623	0.66
487	103	235	490	1826	0.49	557	32	31	520	1623	1.04
488	235	103	490	1826	0.49	558	31	32	520	1623	1.04
489	102	103	829	1826	0.829	559	48	32	990	979	1.485
490	103	102	829	1826	0.829	560	32	48	990	979	1.485

Table D.7 Link characteristics from link 561 to link 700

No.	Start node	End node	Length (m)	Traffic capacity (pcu)	Free-flow travel time (Min)	No.	Start node	End node	Length (m)	Traffic capacity (pcu)	Free-flow travel time (Min)
561	54	48	980	979	1.47	631	225	208	680	100	2.04
562	48	54	980	979	1.47	632	208	225	680	100	2.04
563	73	54	470	1118	0.705	633	92	225	560	400	1.12
564	54	73	470	1118	0.705	634	225	92	560	400	1.12
565	90	73	1005	1118	1.206	635	90	92	1030	405	1.545
566	73	90	1005	1118	1.206	636	92	90	1030	405	1.545
567	97	90	850	816	1.02	637	160	90	620	405	0.93
568	90	97	850	816	1.02	638	90	160	620	405	0.93
569	98	97	272	816	0.3264	639	209	160	592	405	0.888
570	97	98	272	816	0.3264	640	160	209	592	405	0.888
571	222	98	320	816	0.384	641	125	209	696	405	1.044
572	98	222	320	816	0.384	642	209	125	696	405	1.044
573	99	222	360	816	0.432	643	219	125	1000	405	1.5
574	222	99	360	816	0.432	644	125	219	1000	405	1.5
575	100	99	424	816	0.5088	645	161	162	552	500	0.828
576	99	100	424	816	0.5088	646	162	161	552	500	0.828
577	207	100	860	550	1.29	647	224	102	460	1000	0.552
578	100	207	860	550	1.29	648	102	224	460	1000	0.552
579	267	207	1145	550	1.7175	649	101	224	400	1000	0.48
580	207	267	1145	550	1.7175	650	224	101	400	1000	0.48
581	133	267	820	550	1.23	651	100	101	981	1826	0.981
582	267	133	820	550	1.23	652	101	100	981	1826	0.981
583	134	133	368	550	0.552	653	221	100	1070	1000	1.284
584	133	134	368	550	0.552	654	100	221	1070	1000	1.284
585	269	55	720	600	1.44	655	131	221	970	1000	1.164
586	55	269	720	600	1.44	656	221	131	970	1000	1.164
587	75	76	2246	400	4.492	657	154	131	840	1800	0.84
588	76	75	2246	400	4.492	658	131	154	840	1800	0.84
589	74	75	312	438	0.468	659	206	205	771	100	2.313
590	75	74	312	438	0.468	660	205	206	771	100	2.313
591	269	74	300	438	0.45	661	207	206	1240	100	3.72
592	74	269	300	438	0.45	662	206	207	1240	100	3.72
593	54	269	750	438	1.5	663	221	207	480	100	1.44
594	269	54	750	438	1.5	664	207	221	480	100	1.44
595	73	269	344	350	0.688	665	130	221	1230	100	3.69
596	269	73	344	350	0.688	666	221	130	1230	100	3.69
597	91	73	800	400	1.6	667	152	130	1480	350	2.96
598	73	91	800	400	1.6	668	130	152	1480	350	2.96
599	90	91	576	404	0.864	669	217	152	1240	800	1.86
600	91	90	576	404	0.864	670	152	217	1240	800	1.86
601	91	74	608	404	0.912	671	220	130	1750	800	2.625
602	74	91	608	404	0.912	672	130	220	1750	800	2.625
603	91	75	780	400	1.56	673	213	236	370	100	1.11
604	75	91	780	400	1.56	674	236	213	370	100	1.11
605	91	226	590	500	0.885	675	267	213	1570	100	4.71
606	226	91	590	500	0.885	676	213	267	1570	100	4.71
607	225	226	330	100	0.99	677	131	267	1000	100	3
608	226	225	330	100	0.99	678	267	131	1000	100	3
609	224	225	940	400	1.88	679	153	131	1464	350	2.928
610	225	224	940	400	1.88	680	131	153	1464	350	2.928
611	161	224	500	400	1	681	222	223	775	400	1.55
612	224	161	500	400	1	682	223	222	775	400	1.55
613	206	161	1103	400	2.206	683	129	99	984	500	1.476
614	161	206	1103	400	2.206	684	99	129	984	500	1.476
615	213	206	1070	400	2.14	685	151	129	1456	350	2.912
616	206	213	1070	400	2.14	686	129	151	1456	350	2.912
617	163	213	1170	400	2.34	687	158	151	880	350	1.76
618	213	163	1170	400	2.34	688	151	158	880	350	1.76
619	226	75	730	800	1.095	689	128	98	1056	100	3.168
620	75	226	730	800	1.095	690	98	128	1056	100	3.168
621	92	226	310	500	0.465	691	150	128	1128	350	2.256
622	226	92	310	500	0.465	692	128	150	1128	350	2.256
623	223	92	870	800	1.305	693	127	97	1112	350	2.224
624	92	223	870	800	1.305	694	97	127	1112	350	2.224
625	101	223	240	800	0.36	695	149	127	640	350	1.28
626	223	101	240	800	0.36	696	127	149	640	350	1.28
627	101	161	536	500	0.804	697	218	149	1140	500	1.71
628	161	101	536	500	0.804	698	149	218	1140	500	1.71
629	208	76	920	100	2.76	699	227	90	580	400	1.16
630	76	208	920	100	2.76	700	90	227	580	400	1.16

Table D.8 Link characteristics from link 701 to link 840

No.	Start node	End node	Length (m)	Traffic capacity (pcu)	Free-flow travel time (Min)	No.	Start node	End node	Length (m)	Traffic capacity (pcu)	Free-flow travel time (Min)
701	96	227	460	400	0.92	771	97	227	560	100	1.68
702	227	96	460	400	0.92	772	227	97	560	100	1.68
703	126	96	592	449	0.888	773	227	160	540	100	1.62
704	96	126	592	449	0.888	774	160	227	540	100	1.62
705	148	126	940	449	1.41	775	160	88	1251	405	1.8765
706	126	148	940	449	1.41	776	88	160	1251	405	1.8765
707	157	148	1297	456	1.9455	777	88	87	288	405	0.432
708	148	157	1297	456	1.9455	778	87	88	288	405	0.432
709	159	217	990	800	1.485	779	87	86	408	405	0.612
710	217	159	990	800	1.485	780	86	87	408	405	0.612
711	217	158	410	800	0.615	781	86	85	845	405	1.2675
712	158	217	410	800	0.615	782	85	86	845	405	1.2675
713	158	218	350	800	0.525	783	72	73	1650	500	1.98
714	218	158	350	800	0.525	784	73	72	1650	500	1.98
715	218	157	350	800	0.525	785	71	72	550	500	0.66
716	157	218	350	800	0.525	786	72	71	550	500	0.66
717	157	156	960	386	1.92	787	70	71	750	500	1.125
718	156	157	960	386	1.92	788	71	70	750	500	1.125
719	156	155	290	386	0.58	789	231	70	220	350	0.33
720	155	156	290	386	0.58	790	70	231	220	350	0.33
721	153	152	1008	540	1.512	791	49	231	750	350	1.125
722	152	153	1008	540	1.512	792	231	49	750	350	1.125
723	152	220	470	400	0.94	793	232	49	420	350	0.63
724	220	152	470	400	0.94	794	49	232	420	350	0.63
725	220	151	440	400	0.88	795	26	232	330	350	0.495
726	151	220	440	400	0.88	796	232	26	330	350	0.495
727	151	150	616	540	0.924	797	53	48	830	300	1.66
728	150	151	616	540	0.924	798	48	53	830	300	1.66
729	150	149	336	540	0.504	799	72	53	1120	500	1.68
730	149	150	336	540	0.504	800	53	72	1120	500	1.68
731	149	148	320	540	0.48	801	87	72	544	100	1.632
732	148	149	320	540	0.48	802	72	87	544	100	1.632
733	148	219	420	400	0.84	803	229	87	290	400	0.58
734	219	148	420	400	0.84	804	87	229	290	400	0.58
735	219	147	810	400	1.62	805	95	229	360	400	0.72
736	147	219	810	400	1.62	806	229	95	360	400	0.72
737	146	147	540	540	0.81	807	123	95	990	400	1.98
738	147	146	540	540	0.81	808	95	123	990	400	1.98
739	132	131	912	403	1.368	809	173	13	90	100	0.18
740	131	132	912	403	1.368	810	13	173	90	100	0.18
741	131	130	736	403	1.104	811	30	173	510	100	1.02
742	130	131	736	403	1.104	812	173	30	510	100	1.02
743	130	129	1000	403	1.5	813	174	1	260	813	0.39
744	129	130	1000	403	1.5	814	1	174	260	813	0.39
745	129	128	384	403	0.576	815	29	174	180	813	0.27
746	128	129	384	403	0.576	816	174	29	180	813	0.27
747	128	127	488	403	0.732	817	47	29	880	768	1.32
748	127	128	488	403	0.732	818	29	47	880	768	1.32
749	127	126	680	403	1.02	819	52	47	670	768	1.005
750	126	127	680	403	1.02	820	47	52	670	768	1.005
751	126	125	446	405	0.669	821	51	52	530	1176	0.636
752	125	126	446	405	0.669	822	52	51	530	1176	0.636
753	125	124	904	405	1.356	823	72	51	860	500	1.032
754	124	125	904	405	1.356	824	51	72	860	500	1.032
755	124	123	812	405	1.218	825	88	72	617	386	1.234
756	123	124	812	405	1.218	826	72	88	617	386	1.234
757	228	95	530	800	0.795	827	89	88	296	386	0.592
758	95	228	530	800	0.795	828	88	89	296	386	0.592
759	95	94	536	800	0.804	829	228	89	350	500	0.525
760	94	95	536	800	0.804	830	89	228	350	500	0.525
761	94	93	224	1176	0.2688	831	124	228	540	500	0.81
762	93	94	224	1176	0.2688	832	228	124	540	500	0.81
763	96	209	413	100	1.239	833	147	124	824	386	1.648
764	209	96	413	100	1.239	834	124	147	824	386	1.648
765	209	89	120	100	0.36	835	156	147	1307	386	2.614
766	89	209	120	100	0.36	836	147	156	1307	386	2.614
767	89	229	440	100	1.32	837	71	51	620	1176	0.744
768	229	89	440	100	1.32	838	51	71	620	1176	0.744
769	229	230	570	100	1.71	839	86	71	584	1176	0.7008
770	230	229	570	100	1.71	840	71	86	584	1176	0.7008

Table D.9 Link characteristics from link 841 to link 964

No.	Start node	End node	Length (m)	Traffic capacity (pcu)	Free-flow travel time (Min)	No.	Start node	End node	Length (m)	Traffic capacity (pcu)	Free-flow travel time (Min)
841	230	86	200	800	0.3	903	28	212	450	581	0.9
842	86	230	200	800	0.3	904	212	28	450	581	0.9
843	94	230	410	800	0.615	905	212	27	520	581	1.04
844	230	94	410	800	0.615	906	27	212	520	581	1.04
845	175	12	400	350	0.6	907	232	27	660	350	1.32
846	12	175	400	350	0.6	908	27	232	660	350	1.32
847	28	175	230	300	0.345	909	27	26	530	581	1.06
848	175	28	230	300	0.345	910	26	27	530	581	1.06
849	233	28	340	581	0.51	911	26	216	860	581	1.72
850	28	233	340	581	0.51	912	216	26	860	581	1.72
851	46	233	830	581	1.245	913	26	265	1630	400	3.26
852	233	46	830	581	1.245	914	265	26	1630	400	3.26
853	50	46	470	581	0.705	915	265	178	610	100	1.22
854	46	50	470	581	0.705	916	178	265	610	100	1.22
855	70	50	530	581	0.795	917	159	153	1784	553	2.676
856	50	70	530	581	0.795	918	153	159	1784	553	2.676
857	85	70	848	581	1.272	919	153	154	872	553	1.308
858	70	85	848	581	1.272	920	154	153	872	553	1.308
859	93	85	1562	581	2.343	921	154	132	848	553	1.272
860	85	93	1562	581	2.343	922	132	154	848	553	1.272
861	123	93	980	386	1.96	923	132	133	344	553	0.516
862	93	123	980	386	1.96	924	133	132	344	553	0.516
863	146	123	910	386	1.82	925	133	163	1904	473	2.856
864	123	146	910	386	1.82	926	163	133	1904	473	2.856
865	155	146	860	386	1.72	927	163	135	392	473	0.588
866	146	155	860	386	1.72	928	135	163	392	473	0.588
867	51	50	800	405	1.2	929	135	164	1705	625	2.5575
868	50	51	800	405	1.2	930	164	135	1705	625	2.5575
869	50	210	290	405	0.435	931	164	238	530	625	0.795
870	210	50	290	405	0.435	932	238	164	530	625	0.795
871	210	49	480	350	0.72	933	238	136	400	625	0.6
872	49	210	480	350	0.72	934	136	238	400	625	0.6
873	54	53	480	768	0.72	935	136	241	1340	625	2.01
874	53	54	480	768	0.72	936	241	136	1340	625	2.01
875	53	52	750	768	1.125	937	241	165	610	625	0.915
876	52	53	750	768	1.125	938	165	241	610	625	0.915
877	52	46	690	438	1.035	939	135	203	2800	400	5.6
878	46	52	690	438	1.035	940	203	135	2800	400	5.6
879	46	211	340	438	0.51	941	203	238	1870	400	3.74
880	211	46	340	438	0.51	942	238	203	1870	400	3.74
881	211	27	660	438	0.99	943	203	165	5328	400	10.656
882	27	211	660	438	0.99	944	165	203	5328	400	10.656
883	27	177	310	100	0.62	945	165	166	841	625	1.2615
884	177	27	310	100	0.62	946	166	165	841	625	1.2615
885	177	25	630	100	1.26	947	166	137	1074	625	1.611
886	25	177	630	100	1.26	948	137	166	1074	625	1.611
887	176	11	560	438	0.84	949	137	138	1220	1429	1.464
888	11	176	560	438	0.84	950	138	137	1220	1429	1.464
889	212	176	550	100	1.65	951	138	139	1335	1429	1.602
890	176	212	550	100	1.65	952	139	138	1335	1429	1.602
891	211	212	680	100	2.04	953	139	140	1784	1111	2.1408
892	212	211	680	100	2.04	954	140	139	1784	1111	2.1408
893	210	211	520	100	1.04	955	140	141	1139	1111	1.3668
894	211	210	520	100	1.04	956	141	140	1139	1111	1.3668
895	231	210	500	400	1	957	141	142	680	1111	0.816
896	210	231	500	400	1	958	142	141	680	1111	0.816
897	32	30	850	954	1.275	959	142	143	480	1111	0.576
898	30	32	850	954	1.275	960	143	142	480	1111	0.576
899	30	29	190	954	0.285	961	27	176	330	438	0.495
900	29	30	190	954	0.285	962	176	27	330	438	0.495
901	29	28	500	581	1	963	216	265	650	100	1.3
902	28	29	500	581	1	964	265	216	650	100	1.3

Appendix E. Fortran coding for sensitivity analysis methods in Chapter 3

The sensitivity analysis methods in Chapter 3 were coded using the Fortran.90 programming language including Dial's algorithm, STOCH3 and DFS algorithms to solve logit-based SUE problems and sensitivity analysis methods for them. The programs have been applied to the small network with 6 nodes and 8 links and Kanazawa road network. Here we present the proposed algorithms for the small network, the application to Kanazawa road network is similar except that the parameters are changed.

E.1 Dial's algorithm with the MSA for solving logit-based SUE problem

```
1:    !**************Static SUE with STOCH2 Dial's algorithm**************************
2:        Integer l, i, j, k, ii, ioi, io, nori
3:        Integer, Parameter :: node = 6 !number of nodes
4:        Integer, Parameter :: link = 8 !number of links
5:        Integer, Parameter :: nod = 2 !number of OD pairs
6:        Integer, Parameter :: round = 10000 !Calculation round
7:        Real *8, Parameter :: eee = 1.0D-3 !loop termination condition
8:        Real *8, Parameter :: theta = 1 !theta parameter
9:        Real *8, Parameter :: alpha = 1 !BPR parameter
10:       Real *8, Parameter :: beta = 2 !BPR parameter
11:       Integer s(link), e(link), ls(node, node) !start node, end node and link number of network
12:       Integer ori(nod), des(nod) !Origin, Destination of OD pairs
13:       Real *8 yde(nod) !OD demand of OD pairs
14:       Real *8 t(link), cap(link), p(node, node)
15:   !t:free-flow travel time,cap: Capacity of link,p:shortest path from node to node
16:       Real *8 xo(link), xn(link) !xo: link flow after, xn:link flow before
17:       Real *8 tt(link) !tt: link travel time after update,
18:       Real *8 maxrg, g(link) !Use in MSA method
19:       Real *8 x(link), xx(node), oxx(node, node)
20:   !x: link travel is assigned, xx:total node travel in Dial's Algorithm
21:       Integer mark(node), listori(node)
22:   !*****************************************************
23:       Open (1, File='1.network.txt')
24:       Open (2, File='2.linkparameter.txt')
25:       Open (3, File='3.demand.txt')
26:       Open (11, File='11.CalculationTime(SUE).txt', Action='write')
27:
28:       Do i = 1, link
29:         Read (1, *) s(i), e(i), j !s(i):start node, e(i):end node
30:         Read (2, *) t(i), cap(i) !t(i):free flow traveltime, cap(i):Capacity of link i
31:       End Do
32:
33:       Do i = 1, nod
34:         Read (3, *) ori(i), des(i), yde(i)
35:       End Do
```

```
!c     make network
       ls = 0.0
       Do i = 1, link
         ls(s(i), e(i)) = i
       End Do

       nori = 0
       oxx = 0.0
       listori = 0
       mark = 0
       Do i = 1, nod
         io = ori(i)
         oxx(ori(i), des(i)) = yde(i)
         If (mark(io)==0) Then
           nori = nori + 1
           listori(nori) = io
           Do j = 1, nod
             If (io==ori(j)) Then
               oxx(io, des(j)) = yde(j)
             End If
           End Do
           mark(io) = 1
         End If
       End Do
       Write (6, *) 'theta', theta
       Write (6, *) 'alpha', alpha
       Write (6, *) 'beta', beta
!c***********STEP 1: SUE calculation************************

!c********* Dijkstra method calculation *****************
       Call dijikstra(p, t)

!c****** DIAL's algorithm ******************************
       xo = 0
       Do ioi = 1, nori
         io = listori(ioi)
         xx = 0.0
         xx = oxx(io, :)
         Call dial(x, xx, t, p)
!c*************************************************************
!c*****************************
!Recording after loop termination
         Do i = 1, link
           xo(i) = xo(i) + x(i)
         End Do
       End Do
!c********starting calculation round*************************************
       xn = 0.0
       g = 0.0
       Do l = 1, round !Calculate the number of calculations
!c*****Update link travel time
         Do i = 1, link
```

```
88:           tt(i) = t(i)*(1+alpha*((xo(i)/cap(i))**beta)) !BPR calculation
89:         End Do
90:   !c********* Dijkstra Recalculation ***********
91:         Call dijikstra(p, tt)
92:   !c****** DIAL's algorithm ****************************
93:         xn = 0.0
94:         Do ioi = 1, nori
95:           io = listori(ioi)
96:           xx = 0.0
97:           xx = oxx(io, :)
98:           Call dial(x, xx, tt, p)
99:   !c**************************
100:  !Recording after loop termination
101:          Do i = 1, link
102:            xn(i) = xn(i) + x(i)
103:          End Do
104:  !c***************************************
105:        End Do
106:  !c***************************************
107:  !c Convergence determination calculation
108:        k = dble(l)
109:        Do i = 1, link
110:          g(i) = xn(i) - xo(i)
111:          xn(i) = xo(i) + g(i)*(1/(1.0D0+k))
112:        End Do
113:        xo = xn
114:        Do i = 1, link
115:          maxrg = maxval(g)
116:        End Do
117:  !       Write (*, *) 'Round= ', l, 'maxRG=', maxrg
118:        If (maxrg<eee) Then !Convergence judgment
119:          ncl = l !Number of calculation loop
120:          Go To 100 !End calculation
121:        End If
122:      End Do
123:      ncl = round !When it can not converge
124:  100   Continue
125:  !c***********Results of STEP 1************************
126:      Write (*, *) 'Round', ncl
127:      Write (*, *) 'Computation gap', maxrg
128:      Call cpu_time(v)
129:      Write (11, *) 'time 1 ', v
130:
131:      Open (12, File='12.xij.txt')
132:      Do i = 1, link
133:        Write (12, *) xo(i)
134:      End Do
135:      Do i = 1, link
136:        tt(i) = t(i)*(1+alpha*((xo(i)/cap(i))**beta)) !BPR calculation
137:      End Do
138:      Open (13, File='13.tij.txt')
139:      Do i = 1, link
```

```
140:         Write (13, *) tt(i)
141:       End Do
142:
143: !c******SUBROUTINE*************************************************************
144:     Contains
145:       Subroutine dijikstra(p, t)
146:         Real *8, Intent (In) :: t(link) !link travel time
147:         Real *8, Intent (Out) :: p(node, node) !Shortest path
148:         Real *8 :: c(node), cmin(node), fin_io(node)
149:
150:         Do j = 1, node
151:           io = j !IO is the source node number
152:           c(:) = 10000000.0 !Set partial path cost • ‡
153:           c(io) = 0.0 !Set the cost of the source node to 0
154:           cmin(:) = 10000000.0
155:           fin_io(:) = 0 !Array for storing the nodes for which calculation has been completed
156:           Do While (minval(cmin,1)<50000000)
157:             Do i = 1, link
158:               If (s(i)==io) Then
159:                 If (c(io)+t(i)<=c(e(i))) Then
160:                   c(e(i)) = c(io) + t(i) !Update with the next link cost with the node
161:                   cmin(e(i)) = c(io) + t(i)
162:                 End If
163:               End If
164:             End Do
165:             ncj = minloc(cmin, 1) !Identify the location of cmin in the array
166:             cmin(ncj) = 50000000.0 !Ineffectiveness of extracted cmin
167: !c**** Calculation end judgment (End when all cmin are disabled) ***********
168:             If (minval(cmin,1)==50000000) Then
169:               Go To 100
170:             End If
171: !c******!Move to the next node******************
172:             io = ncj
173:             If (fin_io(ncj)==1) Then !When calculation is completed, move to another node
174:               inot = minloc(fin_io, 1)
175:               io = inot
176:             End If
177:             fin_io(io) = 1
178:           End Do
179: 100       Continue
180: !c******Record shortest path results************************************
181:           p(j, :) = c(:)
182:         End Do
183:
184:       End Subroutine
185:
186: !Dial's algorithm
187:       Subroutine dial(x, xx, t, p)
188:         Real *8, Intent (In) :: p(node, node), t(link)
189:         Real *8, Intent (Out) :: x(link), xx(node)
190: !x: link travel is assigned, xx:total node travel
191:         Real *8 ps(link), pe(link)
```

```
192:  !ps,pe: shortest path from each origin to start node and end node of the link
193:        Integer ss(link), ee(link)
194:  !Start node and end node in arranged shortest path
195:        Real *8 a(link), wl(link), wn(node)
196:  !a:link likelihood,wl:link weight,wn:node weight
197:        Real *8 pss(link), pee(link) !!Ps(i)Calculate in ascending order
198:
199:        Do i = 1, link
200:          ps(i) = p(io, s(i))
201:          pe(i) = p(io, e(i))
202:        End Do
203:  !c******sort shortest path from current node*****************************
204:        Do i = 1, link
205:          k = minloc(pe, 1)
206:          pss(i) = ps(k)
207:          pee(i) = pe(k)
208:          pe(k) = 10000001.0
209:          ss(i) = s(k) !ss(i)Is the s(i)
210:          ee(i) = e(k) !ee(i)Is the sorted e(i)
211:        End Do
212:  !c****** Calculation of link likeli-hood ********************
213:        a = 0.0
214:        Do i = 1, link !Ps(i)Calculate in ascending order
215:          nn = ls(ss(i), ee(i))
216:          If (pss(i)>pee(i)) Go To 100 !Check efficient path
217:          a(nn) = exp(theta*(pee(i)-pss(i)-t(nn))) !Link likelihood
218:  100    End Do
219:  !c****** Calculation of link weight and node weight*********************
220:        wl = 0.0
221:        wn = 0.0
222:        Do i = 1, link !Ps(i)Calculate in ascending sequence
223:          nn = ls(ss(i), ee(i)) !NN is the link number
224:          wn(io) = 1.0
225:          wl(nn) = wn(ss(i))*a(nn) !Link weight
226:          wn(ee(i)) = wn(ee(i)) + wl(nn) !Node weight
227:        End Do
228:  !c********* Assign trip volume************
229:        x = 0.0
230:        Do i = 1, link
231:          ii = link + 1 - i !Ps(i)Calculate in descending order
232:          nn = ls(ss(ii), ee(ii)) !NN is the link number
233:          If (wn(ee(ii))==0) Go To 110 !Prevents division by 0
234:          x(nn) = xx(ee(ii))*wl(nn)/wn(ee(ii))
235:          xx(ss(ii)) = xx(ss(ii)) + x(nn)
236:  110    End Do
237:  !c*****************************
238:
239:      End Subroutine
240:
241:    End Program
```

E.2 Old sensitivity analysis method for SUE problem based on Dial's algorithm

```
1:      !Sensitivity analysis for SUE with STOCH2 algorithm (The method of Ying and Miyagi)*
2:          Integer i, j, k, ii, ioi, io
3:          Integer, Parameter :: node = 6 !number of nodes
4:          Integer, Parameter :: link = 8 !number of links
5:          Integer, Parameter :: nod = 2 !number of OD pairs
6:          Real *8, Parameter :: theta = 1 !theta parameter
7:          Real *8, Parameter :: alpha = 1 !BPR parameter
8:          Real *8, Parameter :: beta = 2 !BPR parameter
9:          Integer s(link), e(link), ls(node, node) !start node, end node and link number of network
10:         Integer ori(nod), des(nod) !Origin, Destination of OD pairs
11:         Real *8 yde(nod) !OD demand of OD pairs
12:         Real *8 t(link), cap(link), p(node, node)
13:     !t:free-flow travel time,cap: Capacity of link,p:shortest path from node to node
14:         Real *8 xo(link) !xo: link flow at SUE state
15:         Real *8 tt(link) !tt: link travel time after update,
16:         Real *8 x(link), xx(node)
17:     !x: link travel is assigned, xx:total node travel in Dial's Algorithm
18:         Real *8 ps(link), pe(link)
19:     !ps,pe: shortest path from each origin to start node and end node of the link
20:         Integer ss(link), ee(link)
21:     !Start node and end node in arranged shortest path
22:         Real *8 a(link), wl(link), wn(node)
23:     !a:link likelihood,wl:link weight,wn:node weight
24:         Real *8 pss(link), pee(link) !pss(i),pss(i):Used in calculation in ascending order
25:         Real *8 xijrs(link), pjsgh(nod, node, link) !xijrs,pjsgh
26:         Real *8 xijgh(link, link), xijgh1(link, link)
27:         Real *8 tx(link) !tx:derivative of t respect to x
28:         Real *8 xt(link), xzeta(link), tzeta(link)
29:         Real *8 hessian(link, link)
30:         Real *8 gradt(link, link), gradx(link, link)
31:         Real *8 www, pivot, gradt2(link, 2*link)
32:         Real *8 zetaij(link) !zetaij:Total eliminated flow
33:         Real *8 xafter(link) !link travel flow after sensitivity analysis
34:
35:     !c*****************************************************
36:         Open (1, File='1.network.txt', Action='read')
37:         Open (2, File='2.linkparameter.txt', Action='read')
38:         Open (3, File='3.demand.txt', Action='read')
39:         Open (19, File='19.CalculationTime(SA).txt', Action='write')
40:
41:         Do i = 1, link
42:           Read (1, *) s(i), e(i), j !s(i):start node, e(i):end node
43:           Read (2, *) t(i), cap(i) !t(i):free flow traveltime, cap(i):Capacity of link i
44:         End Do
45:
46:         Do i = 1, nod
47:           Read (3, *) ori(i), des(i), yde(i)
48:         End Do
49:
50:     !c    make network
51:         ls = 0.0
```

```
52:         Do i = 1, link
53:           ls(s(i), e(i)) = i
54:         End Do
55:
56:         Write (6, *) 'theta', theta
57:         Write (6, *) 'alpha', alpha
58:         Write (6, *) 'beta', beta
59:
60:         Open (12, File='12.xij.txt')
61:         Do i = 1, link
62:           Read (12, *) xo(i)
63:         End Do
64:         Do i = 1, link
65:           tt(i) = t(i)*(1+alpha*((xo(i)/cap(i))**beta)) !BPR calculation
66:         End Do
67:
68:         Do i = 1, link
69:           tx(i) = t(i)*beta*alpha*(xo(i)**(beta-1))/(cap(i)**beta)
70:           xt(i) = 1.0D0/tx(i)
71:           tzeta(i) = 1.0D0 + alpha*xo(i)**beta/(cap(i)**beta)
72:           xzeta(i) = -tzeta(i)/tx(i)
73:         End Do
74:
75:         Open (21, File='21.tx.txt')
76:         Do i = 1, link
77:           Write (21, *) tx(i)
78:         End Do
79:         Write (*, *) 'finish_tx'
80:
81:         hessian = 0.0
82:         pjsgh = 0.0
83:         xijgh = 0.0
84:         xijgh1 = 0.0
85:
86:     !c********* Dijkstra method calculation ****************
87:         Call dijikstra(p, tt)
88:
89:         Open (22, File='22.xijrs.txt')
90:         Open (23, File='23.pjsgh.txt')
91:         Open (24, File='24.xijgh.txt')
92:
93:     !c****** DIAL's algorithm with each OD pair ******************************
94:         Do ioi = 1, nod
95:
96:           io = ori(ioi)
97:
98:     !c****** Sort***************
99:           xx = 0.0
100:          xx(des(ioi)) = yde(ioi) !Setting the initial value
101:
102:          Do i = 1, link
103:            ps(i) = p(io, s(i))
```

```
104:          pe(i) = p(io, e(i))
105:        End Do
106:
107:  !c******sort shortest path from current node*****************************
108:        Do i = 1, link
109:          k = minloc(pe, 1)
110:          pss(i) = ps(k)
111:          pee(i) = pe(k)
112:          pe(k) = 10000000.0
113:          ss(i) = s(k) !ss(i)Is the s(i)
114:          ee(i) = e(k) !ee(i)Is the sorted e(i)
115:        End Do
116:  !c****** Calculation of link likeli-hood ********************
117:        a = 0.0
118:        Do i = 1, link !Ps(i)Calculate in ascending order
119:          nn = ls(ss(i), ee(i))
120:          If (pss(i)>pee(i)) Go To 100 !Check efficient path
121:          If (pss(i)==10000000.0 .Or. pee(i)==10000000.0) Go To 100
122:          a(nn) = exp(theta*(pee(i)-pss(i)-tt(nn))) !Link likelihood
123:  100     End Do
124:  !c****** Calculation of link weight and node weight*********************
125:        wl = 0.0
126:        wn = 0.0
127:        Do i = 1, link !Ps(i)Calculate in ascending sequence
128:          nn = ls(ss(i), ee(i)) !NN is the link number
129:          wn(io) = 1.0
130:          If (i>=2) Then
131:            Do j = 1, i - 1
132:              If ((ss(i)==ss(j)) .And. (ee(i)==ee(j))) Go To 110
133:            End Do
134:          End If
135:          wl(nn) = wn(ss(i))*a(nn) !Link weight
136:          wn(ee(i)) = wn(ee(i)) + wl(nn) !Node weight
137:  110     End Do
138:  !c********* Assign trip volume************
139:        x = 0.0
140:        Do i = 1, link
141:          ii = link + 1 - i !Ps(i)Calculate in descending order
142:          nn = ls(ss(ii), ee(ii)) !NN is the link number
143:          If (wn(ee(ii))==0) Go To 120 !Prevents division by 0
144:          x(nn) = xx(ee(ii))*wl(nn)/wn(ee(ii))
145:          xx(ss(ii)) = xx(ss(ii)) + x(nn)
146:  120     End Do
147:  !c***************************
148:        xijrs = x
149:        Write (*, *) 'finish_xijrs'
150:  !c************************************************************
151:
152:  !*******Calculate pjsgh*****
153:        Do kk = 1, link
154:          If (xijrs(kk)/=0) Then
155:            Write (22, *) kk, xijrs(kk)
```

```
156:            io = e(kk)
157:            Do ioi1 = 1, ioi - 1
158:              If (des(ioi1)==des(ioi)) Then
159:                Do i = 1, link
160:                  If (pjsgh(ioi1,e(kk),i)/=0) Then
161:                    pjsgh(ioi, e(kk), i) = pjsgh(ioi1, e(kk), i)
162:                  End If
163:                End Do
164:                Go To 160
165:              End If
166:            End Do
167:  !c****** Sort***************
168:            xx = 0.0
169:            xx(des(ioi)) = 1 !Setting the initial value
170:
171:            Do i = 1, link
172:              ps(i) = p(io, s(i))
173:              pe(i) = p(io, e(i))
174:            End Do
175:  !c******sort shortest path from current node*****************************
176:            Do i = 1, link
177:              k = minloc(pe, 1)
178:              pss(i) = ps(k)
179:              pee(i) = pe(k)
180:              pe(k) = 10000000.0
181:              ss(i) = s(k) !ss(i)Is the s(i)
182:              ee(i) = e(k) !ee(i)Is the sorted e(i)
183:            End Do
184:  !c****** Calculation of link likeli-hood **********************
185:            a = 0.0
186:            Do i = 1, link !Ps(i)Calculate in ascending order
187:              nn = ls(ss(i), ee(i))
188:              If (pss(i)>pee(i)) Go To 130 !Check efficient path
189:              If (pss(i)==10000000.0 .Or. pee(i)==10000000.0) Go To 130
190:              a(nn) = exp(theta*(pee(i)-pss(i)-tt(nn))) !Link likelihood
191:  130         End Do
192:  !c****** Calculation of link weight and node weight**********************
193:            wl = 0.0
194:            wn = 0.0
195:            Do i = 1, link !Ps(i)Calculate in ascending sequence
196:              nn = ls(ss(i), ee(i)) !NN is the link number
197:              wn(io) = 1.0
198:              If (i>=2) Then
199:                Do j = 1, i - 1
200:                  If ((ss(i)==ss(j)) .And. (ee(i)==ee(j))) Go To 140
201:                End Do
202:              End If
203:              wl(nn) = wn(ss(i))*a(nn) !Link weight
204:              wn(ee(i)) = wn(ee(i)) + wl(nn) !Node weight
205:  140         End Do
206:  !c********* Assign trip volume************
207:            x = 0.0
```

```
208:            Do i = 1, link
209:              ii = link + 1 - i !Ps(i)Calculate in descending order
210:              nn = ls(ss(ii), ee(ii)) !NN is the link number
211:              If (wn(ee(ii))==0) Go To 150 !Prevents division by 0
212:              x(nn) = xx(ee(ii))*wl(nn)/wn(ee(ii))
213:              xx(ss(ii)) = xx(ss(ii)) + x(nn)
214:  150         End Do
215:  !c*****************************
216:            Do i = 1, link
217:              If (x(i)/=0) Then
218:                pjsgh(ioi, e(kk), i) = x(i)
219:                Write (23, *) e(kk), des(ioi), i, x(i)
220:              End If
221:            End Do
222:          End If
223:  160     End Do
224:
225:         Do i = 1, link
226:           Do j = 1, link
227:             xijgh1(i, j) = xijrs(i)*pjsgh(ioi, e(i), j)
228:           End Do
229:         End Do
230:
231:         xijgh = 0.0
232:         Do i = 1, link
233:           Do j = 1, link
234:             If (xijgh1(i,j)/=0) Then
235:               Write (24, *) i, j, xijgh1(i, j)
236:             End If
237:             xijgh(i, j) = xijgh1(i, j) + xijgh1(j, i)
238:           End Do
239:           xijgh(i, i) = xijrs(i)
240:         End Do
241:
242:         Do i = 1, link
243:           Do j = 1, link
244:             hessian(i, j) = hessian(i, j) + theta*(xijrs(i)*xijrs(j)/yde(ioi)- &
245:               xijgh(i,j))
246:           End Do
247:         End Do
248:         Write (*, *) 'finish_xijgh'
249:       End Do
250:  !c*************************************************************
251:  !*******Record results*****
252:       Open (25, File='25.hessian.txt')
253:       Do i = 1, link
254:         Write (25, *) hessian(i, :)
255:       End Do
256:       Write (*, *) 'finish_hessian'
257:
258:       Do i = 1, link
259:         Do j = 1, link
```

```
260:          If (i==j) Then
261:            gradt(i, j) = xt(i) - hessian(i, j)
262:          Else
263:            gradt(i, j) = -hessian(i, j)
264:          End If
265:        End Do
266:      End Do
267:
268:      Do i = 1, link
269:        Do j = 1, link
270:          gradt2(i, j) = gradt(i, j)
271:        End Do
272:      End Do
273:      Do i = 1, link
274:        Do j = 1, link
275:          gradt2(i, j+link) = 0
276:          gradt2(i, i+link) = 1
277:        End Do
278:      End Do
279:
280:      Do k = 1, link
281:        pivot = gradt2(k, k)
282:        If (pivot==0) Then
283:          i = k + 1
284:          Do While (pivot==0 .And. i<link)
285:            pivot = gradt2(i, k)
286:            i = i + 1
287:          End Do
288:          If (pivot==0) Then
289:            Stop
290:          Else
291:            Do j = 1, 2*link
292:              www = gradt2(k, j)
293:              i = i - 1
294:              gradt2(k, j) = gradt2(i, j)
295:              gradt2(i, j) = www
296:            End Do
297:          End If
298:        End If
299:        Do j = 1, link*2
300:          gradt2(k, j) = gradt2(k, j)/pivot
301:        End Do
302:
303:        Do i = 1, link
304:          If (i/=k) Then
305:            www = gradt2(i, k)
306:            Do j = 1, link*2
307:              gradt2(i, j) = gradt2(i, j) - www*gradt2(k, j)
308:            End Do
309:          End If
310:        End Do
311:      End Do
```

```
312:
313:       gradt = 0.0
314:       Do i = 1, link
315:         Do j = 1, link
316:           gradt(i, j) = -gradt2(i, j+link)*xzeta(j)
317:         End Do
318:       End Do
319:
320:       gradx = 0.0
321:       Do i = 1, link
322:         Do j = 1, link
323:           If (i==j) Then
324:             gradx(i, j) = xzeta(i) + xt(i)*gradt(i, j)
325:           Else
326:             gradx(i, j) = xt(i)*gradt(i, j)
327:           End If
328:         End Do
329:       End Do
330:       Open (26, File='26.Grad.txt')
331:       Do i = 1, link
332:         Write (26, *) gradx(i, :)
333:       End Do
334:  !c******Calculate link travel flow after Sensitivity analysis*****************
335:       zetaij = 0
336:       Do i = 1, link
337:         zetaij(i) = 1
338:       End Do
339:       xafter = matmul(gradx, zetaij)
340:
341:       Do i = 1, link
342:         xafter(i) = xo(i) + xafter(i)
343:       End Do
344:
345:       Open (35, File='35.xafter.txt')
346:       Do i = 1, link
347:         Write (35, *) xafter(i)
348:       End Do
349:
350:       Call cpu_time(v)
351:       Write (19, *) 'time of calculation', v
352:
353:  !c******SUBROUTINE************************************************************
354:     Contains
355:  !c     Dijikstra method
356:       Subroutine dijikstra(p, t)
357:         Real *8, Intent (In) :: t(link) !link travel time
358:         Real *8, Intent (Out) :: p(node, node) !Shortest path
359:         Real *8 :: c(node), cmin(node), fin_io(node)
360:
361:         Do j = 1, node
362:           io = j !IO is the source node number
363:           c(:) = 10000000.0 !Set partial path cost • ‡
```

```
364:          c(io) = 0.0 !Set the cost of the source node to 0
365:          cmin(:) = 10000000.0
366:          fin_io(:) = 0 !Array for storing the nodes for which calculation has been completed
367:          Do While (minval(cmin,1)<50000000)
368:            Do i = 1, link
369:              If (s(i)==io) Then
370:                If (c(io)+t(i)<=c(e(i))) Then
371:                  c(e(i)) = c(io) + t(i) !Update with the next link cost with the node
372:                  cmin(e(i)) = c(io) + t(i)
373:                End If
374:              End If
375:            End Do
376:            ncj = minloc(cmin, 1) !Identify the location of cmin in the array
377:            cmin(ncj) = 50000000.0 !Ineffectiveness of extracted cmin
378:  !c**** Calculation end judgment (End when all cmin are disabled) ***********
379:            If (minval(cmin,1)==50000000) Then
380:              Go To 100
381:            End If
382:  !c******!Move to the next node******************
383:            io = ncj
384:            If (fin_io(ncj)==1) Then !When calculation is completed, move to another node
385:              inot = minloc(fin_io, 1)
386:              io = inot
387:            End If
388:            fin_io(io) = 1
389:          End Do
390:  100       Continue
391:  !c******Record shortest path results***********************************
392:          p(j, :) = c(:)
393:        End Do
394:
395:      End Subroutine
396:
397:  !c******FINISH SUBROUTINE***********************************
398:    End Program
```

E.3 STOCH3 algorithm with the MSA for solving logit-based SUE problem

```
!***Stochastic User Equilibrium with STOCH3 algorithm***
      Integer i, j, k, ii, ioi, io, count, nori
      Integer, Parameter :: node = 6 !number of node
      Integer, Parameter :: link = 8 !number of link
      Integer, Parameter :: nod = 2 !number of OD pairs
      Real *8, Parameter :: eee = 1.0D-6 !loop termination condition
      Real *8, Parameter :: theta = 1.0D0 !theta parameter
      Real *8, Parameter :: alpha = 1.0D0 !BPR parameter
      Real *8, Parameter :: beta = 2.0D0 !BPR parameter
      Integer s(link), e(link), ls(node, node) !start and end node and link number
      Integer ori(nod), des(nod) !origin, destination of od pairs
      Real *8 yde(nod) !od demand of od pairs
      Real t(link), cap(link) !t:free-flow travel time,cap: capacity of link
      Real *8 erh(link), p(node, node) !erh: stoch3 parameter, p:shortest route
      Integer headnode(0:node), anode(link), alink(link) ! adjacent nodes and links
      Real *8 ps(link), pe(link)
!ps,pe: shortest path from each origin to start and end node of the link
      Integer ss(node, link), ee(node, link)
!ss: Start node,ee: end node in arranged shortest path
      Real *8 a(link), wl(link), wn(node)
!a:link likelihood,w:link weight,wn:node weight
      Integer omega(node, link) !Used to check STOCH3-efficient path
      Real *8 xo(link), xn(link) !xo: link traffic flow after, xn:link traffic flow before
      Real *8 tt(link) !tt: link travel time after update,
      Real *8 maxrg, g(link) !Used in MSA method
      Real *8 x(link), xx(node), oxx(node, node)
!x: link travel is assigned, xx:total node travel in STOCH3's Algorithm
      Integer mark(node), listori(node)
!*****************************************************

      Open (1, File='1.network.txt', Action='read')
      Open (2, File='2.linkparameter.txt', Action='read')
      Open (3, File='3.demand.txt', Action='read')
      Open (11, File='11.CalculationTime.txt', Action='write')

      Do i = 1, link
        Read (1, *) s(i), e(i), j !s(i):start node, e(i):end node
        Read (2, *) t(i), cap(i) !t(i):free flow traveltime, cap(i):Capacity of link i
!       erh(i) = 1.5
      End Do
      Do i = 1, nod
        Read (3, *) ori(i), des(i), yde(i)
      End Do
!make network
      ls = 0.0
      Do i = 1, link
        ls(s(i), e(i)) = i
      End Do
! create adjacent nodes and links
      headnode(0) = 0
      count = 0
```

```
52:         Do i = 1, node
53:           Do j = 1, link
54:             If (i==s(j)) Then
55:               count = count + 1
56:               anode(count) = e(j)
57:               alink(count) = j
58:             End If
59:           End Do
60:           headnode(i) = count
61:         End Do
62:         nori = 0
63:         oxx = 0.0
64:         listori = 0
65:         mark = 0
66:         Do i = 1, nod
67:           io = ori(i)
68:           oxx(ori(i), des(i)) = yde(i)
69:           If (mark(io)==0) Then
70:             nori = nori + 1
71:             listori(nori) = io
72:             Do j = 1, nod
73:               If (io==ori(j)) Then
74:                 oxx(io, des(j)) = yde(j)
75:               End If
76:             End Do
77:             mark(io) = 1
78:           End If
79:         End Do
80:         Write (6, *) 'theta', theta
81:         Write (6, *) 'alpha', alpha
82:         Write (6, *) 'beta', beta
83:
84:     !********* Dijkstra method calculation ****************
85:         Call dijikstra(p, t)
86:
87:     !****** STEP1: Static SUE by STOCH3 algorithm ******************************
88:         omega = 0.0D0
89:     !     mark = 0
90:
91:     !****** Step 0: Preliminaries ******************************
92:         Do ioi = 1, nori
93:           io = listori(ioi)
94:     !       write (6,*) io,xx(io,31),xx(io,38)
95:           Do i = 1, link
96:             ps(i) = p(io, s(i))
97:             pe(i) = p(io, e(i))
98:           End Do
99:     !******efficient path ******************************
100:          Do i = 1, link
101:            If (pe(i)>ps(i)) Then
102:    !         If ((1+erh(i))*(pe(i)-ps(i))>=t(i)) Then
103:              omega(io, i) = 1
```

```
104:  !           End If
105:            End If
106:          End Do
107:
108:  !******Sorting shortest path from current node*****************************
109:          Do i = 1, link
110:            k = minloc(pe, 1)
111:            pe(k) = 10000001.0
112:            ss(io, i) = s(k) !ss(i)Is the sorted s(i)
113:            ee(io, i) = e(k) !ee(i)Is the sorted e(i)
114:          End Do
115:        End Do
116:  !****** Step 1: Calculation of link likeli-hood *********************
117:        a = 0.0
118:        xo = 0
119:        Do i = 1, link
120:          a(i) = exp(-theta*(t(i)))
121:        End Do
122:
123:  !****** STEP2,3,4: Calculating for each OD pair*********************
124:        mark = 0
125:        Do ioi = 1, nori
126:
127:  !****** Step 2: Forward pass: Calculation of link weight and node weight******
128:          io = listori(ioi) !IO is origin the  node number
129:          wl = 0.0D0
130:          wn = 0.0D0
131:          Do i = 1, link !Ps(i)Calculate in ascending sequence
132:            nn = ls(ss(io,i), ee(io,i)) !nn is the link number
133:            wn(io) = 1.0
134:            If (omega(io,nn)==1) Then
135:              wl(nn) = wn(ss(io,i))*a(nn) !Link weight
136:              wn(ee(io,i)) = wn(ee(io,i)) + wl(nn) !Node weight
137:            End If
138:          End Do
139:
140:  !**********Step 3: Backward pass: Assign trip volume************
141:          x = 0.0D0
142:          xx = oxx(io, :)
143:          Do i = 1, link
144:            ii = link + 1 - i !Ps(i)Calculate in descending order
145:            nn = ls(ss(io,ii), ee(io,ii)) !nn is the link number
146:            If (wn(ee(io,ii))==0) Go To 100 !Prevents division by 0
147:            If (omega(io,nn)==1) Then
148:              x(nn) = xx(ee(io,ii))*wl(nn)/wn(ee(io,ii))
149:              xx(ss(io,ii)) = xx(ss(io,ii)) + x(nn)
150:            End If
151:  100     End Do
152:
153:  !**********Step 4: Contribution to total link-flows*****************
154:          Do i = 1, link
155:            xo(i) = xo(i) + x(i)
```

```
156:          End Do
157:
158:        End Do
159:
160:  !********starting calculation round************************************
161:        g = 0.0
162:        maxrg = 1
163:        l = 0
164:  !      write (6,*) xo
165:        Do While (maxrg>eee)
166:
167:  !*****Step 5: Update link travel time and link likeli-hood
168:          l = l + 1
169:          Do i = 1, link
170:            tt(i) = t(i)*(1+alpha*((xo(i)/cap(i))**beta)) !BPR calculation
171:            a(i) = exp(-theta*(tt(i))) !Update link likeli-hood
172:          End Do
173:
174:  !****** Step 6: Direction finding ******************************
175:          xn = 0
176:
177:  !****** Step 6.1,6.2,6.3: Calculating for each OD pair *********************
178:          Do ioi = 1, nori
179:
180:  !****** Step 6.1: Forward pass: Calculation of link weight and node weight*******
181:
182:            io = listori(ioi) !IO is origin the  node number
183:            wl = 0.0D0
184:            wn = 0.0D0
185:            Do i = 1, link !Ps(i)Calculate in ascending sequence
186:              nn = ls(ss(io,i), ee(io,i)) !nn is the link number
187:              wn(io) = 1.0
188:              If (omega(io,nn)==1) Then
189:                wl(nn) = wn(ss(io,i))*a(nn) !Link weight
190:                wn(ee(io,i)) = wn(ee(io,i)) + wl(nn) !Node weight
191:              End If
192:            End Do
193:
194:  !*********Step 3: Backward pass: Assign trip volume************
195:            x = 0.0D0
196:            xx = oxx(io, :)
197:            Do i = 1, link
198:              ii = link + 1 - i !Ps(i)Calculate in descending order
199:              nn = ls(ss(io,ii), ee(io,ii)) !nn is the link number
200:              If (wn(ee(io,ii))==0) Go To 110 !Prevents division by 0
201:              If (omega(io,nn)==1) Then
202:                x(nn) = xx(ee(io,ii))*wl(nn)/wn(ee(io,ii))
203:                xx(ss(io,ii)) = xx(ss(io,ii)) + x(nn)
204:              End If
205:  110       End Do
206:
207:  !*********Step 6.3: Contribution to total link-flows******************
```

```
208:           Do i = 1, link
209:             xn(i) = xn(i) + x(i)
210:           End Do
211:
212:  !************************************
213:         End Do
214:
215:  !*********Step 7: Link flow update*****************************
216:         k = dble(l)
217:         Do i = 1, link
218:           g(i) = xn(i) - xo(i)
219:           xn(i) = xo(i) + g(i)*(1.0D0/(1.0D0+k))
220:         End Do
221:
222:  !*********Step 8: Link flow update*****************************
223:         xo = xn
224:         maxrg = maxval(g)
225:  !       Write (*, *) 'Round', l, 'maxRG', maxrg
226:       End Do
227:
228:  !*********RECORDING THE RESULTS*****************************
229:       Write (11, *) 'Round of Static SUE', l, 'Computation gap', maxrg
230:       Call cpu_time(v)
231:       Write (11, *) 'Calculation time 1 (second) ', v
232:       Open (12, File='12.xij.txt')
233:       Do i = 1, link
234:         Write (12, *) xo(i)
235:       End Do
236:       Do i = 1, link
237:         tt(i) = t(i)*(1+alpha*((xo(i)/cap(i))**beta)) !BPR calculation
238:       End Do
239:       Open (13, File='13.tij.txt')
240:       Do i = 1, link
241:         Write (13, *) tt(i)
242:       End Do
243:
244:  !******SUBROUTINE*************************************************************
245:     Contains
246:       Subroutine dijikstra(p, t)
247:         Real, Intent (In) :: t(link)
248:         Real *8, Intent (Out) :: p(node, node)
249:         Real *8 :: c(node), cmin(node), fin_io(node)
250:
251:         Do j = 1, node
252:           io = j !io is the source node number
253:           c(:) = 10000000.0 !set partial path cost • ‡
254:           c(io) = 0.0 !set the cost of the source node to 0
255:           cmin(:) = 10000000.0
256:           fin_io(:) = 0 !array for storing the nodes for which calculation has been completed
257:           Do While (minval(cmin,1)<50000000)
258:             Do iii = headnode(io-1) + 1, headnode(io)
259:               i = ls(io, anode(iii))
```

```
260:              If (c(io)+t(i)<=c(e(i))) Then
261:                c(e(i)) = c(io) + t(i) !update with the next link cost with the node
262:                cmin(e(i)) = c(io) + t(i)
263:              End If
264:            End Do
265:            ncj = minloc(cmin, 1) !identify the location of cmin in the array
266:            cmin(ncj) = 50000000.0 !ineffectiveness of extracted cmin
267:  !**** calculation end judgment (end when all cmin are disabled) ***********
268:            If (minval(cmin,1)==50000000) Then
269:              Go To 100
270:            End If
271:  !******!move to the next node******************
272:            io = ncj
273:            If (fin_io(ncj)==1) Then !when calculation is completed, move to another node
274:              inot = minloc(fin_io, 1)
275:              io = inot
276:            End If
277:            fin_io(io) = 1
278:          End Do
279:  100       Continue
280:  !******record shortest path results***********************************
281:          p(j, :) = c(:)
282:        End Do
283:
284:      End Subroutine
285:
286:  !******FINISH SUBROUTINE***********************************
287:    End Program
```

E.4 Old sensitivity analysis method for SUE problem based on STOCH3 algorithm

```
1:    !Sensitivity analysis for SUE with STOCH3 algorithm (The method of Ying and Miyagi)*
2:        Integer i, j, k, ii, ioi, io, count
3:        Integer, Parameter :: node = 6 !number of nodes
4:        Integer, Parameter :: link = 8 !number of links
5:        Integer, Parameter :: nod = 2 !number of OD pairs
6:        Real *8, Parameter :: theta = 1.0D0 !theta parameter
7:        Real *8, Parameter :: alpha = 1.0D0 !BPR parameter
8:        Real *8, Parameter :: beta = 2.0D0 !BPR parameter
9:        Integer s(link), e(link), ls(node, node) !start and end node and link number
10:       Integer ori(nod), des(nod) !origin, destination of od pairs
11:       Real *8 yde(nod) !od demand of od pairs
12:       Real t(link), cap(link) !t:free-flow travel time,cap: capacity of link
13:       Real *8 erh(link), p(node, node) !erh: stoch3 parameter, p:shortest route
14:       Integer headnode(0:node), anode(link) ! adjacent nodes
15:       Real *8 ps(link), pe(link)
16:   !ps,pe: shortest path from each origin to start and end node of the link
17:       Integer ss(link), ee(link)
18:   !ss: Start node,ee: end node in arranged shortest path
19:       Real *8 a(link), wl(link), wn(node)
20:   !a:link likelihood,w:link weight,wn:node weight
21:       Integer omega(nod, link) !Used to check STOCH3-efficient path
22:       Real *8 xo(link) !xo: link traffic flow at SUE
23:       Real *8 tt(link) !tt: link travel time at SUE
24:       Real *8 x(link), xx(node)
25:   !x: link travel is assigned, xx:total node travel in STOCH3's Algorithm
26:       Real *8 xijrs(link)
27:       Real *8 xijgh1(link, link), xijgh(link, link) ! Used to calculate xijgh
28:       Real *8 tx(link) !tx:derivative of t respect to x
29:       Real *8 hessian(link, link)
30:       Real *8 gradt(link, link), gradzeta(link, link)
31:       Real *8 www, pivot, gradt2(link, 2*link)
32:       Real *8 xi(nod), gradxi(link, nod), gradq(link, nod), gradxi1(link, nod)
33:       Real *8 zeta(link)
34:       Real *8 xafter(link) !link travel flow after sensitivity analysis
35:       Integer mark(node, node), omega1(link)
36:       Real *8 xt(link), xzeta(link)
37:       Real *8, Allocatable :: pjsgh(:, :, :)
38:   !*************************************************
39:
40:       Open (1, File='1.network.txt', Action='read')
41:       Open (2, File='2.linkparameter.txt', Action='read')
42:       Open (3, File='3.demand.txt', Action='read')
43:       Open (19, File='19.CalculationTime(SA).txt', Action='write')
44:       Do i = 1, link
45:         Read (1, *) s(i), e(i), j !s(i):start node, e(i):end node
46:         Read (2, *) t(i), cap(i) !t(i):free flow traveltime, cap(i):Capacity of link i
47:         erh(i) = 1.5
48:       End Do
49:       Do i = 1, nod
```

```
50:           Read (3, *) ori(i), des(i), yde(i)
51:         End Do
52:     !make network
53:         ls = 0.0
54:         Do i = 1, link
55:           ls(s(i), e(i)) = i
56:         End Do
57:     ! create adjacent nodes and links
58:         headnode(0) = 0
59:         count = 0
60:         Do i = 1, node
61:           Do j = 1, link
62:             If (i==s(j)) Then
63:               count = count + 1
64:               anode(count) = e(j)
65:     !          alink(count) = j
66:             End If
67:           End Do
68:           headnode(i) = count
69:         End Do
70:
71:         Write (6, *) 'theta', theta
72:         Write (6, *) 'alpha', alpha
73:         Write (6, *) 'beta', beta
74:
75:         Open (12, File='12.xij.txt')
76:         Do i = 1, link
77:           Read (12, *) xo(i)
78:         End Do
79:         tt = 0.0
80:         Do i = 1, link
81:           tt(i) = t(i)*(1+alpha*((xo(i)/cap(i))**beta)) !BPR calculation
82:         End Do
83:     !***** Calculating apparent derivitives of BPR function*****************
84:         tx = 0.0
85:         xt = 0.0
86:         xzeta = 0.0
87:         Do i = 1, link
88:           If (xo(i)/=0) Then
89:             tx(i) = t(i)*beta*alpha*(xo(i)**(beta-1))/(cap(i)**beta)
90:             xt(i) = 1.0D0/tx(i)
91:             xzeta(i) = -(cap(i)**beta+alpha*xo(i)**beta)/ &
92:               (t(i)*beta*alpha*(xo(i)**(beta-1)))
93:           End If
94:         End Do
95:         Open (21, File='21.tx.txt')
96:         Do i = 1, link
97:           Write (21, *) tx(i)
98:         End Do
99:         Write (*, *) 'finish_tx'
100:
101:        Open (22, File='22.xijrs.txt')
```

```
      Open (23, File='23.pjsgh.txt')
      Open (24, File='24.xijgh.txt')
      omega = 0
      hessian = 0.0
      xijrs = 0
      gradq = 0.0
      Allocate (pjsgh(node,node,link))
!      graduq=0.0
      pjsgh = 0

!******** Dijkstra method calculation ****************
      Call dijikstra(p, t)

!************************
      mark = 0
      a = 0.0
      Do i = 1, link
        a(i) = exp(-theta*(tt(i)))
      End Do
!********STOCH3 algorithm with each OD pair*****************************
      Do ioi = 1, nod

        io = ori(ioi)
        xx = 0.0
        xx(des(ioi)) = yde(ioi)

        Do i = 1, link
          ps(i) = p(io, s(i))
          pe(i) = p(io, e(i))
        End Do
        Do i = 1, link
          If (pe(i)>ps(i)) Then
!          If ((1+erh(i))*(pe(i)-ps(i))>=t(i)) Then
            omega(ioi, i) = 1
!          End If
          End If
        End Do

!*****sort shortest path from current node*****************************
        Do i = 1, link
          k = minloc(pe, 1)
          pe(k) = 10000001.0
          ss(i) = s(k)
          ee(i) = e(k)
        End Do
!***** Calculation of link weight and node weight*********************
        wl = 0.0
        wn = 0.0
        Do i = 1, link !Ps(i)Calculate in ascending sequence
          nn = ls(ss(i), ee(i)) !NN is the link number
          wn(io) = 1.0
          If (omega(ioi,nn)==1) Then
```

```
154:            wl(nn) = wn(ss(i))*a(nn) !Link weight
155:            wn(ee(i)) = wn(ee(i)) + wl(nn) !Node weight
156:          End If
157:        End Do
158:  !******** Assign trip volume************
159:        x = 0.0
160:        Do i = 1, link
161:          ii = link + 1 - i !Ps(i)Calculate in descending order
162:          nn = ls(ss(ii), ee(ii)) !NN is the link number
163:          If (wn(ee(ii))==0) Go To 100 !Prevents division by 0
164:          If (omega(ioi,nn)==1) Then
165:            x(nn) = xx(ee(ii))*wl(nn)/wn(ee(ii))
166:            xx(ss(ii)) = xx(ss(ii)) + x(nn)
167:          End If
168:  100     End Do
169:  !     write (*,*) 'finish_xijrs'
170:  !************************************************************
171:        xijrs = x
172:        Do i = 1, link
173:          gradq(i, ioi) = -x(i)/yde(ioi)
174:          Write (22, *) ori(ioi), des(ioi), s(i), e(i), xijrs(i)
175:        End Do
176:  !
177:  !*************
178:        Do kk = 1, link
179:          If (xijrs(kk)/=0) Then
180:            io = e(kk)
181:            If (mark(des(ioi),io)==1) Then
182:              Go To 120
183:            End If
184:            mark(des(ioi), io) = 1
185:
186:            xx = 0.0
187:            xx(des(ioi)) = 1.0
188:
189:            Do i = 1, link
190:              ps(i) = p(io, s(i))
191:              pe(i) = p(io, e(i))
192:            End Do
193:            omega1 = 0
194:            Do i = 1, link
195:              If (pe(i)>ps(i)) Then
196:  !             If ((1+erh(i))*(pe(i)-ps(i))>=t(i)) Then
197:                omega1(i) = 1
198:  !             End If
199:              End If
200:            End Do
201:            Do i = 1, link
202:              k = minloc(pe, 1)
203:              pe(k) = 10000001.0
204:              ss(i) = s(k)
205:              ee(i) = e(k)
```

```
206:            End Do
207:
208:  !***** Calculation of link weight and node weight*********************
209:            wl = 0.0
210:            wn = 0.0
211:            Do i = 1, link !Ps(i)Calculate in ascending sequence
212:              nn = ls(ss(i), ee(i)) !NN is the link number
213:              wn(io) = 1.0
214:              If (omega1(nn)==1) Then
215:                wl(nn) = wn(ss(i))*a(nn) !Link weight
216:                wn(ee(i)) = wn(ee(i)) + wl(nn) !Node weight
217:              End If
218:            End Do
219:  !******** Assign trip volume************
220:            x = 0.0
221:            Do i = 1, link
222:              ii = link + 1 - i !Ps(i)Calculate in descending order
223:              nn = ls(ss(ii), ee(ii)) !NN is the link number
224:              If (wn(ee(ii))==0) Go To 110 !Prevents division by 0
225:              If (omega1(nn)==1) Then
226:                x(nn) = xx(ee(ii))*wl(nn)/wn(ee(ii))
227:                xx(ss(ii)) = xx(ss(ii)) + x(nn)
228:              End If
229:  110         End Do
230:  !***************************
231:
232:            Do i = 1, link
233:              If (x(i)/=0) Then
234:                pjsgh(des(ioi), io, i) = x(i)
235:                Write (23, *) e(kk), des(ioi), s(i), e(i), x(i)
236:              End If
237:            End Do
238:
239:          End If
240:  120     End Do
241:
242:         xijgh1 = 0.0
243:         Do i = 1, link
244:           Do j = 1, link
245:             xijgh1(i, j) = xijrs(i)*pjsgh(des(ioi), e(i), j)
246:             If (xijgh1(i,j)/=0) Write (24, *) ori(ioi), des(ioi), s(i), e(i), &
247:               s(j), e(j), xijgh1(i, j)
248:           End Do
249:         End Do
250:         xijgh = 0.0
251:         Do i = 1, link
252:           Do j = 1, link
253:             xijgh(i, j) = xijgh1(i, j) + xijgh1(j, i)
254:           End Do
255:           xijgh(i, i) = xijrs(i)
256:         End Do
257:
```

```
258:          Do i = 1, link
259:            Do j = 1, link
260:              hessian(i, j) = hessian(i, j) + theta*(xijrs(i)*xijrs(j)/yde(ioi)- &
261:                xijgh(i,j))
262:            End Do
263:          End Do
264:  !       write (*,*) 'finish_xijgh'
265:        End Do
266:
267:        Deallocate (pjsgh)
268:  !************************************************************
269:        Call cpu_time(v)
270:        Write (19, *) 'Calculation time 1 (second)', v
271:
272:  !*******Record results*****
273:        Open (25, File='25.hessian.txt')
274:        Do i = 1, link
275:          Write (25, *) hessian(i, :)
276:        End Do
277:        Write (*, *) 'finish_hessian'
278:
279:  !*******Calculation Gradient**********
280:        gradt = 0.0
281:        gradt2 = 0.0
282:        Do i = 1, link
283:          Do j = 1, link
284:            If (i==j) Then
285:              gradt(i, j) = xt(i) - hessian(i, j)
286:            Else
287:              gradt(i, j) = -hessian(i, j)
288:            End If
289:          End Do
290:        End Do
291:
292:        Do i = 1, link
293:          Do j = 1, link
294:            gradt2(i, j) = gradt(i, j)
295:          End Do
296:        End Do
297:        Do i = 1, link
298:          Do j = 1, link
299:            gradt2(i, j+link) = 0.0
300:            gradt2(i, i+link) = 1.0
301:          End Do
302:        End Do
303:
304:        Do k = 1, link
305:          pivot = gradt2(k, k)
306:          If (pivot==0) Then
307:            i = k + 1
308:            Do While (pivot==0 .And. i<link)
309:              pivot = gradt2(i, k)
```

```
310:            i = i + 1
311:          End Do
312:          If (pivot==0) Then
313:            Stop
314:          Else
315:            Do j = 1, 2*link
316:              www = gradt2(k, j)
317:              i = i - 1
318:              gradt2(k, j) = gradt2(i, j)
319:              gradt2(i, j) = www
320:            End Do
321:          End If
322:        End If
323:        Do j = 1, link*2
324:          gradt2(k, j) = gradt2(k, j)/pivot
325:        End Do
326:
327:        Do i = 1, link
328:          If (i/=k) Then
329:            www = gradt2(i, k)
330:            Do j = 1, link*2
331:              gradt2(i, j) = gradt2(i, j) - www*gradt2(k, j)
332:            End Do
333:          End If
334:        End Do
335:      End Do
336:
337:      gradt = 0.0
338:      Do i = 1, link
339:        Do j = 1, link
340:          gradt(i, j) = -gradt2(i, j+link)*xzeta(j)
341:        End Do
342:      End Do
343:
344:      gradzeta = 0.0
345:      Do i = 1, link
346:        Do j = 1, link
347:          If (i==j) Then
348:            gradzeta(i, j) = xzeta(i) + xt(i)*gradt(i, j)
349:          Else
350:            gradzeta(i, j) = xt(i)*gradt(i, j)
351:          End If
352:        End Do
353:      End Do
354:      Open (26, File='26.Gradzeta.txt')
355:      Do i = 1, link
356:        Write (26, *) gradzeta(i, :)
357:      End Do
358:      gradxi1 = 0.0
359:      Do i = 1, link
360:        Do j = 1, link
361:          Do k = 1, nod
```

```
362:              gradxi1(i, k) = gradxi1(i, k) - gradt2(i, j+link)*gradq(j, k)
363:            End Do
364:          End Do
365:        End Do
366:
367:        gradxi = 0.0
368:        Do i = 1, link
369:          Do j = 1, nod
370:            gradxi(i, j) = xt(i)*gradxi1(i, j)
371:          End Do
372:        End Do
373:        Open (27, File='27.Gradxi.txt')
374:        Do i = 1, link
375:          Write (27, *) gradxi(i, :)
376:        End Do
377:        Write (6, *) 'finish_gradient'
378:  !c******Calculate link travel flow after Sensitivity analysis******************
379:        zeta = 0
380:        Do i = 1, link
381:          zeta(i) = 1
382:        End Do
383:        xafter = matmul(gradzeta, zeta)
384:
385:        Do i = 1, link
386:          xafter(i) = xo(i) + xafter(i)
387:        End Do
388:
389:        Open (35, File='35.xafter.txt')
390:        Do i = 1, link
391:          Write (35, *) xafter(i)
392:        End Do
393:
394:        Call cpu_time(v)
395:        Write (19, *) 'time of calculation', v
396:
397:  !*****SUBROUTINE************************************************************
398:      Contains
399:        Subroutine dijikstra(p, t)
400:          Real, Intent (In) :: t(link)
401:          Real *8, Intent (Out) :: p(node, node)
402:          Real *8 :: c(node), cmin(node), fin_io(node)
403:
404:          Do j = 1, node
405:            io = j !io is the source node number
406:            c(:) = 10000000.0 !set partial path cost • ‡
407:            c(io) = 0.0 !set the cost of the source node to 0
408:            cmin(:) = 10000000.0
409:            fin_io(:) = 0 !array for storing the nodes for which calculation has been completed
410:            Do While (minval(cmin,1)<50000000)
411:              Do iii = headnode(io-1) + 1, headnode(io)
412:                i = ls(io, anode(iii))
413:                If (c(io)+t(i)<=c(e(i))) Then
```

```
414:                c(e(i)) = c(io) + t(i) !update with the next link cost with the node
415:                cmin(e(i)) = c(io) + t(i)
416:              End If
417:            End Do
418:            ncj = minloc(cmin, 1) !identify the location of cmin in the array
419:            cmin(ncj) = 50000000.0 !ineffectiveness of extracted cmin
420:  !**** calculation end judgment (end when all cmin are disabled) ***********
421:            If (minval(cmin,1)==50000000) Then
422:              Go To 100
423:            End If
424:  !******!move to the next node******************
425:            io = ncj
426:            If (fin_io(ncj)==1) Then !when calculation is completed, move to another node
427:              inot = minloc(fin_io, 1)
428:              io = inot
429:            End If
430:            fin_io(io) = 1
431:          End Do
432:  100       Continue
433:  !******record shortest path results************************************
434:          p(j, :) = c(:)
435:        End Do
436:
437:      End Subroutine
438:  !*****FINISH SUBROUTINE***********************************
439:    End Program
```

E.5 New sensitivity analysis formulation for SUE problem combines with old calculation process based on STOCH3 algorithm

```
!Sensitivity analysis for SUE with STOCH3 algorithm
!The new approach in formulations + The method of Ying and Miyagi to calculate xijgh*
    Integer i, j, k, ii, ioi, io, count
    Integer, Parameter :: node = 6 !number of nodes
    Integer, Parameter :: link = 8 !number of links
    Integer, Parameter :: nod = 2 !number of OD pairs
    Real *8, Parameter :: theta = 1.0D0 !theta parameter
    Real *8, Parameter :: alpha = 1.0D0 !BPR parameter
    Real *8, Parameter :: beta = 2.0D0 !BPR parameter
    Integer s(link), e(link), ls(node, node) !start and end node and link number
    Integer ori(nod), des(nod) !origin, destination of od pairs
    Real *8 yde(nod) !od demand of od pairs
    Real t(link), cap(link) !t:free-flow travel time,cap: capacity of link
    Real *8 erh(link), p(node, node) !erh: stoch3 parameter, p:shortest route
    Integer headnode(0:node), anode(link) ! adjacent nodes
    Real *8 ps(link), pe(link)
!ps,pe: shortest path from each origin to start and end node of the link
    Integer ss(link), ee(link)
!ss: Start node,ee: end node in arranged shortest path
    Real *8 a(link), wl(link), wn(node)
!a:link likelihood,w:link weight,wn:node weight
    Integer omega(nod, link) !Used to check STOCH3-efficient path
    Real *8 xo(link) !xo: link traffic flow at SUE
    Real *8 tt(link) !tt: link travel time at SUE
    Real *8 x(link), xx(node)
!x: link travel is assigned, xx:total node travel in STOCH3's Algorithm
    Real *8 xijrs(link)
    Real *8 xijgh1(link, link), xijgh(link, link) ! Used to calculate xijgh
    Real *8 tx(link) !tx:derivative of t respect to x
    Real *8 gradientgt(link, link)
    Real *8 part1(link, link), gradx(link, link)
    Real *8 www, pivot, part2(link, 2*link), gradzeta(link, link)
    Real *8 gradxi(link, nod), gradgq(link, nod)
    Real *8 zeta(link), xi(nod)
    Real *8 xafter(link) !link travel flow after sensitivity analysis
    Integer mark(node, node), omega1(link)
    Real *8 tzeta(link)
    Real *8, Allocatable :: pjsgh(:, :, :)
!*****************************************************

    Open (1, File='1.network.txt', Action='read')
    Open (2, File='2.linkparameter.txt', Action='read')
    Open (3, File='3.demand.txt', Action='read')
    Open (11, File='19.CalculationTime(SA).txt', Action='write')
    Do i = 1, link
      Read (1, *) s(i), e(i), j !s(i):start node, e(i):end node
      Read (2, *) t(i), cap(i) !t(i):free flow traveltime, cap(i):Capacity of link i
      erh(i) = 1.5
    End Do
```

```
50:         Do i = 1, nod
51:           Read (3, *) ori(i), des(i), yde(i)
52:         End Do
53:     !make network
54:         ls = 0.0
55:         Do i = 1, link
56:           ls(s(i), e(i)) = i
57:         End Do
58:     ! create adjacent nodes and links
59:         headnode(0) = 0
60:         count = 0
61:         Do i = 1, node
62:           Do j = 1, link
63:             If (i==s(j)) Then
64:               count = count + 1
65:               anode(count) = e(j)
66:     !          alink(count) = j
67:             End If
68:           End Do
69:           headnode(i) = count
70:         End Do
71:
72:         Write (6, *) 'theta', theta
73:         Write (6, *) 'alpha', alpha
74:         Write (6, *) 'beta', beta
75:
76:         Open (12, File='12.xij.txt')
77:         Do i = 1, link
78:           Read (12, *) xo(i)
79:         End Do
80:         Do i = 1, link
81:           tt(i) = t(i)*(1+alpha*((xo(i)/cap(i))**beta)) !BPR calculation
82:         End Do
83:     !***** Calculating derivitives of implicit BPR function*****************
84:         Do i = 1, link
85:           tx(i) = t(i)*beta*alpha*(xo(i)**(beta-1))/(cap(i)**beta)
86:           tzeta(i) = 1 + alpha*xo(i)**beta/(cap(i)**beta)
87:         End Do
88:         Open (21, File='21.tx.txt')
89:         Do i = 1, link
90:           Write (21, *) tx(i)
91:         End Do
92:         Write (*, *) 'finish_tx'
93:         Open (23, File='23.pjsgh.txt')
94:         omega = 0
95:         gradientgt = 0.0
96:         xijrs = 0
97:         Allocate (pjsgh(node,node,link))
98:         gradgq = 0.0
99:         pjsgh = 0
100:
101:    !******** Dijkstra method calculation ****************
```

```
102:        Call dijikstra(p, t)
103:
104:   !************************
105:        mark = 0
106:        a = 0.0
107:        Do i = 1, link
108:          a(i) = exp(-theta*(tt(i)))
109:        End Do
110:   !********STOCH3 algorithm with each OD pair*****************************
111:        Do ioi = 1, nod
112:
113:          io = ori(ioi)
114:          xx = 0.0
115:          xx(des(ioi)) = yde(ioi)
116:
117:          Do i = 1, link
118:            ps(i) = p(io, s(i))
119:            pe(i) = p(io, e(i))
120:          End Do
121:          Do i = 1, link
122:            If (pe(i)>ps(i)) Then
123:   !          If ((1+erh(i))*(pe(i)-ps(i))>=t(i)) Then
124:              omega(ioi, i) = 1
125:   !          End If
126:            End If
127:          End Do
128:
129:   !*****sort shortest path from current node*****************************
130:          Do i = 1, link
131:            k = minloc(pe, 1)
132:            pe(k) = 10000001.0
133:            ss(i) = s(k)
134:            ee(i) = e(k)
135:          End Do
136:   !***** Calculation of link weight and node weight*********************
137:          wl = 0.0
138:          wn = 0.0
139:          Do i = 1, link !Ps(i)Calculate in ascending sequence
140:            nn = ls(ss(i), ee(i)) !NN is the link number
141:            wn(io) = 1.0
142:            If (omega(ioi,nn)==1) Then
143:              wl(nn) = wn(ss(i))*a(nn) !Link weight
144:              wn(ee(i)) = wn(ee(i)) + wl(nn) !Node weight
145:            End If
146:          End Do
147:   !******** Assign trip volume************
148:          x = 0.0
149:          Do i = 1, link
150:            ii = link + 1 - i !Ps(i)Calculate in descending order
151:            nn = ls(ss(ii), ee(ii)) !NN is the link number
152:            If (wn(ee(ii))==0) Go To 100 !Prevents division by 0
153:            If (omega(ioi,nn)==1) Then
```

```
154:              x(nn) = xx(ee(ii))*wl(nn)/wn(ee(ii))
155:              xx(ss(ii)) = xx(ss(ii)) + x(nn)
156:            End If
157:  100     End Do
158:  !      write (*,*) 'finish_xijrs'
159:  !************************************************************
160:          xijrs = x
161:          Do i = 1, link
162:            gradgq(i, ioi) = gradgq(i, ioi) + x(i)/yde(ioi)
163:          End Do
164:  !
165:  !************
166:          Do kk = 1, link
167:            If (xijrs(kk)/=0) Then
168:              io = e(kk)
169:              If (mark(des(ioi),io)==1) Then
170:                Go To 120
171:              End If
172:              mark(des(ioi), io) = 1
173:
174:              xx = 0.0
175:              xx(des(ioi)) = 1.0
176:
177:              Do i = 1, link
178:                ps(i) = p(io, s(i))
179:                pe(i) = p(io, e(i))
180:              End Do
181:              omega1 = 0
182:              Do i = 1, link
183:                If (pe(i)>ps(i)) Then
184:  !               If ((1+erh(i))*(pe(i)-ps(i))>=t(i)) Then
185:                  omega1(i) = 1
186:  !               End If
187:                End If
188:              End Do
189:              Do i = 1, link
190:                k = minloc(pe, 1)
191:                pe(k) = 10000001.0
192:                ss(i) = s(k)
193:                ee(i) = e(k)
194:              End Do
195:
196:  !***** Calculation of link weight and node weight*********************
197:              wl = 0.0
198:              wn = 0.0
199:              Do i = 1, link !Ps(i)Calculate in ascending sequence
200:                nn = ls(ss(i), ee(i)) !NN is the link number
201:                wn(io) = 1.0
202:                If (omega1(nn)==1) Then
203:                  wl(nn) = wn(ss(i))*a(nn) !Link weight
204:                  wn(ee(i)) = wn(ee(i)) + wl(nn) !Node weight
205:                End If
```

```
206:            End Do
207:  !******** Assign trip volume************
208:            x = 0.0
209:            Do i = 1, link
210:              ii = link + 1 - i !Ps(i)Calculate in descending order
211:              nn = ls(ss(ii), ee(ii)) !NN is the link number
212:              If (wn(ee(ii))==0) Go To 110 !Prevents division by 0
213:              If (omega1(nn)==1) Then
214:                x(nn) = xx(ee(ii))*wl(nn)/wn(ee(ii))
215:                xx(ss(ii)) = xx(ss(ii)) + x(nn)
216:              End If
217:  110         End Do
218:  !**************************
219:
220:            Do i = 1, link
221:              If (x(i)/=0) Then
222:                pjsgh(des(ioi), io, i) = x(i)
223:                Write (23, *) e(kk), des(ioi), i, x(i)
224:              End If
225:            End Do
226:
227:          End If
228:  120     End Do
229:
230:         xijgh1 = 0.0
231:         Do i = 1, link
232:           Do j = 1, link
233:             xijgh1(i, j) = xijrs(i)*pjsgh(des(ioi), e(i), j)
234:           End Do
235:         End Do
236:         xijgh = 0.0
237:         Do i = 1, link
238:           Do j = 1, link
239:             xijgh(i, j) = xijgh1(i, j) + xijgh1(j, i)
240:           End Do
241:           xijgh(i, i) = xijrs(i)
242:         End Do
243:
244:         Do i = 1, link
245:           Do j = 1, link
246:             gradientgt(i, j) = gradientgt(i, j) + theta*(xijrs(i)*xijrs(j)/yde &
247:               (ioi)-xijgh(i,j))
248:           End Do
249:         End Do
250:  !      write (*,*) 'finish_xijgh'
251:       End Do
252:
253:       Deallocate (pjsgh)
254:  !************************************************************
255:       Call cpu_time(v)
256:       Write (11, *) 'Calculation time 1 (second)', v
257:
```

```
!*******Record results*****
      Open (24, File='22.gradientgt.txt')
      Do i = 1, link
        Write (24, *) gradientgt(i, :)
      End Do
      Write (*, *) 'finish_gradientgt'

!*******Calculation Gradient**********
      gradx = 0.0
      part1 = 0.0
      part2 = 0.0
      Do i = 1, link
        Do j = 1, link
          gradx(i, j) = gradientgt(i, j)*tx(j)
        End Do
      End Do

      Do i = 1, link
        Do j = 1, link
          If (i==j) Then
            part1(i, j) = 1 - gradx(i, j)
          Else
            part1(i, j) = -gradx(i, j)
          End If
        End Do
      End Do
      Do i = 1, link
        Do j = 1, link
          part2(i, j) = part1(i, j)
        End Do
      End Do

      Do i = 1, link
        Do j = 1, link
          part2(i, j+link) = 0
          part2(i, i+link) = 1
        End Do
      End Do

      Do k = 1, link
        pivot = part2(k, k)
        If (pivot==0) Then
          i = k + 1
          Do While (pivot==0 .And. i<link)
            pivot = part2(i, k)
            i = i + 1
          End Do
          If (pivot==0) Then
            Stop
          Else
            Do j = 1, 2*link
              www = part2(k, j)
```

```
310:              i = i - 1
311:              part2(k, j) = part2(i, j)
312:              part2(i, j) = www
313:            End Do
314:          End If
315:        End If
316:        Do j = 1, link*2
317:          part2(k, j) = part2(k, j)/pivot
318:        End Do
319:
320:        Do i = 1, link
321:          If (i/=k) Then
322:            www = part2(i, k)
323:            Do j = 1, link*2
324:              part2(i, j) = part2(i, j) - www*part2(k, j)
325:            End Do
326:          End If
327:        End Do
328:      End Do
329:
330:      gradx = 0.0
331:      Do i = 1, link
332:        Do j = 1, link
333:          gradx(i, j) = part2(i, j+link)
334:        End Do
335:      End Do
336:
337:      part1 = 0.0
338:      Do i = 1, link
339:        Do j = 1, link
340:          part1(i, j) = gradientgt(i, j)*tzeta(j)
341:        End Do
342:      End Do
343:      gradzeta = 0.0
344:      gradzeta = matmul(gradx, part1)
345:      gradxi = 0.0
346:      gradxi = matmul(gradx, gradgq)
347:      Open (27, File='27.Gradient(zeta).txt')
348:      Do i = 1, link
349:        Write (27, *)(gradzeta(i,j), j=1, link)
350:      End Do
351:      Open (28, File='28.Gradient(xi).txt')
352:      Do i = 1, link
353:        Write (28, *)(gradxi(i,j), j=1, nod)
354:      End Do
355:      Write (*, *) 'finish_Gradient'
356:
357:  !******Calculate link travel flow with Sensitivity analysis*****************
358:  !****Results with the change of zeta*********
359:      zeta = 0.0D0
360:  !     xi = 0.0D0
361:
```

```
362:       Open (35, File='35.xafter(zeta).txt')
363: !     Open (36, File='36.xafter(xi).txt')
364: !     Open (58, File='58.Objective(zeta).txt')
365: !     Open (59, File='59.Objective(xi).txt')
366:
367:       zeta = 0.0D0
368:       xi = 0.0D0
369: !     Do ii = 1, 10
370: !       Write (6, *) 'round', ii
371: !       If (ii<=5) Then
372: !         zeta = -t*ii/100.0D0
373:       zeta = 1.0D0
374:       xafter = matmul(gradzeta, zeta)
375: !     obj = 0.0D0
376:       Do i = 1, link
377:         xafter(i) = xo(i) + xafter(i)
378: !       tafter(i) = (t(i)+zeta(i))*(1+alpha*((xafter(i)/cap(i))**beta))
379: !           obj = obj + xafter(i)*tafter(i)
380:       End Do
381:
382:       Do i = 1, link
383:         Write (35, *) xafter(i)
384:       End Do
385: !         Write (58, *) obj
386: !   Else
387: !!****Results with the change of xi*********
388: !         xi = -(ii-5.0)*yde/100.0D0
389: !         xafter = matmul(gradxi, xi)
390: !         obj = 0.0D0
391: !         Do i = 1, link
392: !           xafter(i) = xo(i) + xafter(i)
393: !           tafter(i) = t(i)*(1+alpha*((xafter(i)/cap(i))**beta))
394: !           obj = obj + xafter(i)*tafter(i)
395: !         End Do
396: !
397: !
398: !         Do i = 1, link
399: !           Write (36, *) xafter(i)
400: !         End Do
401: !         Write (59, *) obj
402: !       End If
403: !     End Do
404:       Call cpu_time(v)
405:       Write (11, *) 'time 2 (sec)', v
406:
407: !*****SUBROUTINE************************************************************
408:     Contains
409:       Subroutine dijikstra(p, t)
410:         Real, Intent (In) :: t(link)
411:         Real *8, Intent (Out) :: p(node, node)
412:         Real *8 :: c(node), cmin(node), fin_io(node)
413:
```

```
414:          Do j = 1, node
415:            io = j !io is the source node number
416:            c(:) = 10000000.0 !set partial path cost • ‡
417:            c(io) = 0.0 !set the cost of the source node to 0
418:            cmin(:) = 10000000.0
419:            fin_io(:) = 0 !array for storing the nodes for which calculation has been completed
420:            Do While (minval(cmin,1)<50000000)
421:              Do iii = headnode(io-1) + 1, headnode(io)
422:                i = ls(io, anode(iii))
423:                If (c(io)+t(i)<=c(e(i))) Then
424:                  c(e(i)) = c(io) + t(i) !update with the next link cost with the node
425:                  cmin(e(i)) = c(io) + t(i)
426:                End If
427:              End Do
428:              ncj = minloc(cmin, 1) !identify the location of cmin in the array
429:              cmin(ncj) = 50000000.0 !ineffectiveness of extracted cmin
430:   !**** calculation end judgment (end when all cmin are disabled) ***********
431:              If (minval(cmin,1)==50000000) Then
432:                Go To 100
433:              End If
434:   !******!move to the next node******************
435:              io = ncj
436:              If (fin_io(ncj)==1) Then !when calculation is completed, move to another node
437:                inot = minloc(fin_io, 1)
438:                io = inot
439:              End If
440:              fin_io(io) = 1
441:            End Do
442:   100       Continue
443:   !******record shortest path results***********************************
444:            p(j, :) = c(:)
445:          End Do
446:
447:        End Subroutine
448:   !*****FINISH SUBROUTINE**********************************
449:      End Program
```

E.6 DFS algorithm with the MSA for solving logit-based SUE problem

```
!***Stochastic User Equilibrium with DFS algorithm and stoch3-efficient route***
      Integer l, i, j, k, ioi, io, count
      Integer, Parameter :: node = 6 !number of nodes
      Integer, Parameter :: link = 8 !number of links
      Integer, Parameter :: nod = 2 !number of OD pairs in time period 1
      Real *8, Parameter :: eee = 1.0D-6 !loop termination condition
      Real *8, Parameter :: theta = 1.0D0 !theta parameter
      Real *8, Parameter :: alpha = 1.0D0 !BPR parameter
      Real *8, Parameter :: beta = 2.0D0 !BPR parameter
      Integer s(link), e(link), ls(node, node) !start and end node and link number
      Integer ori(nod), des(nod) !origin, destination of od pairs
      Real *8 yde(nod) !od demand of od pairs
      Real t(link), cap(link) !t:free-flow travel time,cap: capacity of link
      Real *8 erh(link), p(node, node) !erh: stoch3 parameter, p:shortest route
      Integer headnode(0:node), anode(link), alink(link) ! adjacent nodes and links
      Real *8 ps(link), pe(link)
!ps,pe: shortest path from each origin to start and end node of the link
      Real *8 a(link) !a:link likelihood
      Integer omega(nod, link), mark(link)
!omega(nod,link): marking efficient links of each od pair
!mark(link): marking a route at each time reaching the destination in dfs algorithm
      Real *8 xo(link), xn(link) !xo: link traffic flow after, xn:link traffic flow before
      Real *8 tt(link) !tt: link travel time after update
      Real *8 maxrg, g(link) !used in msa method
      Real *8 x(link) !x: link travel is assigned
      Real *8 wl(link), xnn, wn(node)
!*******************************************************

      Open (1, File='1.network.txt', Action='read')
      Open (2, File='2.linkparameter.txt', Action='read')
      Open (3, File='3.demand.txt', Action='read')
      Open (11, File='11.CalculationTime(SUE).txt', Action='write')
      Do i = 1, link
        Read (1, *) s(i), e(i), j !s(i):start node, e(i):end node
        Read (2, *) t(i), cap(i) !t(i):free flow traveltime, cap(i):Capacity of link i
        erh(i) = 1.5
      End Do
      Do i = 1, nod
        Read (3, *) ori(i), des(i), yde(i)
      End Do
!make network
      ls = 0.0
      Do i = 1, link
        ls(s(i), e(i)) = i
      End Do
! create adjacent nodes and links
      headnode(0) = 0
      count = 0
      Do i = 1, node
        Do j = 1, link
          If (i==s(j)) Then
```

```
52:              count = count + 1
53:              anode(count) = e(j)
54:              alink(count) = j
55:            End If
56:          End Do
57:          headnode(i) = count
58:        End Do
59:
60:        Write (6, *) 'theta', theta
61:        Write (6, *) 'alpha', alpha
62:        Write (6, *) 'beta', beta
63:    !********* dijkstra method calculation ****************
64:        Call dijikstra(p, t)
65:
66:    !****** dfs algorithm ******************************
67:        xo = 0
68:        omega = 0.0
69:    !***** Creating initial solution based on free-flow travel time***********
70:        a = 0.0
71:        tt = 0.0
72:        Do i = 1, link
73:          tt(i) = t(i)
74:          a(i) = exp(-theta*tt(i))
75:        End Do
76:        Do ioi = 1, nod
77:          io = ori(ioi)
78:          Do i = 1, link
79:            ps(i) = p(io, s(i))
80:            pe(i) = p(io, e(i))
81:          End Do
82:
83:    !****** !check STOCH3-efficient routes ******************************
84:          Do i = 1, link
85:            If (ps(i)>pe(i)) Go To 100
86:    !         If ((1+erh(i))*(pe(i)-ps(i))>=t(i)) Then
87:            omega(ioi, i) = 1
88:    !         End If
89:    100     End Do
90:    !******running the dfs algorithm from origin node r to destination node s ******
91:          wl = 0.0
92:          wn = 0.0D0
93:          wn(ori(ioi)) = 1
94:          j = ori(ioi)
95:          k = 0
96:          xnn = 0
97:          Call calxij(1)
98:    !************************************************************
99:    ! recording after loop termination
100:         Do i = 1, link
101:           omega(ioi, i) = 0
102:           x(i) = yde(ioi)*wl(i)/xnn
103:           If (x(i)/=0) Then
```

```
104:            omega(ioi, i) = 1
105:          End If
106:          xo(i) = xo(i) + x(i)
107:        End Do
108:      End Do
109:  !*******starting calculation round************************************
110:      xn = 0.0
111:      g = 0.0
112:      maxrg = 1
113:      l = 0
114:      mark = 0
115:      Do While (maxrg>eee)
116:  !*****Link travel time and link impedances update**********************
117:        l = l + 1
118:        a = 0.0D0
119:        Do i = 1, link
120:          tt(i) = t(i)*(1+alpha*((xo(i)/cap(i))**beta)) !BPR calculation
121:          a(i) = exp(-theta*(tt(i)))
122:        End Do
123:  !***** STOCH 3 algorithm ******************************
124:        xn = 0.0
125:        Do ioi = 1, nod
126:          wl = 0.0
127:          wn = 0.0D0
128:          wn(ori(ioi)) = 1
129:          j = ori(ioi)
130:          k = 0
131:          xnn = 0
132:          Call calxij2(1)
133:  !**************************
134:          Do i = 1, link
135:            xn(i) = xn(i) + yde(ioi)*wl(i)/xnn
136:          End Do
137:  !************************************
138:        End Do
139:  !************************************
140:  !c Convergence determination calculation
141:        g = 0.0
142:        Do i = 1, link
143:          g(i) = xn(i) - xo(i)
144:          xn(i) = xo(i) + g(i)*(1.0D0/(1.0D0+l))
145:        End Do
146:        xo = xn
147:        maxrg = maxval(abs(g))
148:  !       Write (*, *) 'round', l, 'maxrg', maxrg
149:      End Do
150:  !**********Results************************
151:      Write (11, *) 'Round', l, 'Computation gap', maxrg
152:      Call cpu_time(v)
153:      Write (11, *) 'time of calculation (second) ', v
154:      Open (12, File='12.xij.txt')
155:      Do i = 1, link
```

```
156:          Write (12, *) xo(i)
157:        End Do
158:        Do i = 1, link
159:          tt(i) = t(i)*(1+alpha*((xo(i)/cap(i))**beta)) !BPR calculation
160:        End Do
161:        Open (13, File='13.tij.txt')
162:        Do i = 1, link
163:          Write (13, *) tt(i)
164:        End Do
165:
166:  !*****SUBROUTINE*************************************************************
167:      Contains
168:        Subroutine dijikstra(p, t)
169:          Real, Intent (In) :: t(link)
170:          Real *8, Intent (Out) :: p(node, node)
171:          Real *8 :: c(node), cmin(node), fin_io(node)
172:
173:          Do j = 1, node
174:            io = j !io is the source node number
175:            c(:) = 10000000.0 !set partial path cost • ‡
176:            c(io) = 0.0 !set the cost of the source node to 0
177:            cmin(:) = 10000000.0
178:            fin_io(:) = 0 !array for storing the nodes for which calculation has been completed
179:            Do While (minval(cmin,1)<50000000)
180:              Do iii = headnode(io-1) + 1, headnode(io)
181:                i = ls(io, anode(iii))
182:                If (c(io)+t(i)<=c(e(i))) Then
183:                  c(e(i)) = c(io) + t(i) !update with the next link cost with the node
184:                  cmin(e(i)) = c(io) + t(i)
185:                End If
186:              End Do
187:              ncj = minloc(cmin, 1) !identify the location of cmin in the array
188:              cmin(ncj) = 50000000.0 !ineffectiveness of extracted cmin
189:  !**** calculation end judgment (end when all cmin are disabled) ***********
190:              If (minval(cmin,1)==50000000) Then
191:                Go To 100
192:              End If
193:  !******!move to the next node*****************
194:              io = ncj
195:              If (fin_io(ncj)==1) Then !when calculation is completed, move to another node
196:                inot = minloc(fin_io, 1)
197:                io = inot
198:              End If
199:              fin_io(io) = 1
200:            End Do
201:  100       Continue
202:  !******record shortest path results***********************************
203:            p(j, :) = c(:)
204:          End Do
205:        End Subroutine
206:
207:  !     First Calculate xij with DFS
```

```
      Recursive Subroutine calxij(dfs)
        Integer dfs, iii

        If (j==des(ioi)) Then
          xnn = xnn + wn(j)
          Do i = 1, k
            wl(mark(i)) = wl(mark(i)) + wn(j)
          End Do
        Else
          Do iii = headnode(j-1) + 1, headnode(j)
            If (omega(ioi,alink(iii))/=0) Then
              j = e(alink(iii))
              k = k + 1
              wn(j) = a(alink(iii))*wn(s(alink(iii)))
              mark(k) = alink(iii)
              Call calxij(dfs+1)
              mm = j
              If (mm/=des(ioi) .And. wl(mark(k))==0) Then
                omega(ioi, mark(k)) = 0
              End If
              mark(k) = 0
              k = k - 1
            End If
          End Do
        End If
      End Subroutine
!     Repeatedly calculate xij with DFS
      Recursive Subroutine calxij2(dfs)
        Integer dfs, iii

        If (j==des(ioi)) Then
          xnn = xnn + wn(j)
          Do i = 1, k
            wl(mark(i)) = wl(mark(i)) + wn(j)
          End Do
        Else
          Do iii = headnode(j-1) + 1, headnode(j)
            If (omega(ioi,alink(iii))/=0) Then
              j = e(alink(iii))
              k = k + 1
              wn(j) = a(alink(iii))*wn(s(alink(iii)))
              mark(k) = alink(iii)
              Call calxij2(dfs+1)
              mark(k) = 0
              k = k - 1
            End If
          End Do
        End If
      End Subroutine

    End Program
```

E.7 New sensitivity analysis method for SUE problem based on DFS algorithm

```
1:    ! Sensitivity analysis for SUE with the new formulations and calculation process
2:    ! ***changing parameters of free-flow travel time and travel demand***
3:        Integer i, j, k, ioi, io, count, ii
4:        Integer, Parameter :: node = 6 !number of nodes
5:        Integer, Parameter :: link = 8 !number of links
6:        Integer, Parameter :: nod = 2 !number of OD pairs
7:        Real *8, Parameter :: theta = 1.0D0 !theta parameter
8:        Real *8, Parameter :: alpha = 1.0D0 !BPR parameter
9:        Real *8, Parameter :: beta = 2.0D0 !BPR parameter
10:       Integer s(link), e(link), ls(node, node) !start and end node and link number
11:       Integer ori(nod), des(nod) !origin, destination of od pairs
12:       Real *8 yde(nod) !od demand of od pairs
13:       Real t(link), cap(link) !t:free-flow travel time,cap: capacity of link
14:       Real *8 erh(link), p(node, node) !erh: stoch3 parameter, p:shortest route
15:       Integer headnode(0:node), anode(link), alink(link) ! adjacent nodes and links
16:       Real *8 ps(link), pe(link)
17:   !ps,pe: shortest path from each origin to start and end node of the link
18:       Real *8 a(link) !a:link likelihood
19:       Integer omega(nod, link), mark(link)
20:   !omega(nod,link): marking efficient links of each od pair
21:   !mark(link): marking a route at each time reaching the destination in dfs algorithm
22:       Real *8 xo(link), tt(link) !link traffic flow and travel time at STA SUE
23:       Real *8 dijgh(link, link), xijrs(link)
24:       Real *8 tx(link) !tx:derivative of t respect to x
25:       Real *8 gradientgt(link, link) !used to calculate derivative of g respect to t
26:       Real *8 part1(link, link) ! used to calculate inverse matrix
27:       Real *8 www, pivot, part2(link, 2*link) ! used to calculate inverse matrix
28:       Real *8 gradx(link, link), gradzeta(link, link)
29:       Real *8 zeta(link), xi(nod), gradxi(link, nod), gradq(link, nod)
30:   !zeta: zeta parameter of free-flow travel time
31:   !grad: gradient matrix
32:   !xi: xi paramter of travel demand
33:       Real *8 xafter(link), tafter(link) !link travel flow and time after SA
34:       Real *8 tzeta(link)
35:       Real *8 wl(link), swn, wn(node)
36:   !*******************************************************
37:
38:       Open (1, File='1.network.txt', Action='read')
39:       Open (2, File='2.linkparameter.txt', Action='read')
40:       Open (3, File='3.demand.txt', Action='read')
41:       Open (11, File='19.CalculationTime(SA).txt', Action='write')
42:       Open (20, File='20.xijrs.txt', Action='write')
43:       Open (23, File='23.pijrs.txt', Action='write')
44:       Do i = 1, link
45:         Read (1, *) s(i), e(i), j !s(i):start node, e(i):end node
46:         Read (2, *) t(i), cap(i) !t(i):free flow traveltime, cap(i):Capacity of link i
47:         erh(i) = 1.5
48:       End Do
49:       Do i = 1, nod
50:         Read (3, *) ori(i), des(i), yde(i)
51:       End Do
```

```
52:   !make network
53:         ls = 0.0
54:         Do i = 1, link
55:           ls(s(i), e(i)) = i
56:         End Do
57:   ! create adjacent nodes and links
58:         headnode(0) = 0
59:         count = 0
60:         Do i = 1, node
61:           Do j = 1, link
62:             If (i==s(j)) Then
63:               count = count + 1
64:               anode(count) = e(j)
65:               alink(count) = j
66:             End If
67:           End Do
68:           headnode(i) = count
69:         End Do
70:
71:         Write (6, *) 'theta', theta
72:         Write (6, *) 'alpha', alpha
73:         Write (6, *) 'beta', beta
74:
75:         Open (12, File='12.xij.txt')
76:         Do i = 1, link
77:           Read (12, *) xo(i)
78:         End Do
79:         a = 0.0D0
80:         Do i = 1, link
81:           tt(i) = t(i)*(1+alpha*((xo(i)/cap(i))**beta)) !BPR calculation
82:           a(i) = exp(-theta*tt(i))
83:         End Do
84:   !Step 2: Calculating gradient u of t and gradient u of q
85:   !***** Calculating derivitives of explicit BPR function*****************
86:         Do i = 1, link
87:           tx(i) = t(i)*beta*alpha*(xo(i)**(beta-1))/(cap(i)**beta)
88:           tzeta(i) = 1 + alpha*xo(i)**beta/(cap(i)**beta)
89:         End Do
90:         Open (21, File='21.tx.txt')
91:         Do i = 1, link
92:           Write (21, *) tx(i)
93:         End Do
94:         Write (*, *) 'finish_tx'
95:
96:         omega = 0
97:         gradientgt = 0.0
98:         gradq = 0.0
99:   !*****Calculate xijrs,node traffic flow, link proportion, dijgh and gradientgt*********
100:  !*****Calculate xijrs**************
101:  !********* Dijkstra method calculation *****************
102:        Call dijikstra(p, t)
103:
```

```
104:        Do ioi = 1, nod
105:          xijrs = 0.0
106:          io = ori(ioi)
107:
108:          Do i = 1, link
109:            ps(i) = p(io, s(i))
110:            pe(i) = p(io, e(i))
111:          End Do
112:  !********* !Checking efficient path ****************
113:          Do i = 1, link
114:            If (ps(i)>pe(i)) Go To 100
115:  !          If ((1+erh(i))*(pe(i)-ps(i))>=t(i)) Then
116:            omega(ioi, i) = 1
117:  !          End If
118:  100     End Do
119:  !********* Assign trip volume************
120:          wl = 0.0D0
121:          wn = 0.0D0
122:          j = ori(ioi)
123:          wn(j) = 1.0D0
124:          k = 0
125:          swn = 0.0D0
126:          dijgh = 0.0D0
127:          Call calxij(1)
128:  !*************************************************************
129:  !      write (*,*) 'finish_xijrs'
130:  !*************************************************************
131:
132:  !*******Calculate node traffic flow, link proportion, dijgh and gradientgt*****
133:          Do i = 1, link
134:            If (omega(ioi,i)/=0) Then
135:              gradq(i, ioi) = wl(i)/swn
136:              xijrs(i) = yde(ioi)*wl(i)/swn
137:            End If
138:            Write (20, *) ioi, i, xijrs(i)
139:            Write (23, *) ioi, i, gradq(i, ioi)
140:          End Do
141:
142:          Do i = 1, link
143:            If (omega(ioi,i)/=0) Then
144:              Do j = 1, link
145:                If (omega(ioi,j)/=0) Then
146:                  If (i==j) Then
147:                    gradientgt(i, j) = gradientgt(i, j) + &
148:                      theta*(-xijrs(i)+xijrs(i)*xijrs(j)/yde(ioi))
149:                  Else
150:                    gradientgt(i, j) = gradientgt(i, j) + &
151:                      theta*(-yde(ioi)*(dijgh(i,j)+dijgh(j, &
152:                      i))/swn+xijrs(i)*xijrs(j)/yde(ioi))
153:                  End If
154:                End If
155:              End Do
```

```
156:            End If
157:          End Do
158:  !       write (*,*) 'finish_dijgh'
159:        End Do
160:        Call cpu_time(v)
161:        Write (11, *) 'time 1 (sec)', v
162:
163:  !*************************************************************
164:  !*******Record results*****
165:
166:        Open (24, File='22.gradientgt.txt')
167:        Do i = 1, link
168:          Write (24, *) gradientgt(i, :)
169:        End Do
170:        Write (*, *) 'finish_gradientgt'
171:
172:  !Step 3: Calculating Gradient matrices**********
173:        gradx = 0.0
174:        part1 = 0.0
175:        part2 = 0.0
176:        Do i = 1, link
177:          Do j = 1, link
178:            gradx(i, j) = gradientgt(i, j)*tx(j)
179:          End Do
180:        End Do
181:
182:        Do i = 1, link
183:          Do j = 1, link
184:            If (i==j) Then
185:              part1(i, j) = 1 - gradx(i, j)
186:            Else
187:              part1(i, j) = -gradx(i, j)
188:            End If
189:          End Do
190:        End Do
191:        Do i = 1, link
192:          Do j = 1, link
193:            part2(i, j) = part1(i, j)
194:          End Do
195:        End Do
196:
197:        Do i = 1, link
198:          Do j = 1, link
199:            part2(i, j+link) = 0
200:            part2(i, i+link) = 1
201:          End Do
202:        End Do
203:
204:        Do k = 1, link
205:          pivot = part2(k, k)
206:          If (pivot==0) Then
207:            i = k + 1
```

```
208:          Do While (pivot==0 .And. i<link)
209:            pivot = part2(i, k)
210:            i = i + 1
211:          End Do
212:          If (pivot==0) Then
213:            Stop
214:          Else
215:            Do j = 1, 2*link
216:              www = part2(k, j)
217:              i = i - 1
218:              part2(k, j) = part2(i, j)
219:              part2(i, j) = www
220:            End Do
221:          End If
222:        End If
223:        Do j = 1, link*2
224:          part2(k, j) = part2(k, j)/pivot
225:        End Do
226:
227:        Do i = 1, link
228:          If (i/=k) Then
229:            www = part2(i, k)
230:            Do j = 1, link*2
231:              part2(i, j) = part2(i, j) - www*part2(k, j)
232:            End Do
233:          End If
234:        End Do
235:      End Do
236:
237:      gradx = 0.0
238:      Do i = 1, link
239:        Do j = 1, link
240:          gradx(i, j) = part2(i, j+link)
241:        End Do
242:      End Do
243:
244:      part1 = 0.0
245:      Do i = 1, link
246:        Do j = 1, link
247:          part1(i, j) = gradientgt(i, j)*tzeta(j)
248:        End Do
249:      End Do
250:      gradzeta = 0.0
251:      gradzeta = matmul(gradx, part1)
252:      gradxi = 0.0
253:      gradxi = matmul(gradx, gradq)
254:      Open (27, File='27.Gradient(zeta).txt')
255:      Do i = 1, link
256:        Write (27, *)(gradzeta(i,j), j=1, link)
257:      End Do
258:      Open (28, File='28.Gradient(xi).txt')
259:      Do i = 1, link
```

```
260:          Write (28, *)(gradxi(i,j), j=1, nod)
261:        End Do
262:        Write (*, *) 'finish_Gradient'
263:
264:  !******Calculate link travel flow with Sensitivity analysis******************
265:  !****Results with the change of zeta*********
266:        zeta = 0.0D0
267:  !     xi = 0.0D0
268:
269:        Open (35, File='35.xafter(zeta).txt')
270:  !     Open (36, File='36.xafter(xi).txt')
271:  !     Open (58, File='58.Objective(zeta).txt')
272:  !     Open (59, File='59.Objective(xi).txt')
273:
274:        zeta = 0.0D0
275:        xi = 0.0D0
276:  !     Do ii = 1, 10
277:  !       Write (6, *) 'round', ii
278:  !       If (ii<=5) Then
279:  !         zeta = -t*ii/100.0D0
280:        zeta = 1.0D0
281:        xafter = matmul(gradzeta, zeta)
282:  !     obj = 0.0D0
283:        Do i = 1, link
284:          xafter(i) = xo(i) + xafter(i)
285:  !       tafter(i) = (t(i)+zeta(i))*(1+alpha*((xafter(i)/cap(i))**beta))
286:  !           obj = obj + xafter(i)*tafter(i)
287:        End Do
288:
289:        Do i = 1, link
290:          Write (35, *) xafter(i)
291:        End Do
292:  !         Write (58, *) obj
293:  !   Else
294:  !!****Results with the change of xi*********
295:  !         xi = -(ii-5.0)*yde/100.0D0
296:  !         xafter = matmul(gradxi, xi)
297:  !         obj = 0.0D0
298:  !         Do i = 1, link
299:  !           xafter(i) = xo(i) + xafter(i)
300:  !           tafter(i) = t(i)*(1+alpha*((xafter(i)/cap(i))**beta))
301:  !           obj = obj + xafter(i)*tafter(i)
302:  !         End Do
303:  !
304:  !
305:  !         Do i = 1, link
306:  !           Write (36, *) xafter(i)
307:  !         End Do
308:  !         Write (59, *) obj
309:  !       End If
310:  !     End Do
311:        Call cpu_time(v)
```

```
312:       Write (11, *) 'time 2 (sec)', v
313:
314:   !*****SUBROUTINE*************************************************************
315:      Contains
316:       Subroutine dijikstra(p, t)
317:         Real, Intent (In) :: t(link)
318:         Real *8, Intent (Out) :: p(node, node)
319:         Real *8 :: c(node), cmin(node), fin_io(node)
320:
321:         Do j = 1, node
322:           io = j !io is the source node number
323:           c(:) = 10000000.0 !set partial path cost • ‡
324:           c(io) = 0.0 !set the cost of the source node to 0
325:           cmin(:) = 10000000.0
326:           fin_io(:) = 0 !array for storing the nodes for which calculation has been completed
327:           Do While (minval(cmin,1)<50000000)
328:             Do iii = headnode(io-1) + 1, headnode(io)
329:               i = ls(io, anode(iii))
330:               If (c(io)+t(i)<=c(e(i))) Then
331:                 c(e(i)) = c(io) + t(i) !update with the next link cost with the node
332:                 cmin(e(i)) = c(io) + t(i)
333:               End If
334:             End Do
335:             ncj = minloc(cmin, 1) !identify the location of cmin in the array
336:             cmin(ncj) = 50000000.0 !ineffectiveness of extracted cmin
337:   !**** calculation end judgment (end when all cmin are disabled) ***********
338:             If (minval(cmin,1)==50000000) Then
339:               Go To 100
340:             End If
341:   !******!move to the next node******************
342:             io = ncj
343:             If (fin_io(ncj)==1) Then !when calculation is completed, move to another node
344:               inot = minloc(fin_io, 1)
345:               io = inot
346:             End If
347:             fin_io(io) = 1
348:           End Do
349:   100      Continue
350:   !******record shortest path results************************************
351:           p(j, :) = c(:)
352:         End Do
353:
354:       End Subroutine
355:
356:   !    Calculate xijrs and dijghrs with DFS
357:       Recursive Subroutine calxij(dfs)
358:         Integer dfs, iii
359:
360:         If (j==des(ioi)) Then
361:           swn = swn + wn(j)
362:           Do i = 1, k - 1
363:             wl(mark(i)) = wl(mark(i)) + wn(j)
```

```
364:              Do ii = i, k
365:                dijgh(mark(i), mark(ii)) = dijgh(mark(i), mark(ii)) + wn(j)
366:              End Do
367:            End Do
368:            wl(mark(k)) = wl(mark(k)) + wn(j)
369:          Else
370:            Do iii = headnode(j-1) + 1, headnode(j)
371:              If (omega(ioi,alink(iii))/=0) Then
372:                j = e(alink(iii))
373:                k = k + 1
374:                wn(j) = a(alink(iii))*wn(s(alink(iii)))
375:                mark(k) = alink(iii)
376:                Call calxij(dfs+1)
377:                mark(k) = 0
378:                k = k - 1
379:              End If
380:            End Do
381:          End If
382:        End Subroutine
383:   !******FINISH SUBROUTINE***********************************
384:
385:       End Program
```

E.8 New sensitivity analysis method for SUE problem based on STOCH3 algorithm

```
1:      !Sensitivity analysis for SUE with STOCH3 algorithm
2:          Integer i, j, k, ii, ioi, io, count
3:          Integer, Parameter :: node = 6 !number of nodes
4:          Integer, Parameter :: link = 8 !number of links
5:          Integer, Parameter :: nod = 2 !number of OD pairs
6:          Real *8, Parameter :: theta = 1.0D0 !theta parameter
7:          Real *8, Parameter :: alpha = 1.0D0 !BPR parameter
8:          Real *8, Parameter :: beta = 2.0D0 !BPR parameter
9:          Integer s(link), e(link), ls(node, node) !start and end node and link number
10:         Integer ori(nod), des(nod) !origin, destination of od pairs
11:         Real *8 yde(nod) !od demand of od pairs
12:         Real t(link), cap(link) !t:free-flow travel time,cap: capacity of link
13:         Real *8 erh(link), p(node, node) !erh: stoch3 parameter, p:shortest route
14:         Integer headnode(0:node), anode(link) ! adjacent nodes
15:         Real *8 ps(link), pe(link)
16:     !ps,pe: shortest path from each origin to start and end node of the link
17:         Integer ss(link), ee(link)
18:     !ss: Start node,ee: end node in arranged shortest path
19:         Real *8 a(link), wl(link), wn(node)
20:     !a:link likelihood,w:link weight,wn:node weight
21:         Integer omega(nod, link) !Used to check STOCH3-efficient path
22:         Real *8 xo(link) !xo: link traffic flow at SUE
23:         Real *8 tt(link) !tt: link travel time at SUE
24:         Real *8 x(link), xx(node)
25:     !x: link travel is assigned, xx:total node travel in STOCH3's Algorithm
26:         Real *8 xijrs(link)
27:         Real *8 xijgh1(link, link), xijgh(link, link) ! Used to calculate xijgh
28:         Real *8 tx(link) !tx:derivative of t respect to x
29:         Real *8 gradientgt(link, link)
30:         Real *8 part1(link, link), gradx(link, link)
31:         Real *8 www, pivot, part2(link, 2*link), gradzeta(link, link)
32:         Real *8 gradxi(link, nod), gradgq(link, nod)
33:         Real *8 zeta(link), xi(nod)
34:         Real *8 xafter(link) !link travel flow after sensitivity analysis
35:         Real *8 tzeta(link)
36:     !*****************************************************
37:
38:         Open (1, File='1.network.txt', Action='read')
39:         Open (2, File='2.linkparameter.txt', Action='read')
40:         Open (3, File='3.demand.txt', Action='read')
41:         Open (11, File='19.CalculationTime(SA).txt', Action='write')
42:         Do i = 1, link
43:           Read (1, *) s(i), e(i), j !s(i):start node, e(i):end node
44:           Read (2, *) t(i), cap(i) !t(i):free flow traveltime, cap(i):Capacity of link i
45:           erh(i) = 1.5
46:         End Do
47:         Do i = 1, nod
48:           Read (3, *) ori(i), des(i), yde(i)
49:         End Do
```

```
50:   !make network
51:         ls = 0.0
52:         Do i = 1, link
53:           ls(s(i), e(i)) = i
54:         End Do
55:   ! create adjacent nodes and links
56:         headnode(0) = 0
57:         count = 0
58:         Do i = 1, node
59:           Do j = 1, link
60:             If (i==s(j)) Then
61:               count = count + 1
62:               anode(count) = e(j)
63:   !            alink(count) = j
64:             End If
65:           End Do
66:           headnode(i) = count
67:         End Do
68:
69:         Write (6, *) 'theta', theta
70:         Write (6, *) 'alpha', alpha
71:         Write (6, *) 'beta', beta
72:
73:         Open (12, File='12.xij.txt')
74:         Do i = 1, link
75:           Read (12, *) xo(i)
76:         End Do
77:         Do i = 1, link
78:           tt(i) = t(i)*(1+alpha*((xo(i)/cap(i))**beta)) !BPR calculation
79:         End Do
80:   !***** Calculating derivitives of implicit BPR function*****************
81:         Do i = 1, link
82:           tx(i) = t(i)*beta*alpha*(xo(i)**(beta-1))/(cap(i)**beta)
83:           tzeta(i) = 1 + alpha*xo(i)**beta/(cap(i)**beta)
84:         End Do
85:         Open (21, File='21.tx.txt')
86:         Do i = 1, link
87:           Write (21, *) tx(i)
88:         End Do
89:         Write (*, *) 'finish_tx'
90:         Open (23, File='23.pjsgh.txt')
91:         omega = 0
92:         gradientgt = 0.0
93:         xijrs = 0
94:         gradgq = 0.0
95:
96:   !******** Dijkstra method calculation ****************
97:         Call dijikstra(p, t)
98:
99:   !************************
100:        a = 0.0
101:        Do i = 1, link
```

```
102:         a(i) = exp(-theta*(tt(i)))
103:       End Do
104:  !********STOCH3 algorithm with each OD pair*****************************
105:       Do ioi = 1, nod
106:
107:         io = ori(ioi)
108:         xx = 0.0
109:         xx(des(ioi)) = yde(ioi)
110:
111:         Do i = 1, link
112:           ps(i) = p(io, s(i))
113:           pe(i) = p(io, e(i))
114:         End Do
115:         Do i = 1, link
116:           If (pe(i)>ps(i)) Then
117:  !          If ((1+erh(i))*(pe(i)-ps(i))>=t(i)) Then
118:             omega(ioi, i) = 1
119:  !          End If
120:           End If
121:         End Do
122:
123:  !*****sort shortest path from current node*****************************
124:         Do i = 1, link
125:           k = minloc(pe, 1)
126:           pe(k) = 10000001.0
127:           ss(i) = s(k)
128:           ee(i) = e(k)
129:         End Do
130:  !***** Calculation of link weight and node weight**********************
131:         wl = 0.0
132:         wn = 0.0
133:         Do i = 1, link !Ps(i)Calculate in ascending sequence
134:           nn = ls(ss(i), ee(i)) !NN is the link number
135:           wn(io) = 1.0
136:           If (omega(ioi,nn)==1) Then
137:             wl(nn) = wn(ss(i))*a(nn) !Link weight
138:             wn(ee(i)) = wn(ee(i)) + wl(nn) !Node weight
139:           End If
140:         End Do
141:  !******** Assign trip volume************
142:         x = 0.0
143:         Do i = 1, link
144:           ii = link + 1 - i !Ps(i)Calculate in descending order
145:           nn = ls(ss(ii), ee(ii)) !NN is the link number
146:           If (wn(ee(ii))==0) Go To 100 !Prevents division by 0
147:           If (omega(ioi,nn)==1) Then
148:             x(nn) = xx(ee(ii))*wl(nn)/wn(ee(ii))
149:             xx(ss(ii)) = xx(ss(ii)) + x(nn)
150:           End If
151:  100    End Do
152:  !      write (*,*) 'finish_xijrs'
153:  !************************************************************
```

```
154:          xijrs = x
155:          Do i = 1, link
156:            gradgq(i, ioi) = gradgq(i, ioi) + x(i)/yde(ioi)
157:          End Do
158:          xijgh1 = 0.0
159:    !************
160:          Do kk = 1, link
161:            If (xijrs(kk)/=0) Then
162:              io = e(kk)
163:    !***** Calculation of link weight and node weight*********************
164:              wl = 0.0
165:              wn = 0.0
166:              Do i = 1, link !Ps(i)Calculate in ascending sequence
167:                nn = ls(ss(i), ee(i)) !NN is the link number
168:                wn(io) = 1.0
169:                If (omega(ioi,nn)==1) Then
170:                  wl(nn) = wn(ss(i))*a(nn) !Link weight
171:                  wn(ee(i)) = wn(ee(i)) + wl(nn) !Node weight
172:                End If
173:              End Do
174:    !******** Assign trip volume************
175:              x = 0.0
176:              xx = 0.0
177:              xx(des(ioi)) = xijrs(kk)
178:              Do i = 1, link
179:                ii = link + 1 - i !Ps(i)Calculate in descending order
180:                nn = ls(ss(ii), ee(ii)) !NN is the link number
181:                If (wn(ee(ii))==0) Go To 110 !Prevents division by 0
182:                If (omega(ioi,nn)==1) Then
183:                  x(nn) = xx(ee(ii))*wl(nn)/wn(ee(ii))
184:                  xx(ss(ii)) = xx(ss(ii)) + x(nn)
185:                End If
186:    110         End Do
187:    !***************************
188:              Do j = 1, link
189:                xijgh1(kk, j) = x(j)
190:              End Do
191:            End If
192:          End Do
193:
194:          xijgh = 0.0
195:          Do i = 1, link
196:            Do j = 1, link
197:              xijgh(i, j) = xijgh1(i, j) + xijgh1(j, i)
198:            End Do
199:            xijgh(i, i) = xijrs(i)
200:          End Do
201:
202:          Do i = 1, link
203:            Do j = 1, link
204:              gradientgt(i, j) = gradientgt(i, j) + theta*(xijrs(i)*xijrs(j)/yde &
205:                (ioi)-xijgh(i,j))
```

```
206:            End Do
207:          End Do
208:  !       write (*,*) 'finish_xijgh'
209:        End Do
210:  !*************************************************************
211:        Call cpu_time(v)
212:        Write (11, *) 'Calculation time 1 (second)', v
213:
214:  !*******Record results*****
215:        Open (24, File='22.gradientgt.txt')
216:        Do i = 1, link
217:          Write (24, *) gradientgt(i, :)
218:        End Do
219:        Write (*, *) 'finish_gradientgt'
220:
221:  !*******Calculation Gradient**********
222:        gradx = 0.0
223:        part1 = 0.0
224:        part2 = 0.0
225:        Do i = 1, link
226:          Do j = 1, link
227:            gradx(i, j) = gradientgt(i, j)*tx(j)
228:          End Do
229:        End Do
230:
231:        Do i = 1, link
232:          Do j = 1, link
233:            If (i==j) Then
234:              part1(i, j) = 1 - gradx(i, j)
235:            Else
236:              part1(i, j) = -gradx(i, j)
237:            End If
238:          End Do
239:        End Do
240:        Do i = 1, link
241:          Do j = 1, link
242:            part2(i, j) = part1(i, j)
243:          End Do
244:        End Do
245:
246:        Do i = 1, link
247:          Do j = 1, link
248:            part2(i, j+link) = 0
249:            part2(i, i+link) = 1
250:          End Do
251:        End Do
252:
253:        Do k = 1, link
254:          pivot = part2(k, k)
255:          If (pivot==0) Then
256:            i = k + 1
257:            Do While (pivot==0 .And. i<link)
```

```
258:            pivot = part2(i, k)
259:            i = i + 1
260:          End Do
261:          If (pivot==0) Then
262:            Stop
263:          Else
264:            Do j = 1, 2*link
265:              www = part2(k, j)
266:              i = i - 1
267:              part2(k, j) = part2(i, j)
268:              part2(i, j) = www
269:            End Do
270:          End If
271:        End If
272:        Do j = 1, link*2
273:          part2(k, j) = part2(k, j)/pivot
274:        End Do
275:
276:        Do i = 1, link
277:          If (i/=k) Then
278:            www = part2(i, k)
279:            Do j = 1, link*2
280:              part2(i, j) = part2(i, j) - www*part2(k, j)
281:            End Do
282:          End If
283:        End Do
284:      End Do
285:
286:      gradx = 0.0
287:      Do i = 1, link
288:        Do j = 1, link
289:          gradx(i, j) = part2(i, j+link)
290:        End Do
291:      End Do
292:
293:      part1 = 0.0
294:      Do i = 1, link
295:        Do j = 1, link
296:          part1(i, j) = gradientgt(i, j)*tzeta(j)
297:        End Do
298:      End Do
299:      gradzeta = 0.0
300:      gradzeta = matmul(gradx, part1)
301:      gradxi = 0.0
302:      gradxi = matmul(gradx, gradgq)
303:      Open (27, File='27.Gradient(zeta).txt')
304:      Do i = 1, link
305:        Write (27, *)(gradzeta(i,j), j=1, link)
306:      End Do
307:      Open (28, File='28.Gradient(xi).txt')
308:      Do i = 1, link
309:        Write (28, *)(gradxi(i,j), j=1, nod)
```

```
310:        End Do
311:        Write (*, *) 'finish_Gradient'
312:
313:  !******Calculate link travel flow with Sensitivity analysis*****************
314:  !****Results with the change of zeta*********
315:        zeta = 0.0D0
316:  !     xi = 0.0D0
317:
318:        Open (35, File='35.xafter(zeta).txt')
319:  !     Open (36, File='36.xafter(xi).txt')
320:  !     Open (58, File='58.Objective(zeta).txt')
321:  !     Open (59, File='59.Objective(xi).txt')
322:
323:        zeta = 0.0D0
324:        xi = 0.0D0
325:  !     Do ii = 1, 10
326:  !       Write (6, *) 'round', ii
327:  !       If (ii<=5) Then
328:  !         zeta = -t*ii/100.0D0
329:        zeta = 1.0D0
330:        xafter = matmul(gradzeta, zeta)
331:  !     obj = 0.0D0
332:        Do i = 1, link
333:          xafter(i) = xo(i) + xafter(i)
334:  !       tafter(i) = (t(i)+zeta(i))*(1+alpha*((xafter(i)/cap(i))**beta))
335:  !           obj = obj + xafter(i)*tafter(i)
336:        End Do
337:
338:        Do i = 1, link
339:          Write (35, *) xafter(i)
340:        End Do
341:  !         Write (58, *) obj
342:  !   Else
343:  !!****Results with the change of xi*********
344:  !         xi = -(ii-5.0)*yde/100.0D0
345:  !         xafter = matmul(gradxi, xi)
346:  !         obj = 0.0D0
347:  !         Do i = 1, link
348:  !           xafter(i) = xo(i) + xafter(i)
349:  !           tafter(i) = t(i)*(1+alpha*((xafter(i)/cap(i))**beta))
350:  !           obj = obj + xafter(i)*tafter(i)
351:  !         End Do
352:  !
353:  !
354:  !         Do i = 1, link
355:  !           Write (36, *) xafter(i)
356:  !         End Do
357:  !         Write (59, *) obj
358:  !       End If
359:  !     End Do
360:        Call cpu_time(v)
361:        Write (11, *) 'time 2 (sec)', v
```

```
362:
363:  !*****SUBROUTINE************************************************************
364:     Contains
365:       Subroutine dijikstra(p, t)
366:         Real, Intent (In) :: t(link)
367:         Real *8, Intent (Out) :: p(node, node)
368:         Real *8 :: c(node), cmin(node), fin_io(node)
369:
370:         Do j = 1, node
371:           io = j !io is the source node number
372:           c(:) = 10000000.0 !set partial path cost □‡
373:           c(io) = 0.0 !set the cost of the source node to 0
374:           cmin(:) = 10000000.0
375:           fin_io(:) = 0 !array for storing the nodes for which calculation has been completed
376:           Do While (minval(cmin,1)<50000000)
377:             Do iii = headnode(io-1) + 1, headnode(io)
378:               i = ls(io, anode(iii))
379:               If (c(io)+t(i)<=c(e(i))) Then
380:                 c(e(i)) = c(io) + t(i) !update with the next link cost with the node
381:                 cmin(e(i)) = c(io) + t(i)
382:               End If
383:             End Do
384:             ncj = minloc(cmin, 1) !identify the location of cmin in the array
385:             cmin(ncj) = 50000000.0 !ineffectiveness of extracted cmin
386:  !**** calculation end judgment (end when all cmin are disabled) ***********
387:             If (minval(cmin,1)==50000000) Then
388:               Go To 100
389:             End If
390:  !******!move to the next node******************
391:             io = ncj
392:             If (fin_io(ncj)==1) Then !when calculation is completed, move to another node
393:               inot = minloc(fin_io, 1)
394:               io = inot
395:             End If
396:             fin_io(io) = 1
397:           End Do
398:  100      Continue
399:  !******record shortest path results***********************************
400:           p(j, :) = c(:)
401:         End Do
402:
403:       End Subroutine
404:  !*****FINISH SUBROUTINE***********************************
405:     End Program
```

Appendix F. Fortran coding for Chapter 4

Chapter 4 proposed algorithms including:

- Semi-DTA model based on MNL by solving double-looped fixed-point problem.

- Two link-based sensitivity analysis algorithms for the semi-DTA model.

The following will detail the programs for the proposed algorithm applied to the small network in time period 1. Other networks are similarly applied.

F.1 Semi-DTA model based on MNL by solving double-looped fixed-point problem

```
1:      !* * * * * * *SEMI-DYNAMIC TRAFFIC ASSIGNMENT FOR MNL MODEL* * * * * *
2:      ! * * * * * * * * * * * *SOLVING DOUBLE-LOOPED FIXED-POINT PROBLEM* * * * *
3:            Integer i, j, ij, k, o, ij1, l, m
4:            Integer, Parameter :: round = 5000000 !the number of computation
5:            Real *8, Parameter :: eee = 1.0D-3 !error in computation
6:
7:            Integer, Parameter :: node = 6 !number of node
8:            Integer, Parameter :: link = 6 !number of link
9:            Integer, Parameter :: nod = 3 !number of OD pairs
10:
11:           Integer, Parameter :: path = 4 !the number of paths
12:           Integer, Parameter :: nmax = 3 !maximum number of links on routes+1 in route-set
13:           Real *8, Parameter :: theta = 0.5D0 !theta parameter
14:           Real *8, Parameter :: alpha = 0.15D0 !BPR parameter
15:           Real *8, Parameter :: beta = 4.0D0 !BPR parameter
16:           Real *8, Parameter :: length = 60.0D0 !time length
17:     !*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
18:
19:           Integer delta(path, nmax), kk(nod), sum, n(100000), pn(path, nmax), &
20:             en(link)
21:           Integer b(nmax, nmax), des(nod)
22:           Real *8 t0(link), capa(link), x(link), c0(link), c00(link), rgap(link)
23:           Real *8 q(nod), sgm(nod)
24:           Real *8 c(path), sc(path), p(path), f(path), fgap(path)
25:           Real *8 gapmax, dij
26:           Real *8 od(node, node)
27:           Real *8 y(link), yij(100000), sij(100000, 1), s(link)
28:           Real *8 z(link)
29:
30:     !* * *  * * * * * * * * * * * * * * * * * * * * * * * * *
31:
32:           Open (1, File='1.network.txt')
33:           Do i = 1, link
34:             Read (1, *) k, en(i), j
```

```
35:         End Do
36:
37:         Open (2, File='2.linkparameter.txt')
38:         Do i = 1, link
39:           Read (2, *) t0(i), capa(i)
40:         End Do
41:
42:         Open (3, File='3.demand.txt')
43:         Do i = 1, nod
44:           Read (3, *) o, des(i), q(i)
45:         End Do
46:
47:         Open (66, File='6.NumberofPaths.txt')
48:         Do i = 1, nod
49:           Read (66, *) kk(i)
50:         End Do
51:
52:         Open (7, File='7.pass_link.txt')
53:         Do i = 1, path
54:           Read (7, *)(delta(i,j), j=1, nmax)
55:         End Do
56:
57:     !*********count passed link number******
58:         n(:) = 0.0D0
59:         sum = 0.0D0
60:
61:         Open (46, File='n(i).txt')
62:         Do i = 1, path
63:           Do j = 1, nmax
64:             If (delta(i,j)==0) Go To 100
65:             n(i) = n(i) + 1
66:           End Do
67:     100   End Do
68:
69:         Do i = 1, path
70:           sum = sum + n(i)
71:         End Do
72:
73:         Do i = 1, path
74:           Write (46, *) n(i), sum
75:         End Do
76:
77:     !********* passed node number       ******
78:         pn(:, :) = 0.0D0
79:         Open (47, File='pn.txt')
80:
81:         Do i = 1, path
82:           Do j = 1, n(i)
83:             If (delta(i,j)==0) Go To 110
84:             pn(i, j) = delta(i, j)
85:           End Do
86:     110   End Do
```

```
87:
88:         Do i = 1, path
89:           Do j = 1, n(i)
90:             Write (47, *) pn(i, j)
91:           End Do
92:         End Do
93:
94:     !*********** B  ************
95:         b(:, :) = 0
96:         Do i = 1, nmax
97:           Do j = 1, nmax
98:             If (i<=j) Then
99:               b(i, j) = 0
100:            Else
101:              b(i, j) = 1
102:            End If
103:
104:          End Do
105:        End Do
106:
107:        Open (48, File='b(i,j).txt')
108:        Do i = 1, nmax
109:          Write (48, *)(b(i,j), j=1, nmax)
110:        End Do
111:
112:    !* *  * Solving fixed point problem with reference route flows* * * * *
113:    !* *  * * * creating initial solution of reference route flows * * * * * * * * * * * * * * * * * * * *
114:        m = 0
115:        Do i = 1, nod
116:          Do j = 1, kk(i)
117:            m = m + 1
118:            f(m) = q(i)/kk(i)
119:          End Do
120:        End Do
121:
122:        Do ij = 1, round
123:    !* * *  * * * *Calculating reference link traffic flow * * * * * * * * * *
124:
125:          x(:) = 0.0D0
126:          Do i = 1, path
127:            Do j = 1, nmax
128:              If (delta(i,j)==0) Go To 120
129:              in = delta(i, j)
130:              x(in) = x(in) + f(i)
131:            End Do
132:    120     End Do
133:
134:    !* * *  Solving fixed point problem with link travel time* * * * * *
135:    !* *  * * * creating initial solution of link travel time * * * * * * * * * * * * * * * * * * *
136:          c0 = t0
137:    !* *  * * * solving the problem with MSA method * * * * * * * * * * * * * * * * * * * *
138:          Do ij1 = 1, round
```

```
139:
140:  !*******************  yij************************
141:          yij(:) = 0.0D0
142:          k = 0
143:          Do i = 1, path
144:            Do j = 1, nmax
145:              If (delta(i,j)==0) Go To 130
146:              k = k + 1
147:              o = delta(i, j)
148:              yij(k) = f(i)*c0(o)/length
149:            End Do
150:  130      End Do
151:
152:  !************ sij=b*y  matrix calculation  ****************
153:          sij(:, :) = 0.0D0
154:          m = 0.0D0
155:
156:          Do i = 1, path
157:            sij(m+1:m+n(i), 1) = matmul(b(1:n(i),1:n(i)), yij(m+1:m+n(i)))
158:            m = m + n(i)
159:          End Do
160:
161:  !************ Calculating total eliminated flow  *********************************
162:          s(:) = 0.0D0
163:          k = 0.0D0
164:
165:          Do i = 1, path
166:            Do j = 1, n(i)
167:              k = k + 1
168:              in = pn(i, j)
169:              s(in) = s(in) + sij(k, 1)
170:            End Do
171:          End Do
172:
173:  !************ Calculating adjusted link flows  *********************************
174:          z(:) = 0
175:          Do i = 1, link
176:            z(i) = x(i) - s(i)
177:            If (z(i)<0) Then
178:              z(i) = 0
179:            End If
180:          End Do
181:
182:  ! Calculating auxiliary link travel time and relation gap  *****
183:          c00 = 0.0D0
184:          rgap = 0.0D0
185:          Do i = 1, link
186:            c00(i) = t0(i)*(1+alpha*z(i)**beta/capa(i)**beta)
187:            rgap(i) = c0(i) - c00(i)
188:          End Do
189:
190:          dij = dble(ij1)
```

```
!********** MSA method ********************************
      Do i = 1, link
        c0(i) = c0(i) - rgap(i)*(1.0D0/dij)
      End Do
      gapmax = 0.0D0

      Do i = 1, link
        If (abs(rgap(i))>gapmax) Then
          gapmax = abs(rgap(i))
        End If
      End Do
!       write(*,*)'Computation number (tij)',ij1,gapmax
      If (gapmax<eee) Exit
    End Do
!********** Calculating auxiliary route flow ********************************
!*********** path_travel_time * * * * * * * * * * *
    c(:) = 0.0D0

    Do i = 1, path
      Do k = 1, nmax
        If (delta(i,k)==0) Go To 140
        in = delta(i, k)
        c(i) = c(i) + c0(in)
      End Do
140   End Do

!* * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    sc(:) = 0.0D0
    Do i = 1, path
      sc(i) = exp(-theta*c(i))
    End Do

    sgm = 0.0D0
    m = 0
    Do i = 1, nod
      Do j = 1, kk(i)
        m = m + 1
        sgm(i) = sgm(i) + sc(m)
      End Do
    End Do

!MSA method for solving fixed-point problem with route flows
    m = 0
    Do i = 1, nod
      Do j = 1, kk(i)
        m = m + 1
        fgap(m) = f(m) - q(i)*sc(m)/sgm(i)
      End Do
    End Do

    dij = dble(ij)

```

```
243:         Do i = 1, path
244:           f(i) = f(i) - fgap(i)*(1.0D0/dij)
245:         End Do
246:
247:   !* * * * Checking convergence* * * * * * * * * * * * * * * * * * * * * * * * * *
248:         fgapmax = 0.0D0
249:         Do i = 1, path
250:           If (abs(fgap(i))>fgapmax) Then
251:             fgapmax = abs(fgap(i))
252:           End If
253:         End Do
254:         Write (*, *) 'Computation number (fk)', ij, fgapmax
255:         If (fgapmax<eee) Exit
256:
257:       End Do
258:   !* * * * Results* * * * * * * * * * * * * * * * * * * * * * * * * *
259:   !* *  * * Saving some varibales * * * * * * * * * *
260:       Open (50, File='round,gapmax,c00,Rgap.txt')
261:       Write (50, *) 'round', ij, 'gapmax', gapmax
262:       Do i = 1, link
263:         Write (50, *) c00(i), rgap(i)
264:       End Do
265:
266:       Open (51, File='fgapmax,fgap.txt')
267:       Write (51, *) 'fgapmax', fgapmax
268:       Do i = 1, path
269:         Write (51, *) fgap(i)
270:       End Do
271:
272:   !* *  * * link residual flow * * * * * * * * * *
273:   !*******************  yij ************************
274:       Open (8, File='8.yij.txt')
275:       Open (68, File='68.FlowPropagation.txt')
276:       yij(:) = 0.0D0
277:       k = 0
278:       od = 0.0D0
279:
280:       i = 0
281:       Do l = 1, nod
282:         Do m = 1, kk(l)
283:           i = i + 1
284:           Do j = 1, n(i)
285:             k = k + 1
286:             o = delta(i, j)
287:             yij(k) = f(i)*c0(o)/length
288:             od(en(o), des(l)) = od(en(o), des(l)) + yij(k)
289:             Write (8, *) o, yij(k)
290:           End Do
291:         End Do
292:       End Do
293:       Do i = 1, node
294:         Write (68, *)(od(i,j), j=1, node)
```

```
295:        End Do
296:   !**sum**
297:
298:        y(:) = 0.0D0
299:
300:        Do i = 1, path
301:          Do j = 1, nmax
302:            If (delta(i,j)==0) Go To 150
303:            in = delta(i, j)
304:            y(in) = y(in) + f(i)*c0(in)/length
305:          End Do
306:   150   End Do
307:
308:        Open (11, File='11.y(i).txt')
309:
310:        Do i = 1, link
311:          Write (11, *) y(i)
312:        End Do
313:
314:   !************ sij=b*y  matrix calculation  *****************
315:        Open (12, File='12.sij.txt')
316:        sij(:, :) = 0.0D0
317:        m = 0.0D0
318:
319:        Do i = 1, path
320:          sij(m+1:m+n(i), 1) = matmul(b(1:n(i),1:n(i)), yij(m+1:m+n(i)))
321:          m = m + n(i)
322:        End Do
323:        Do i = 1, sum
324:          Write (12, *) i, sij(i, 1)
325:        End Do
326:
327:   !************ s=sum_sij  *********************************
328:        Open (13, File='13.s(i).txt')
329:        s(:) = 0.0D0
330:        k = 0.0D0
331:
332:        Do i = 1, path
333:          Do j = 1, n(i)
334:            k = k + 1
335:            in = pn(i, j)
336:            s(in) = s(in) + sij(k, 1)
337:          End Do
338:        End Do
339:
340:        Do i = 1, link
341:          Write (13, *) s(i)
342:        End Do
343:
344:   !************ z=x(delta_f)-s   ******************************
345:        Open (14, File='14.z(i).txt')
346:        z(:) = 0
```

```
347:
348:        Do i = 1, link
349:          z(i) = x(i) - s(i)
350:        End Do
351:
352:        Do i = 1, link
353:          Write (14, *) z(i)
354:        End Do
355:
356:   !*********************************************
357:
358:        Write (*, *) 'alpha :', alpha
359:        Write (*, *) 'beta :', beta
360:        Write (*, *) 'theta :', theta
361:        Write (*, *) 'Maximum of the gap', fgapmax
362:
363:   !* ** * * * * * * * * * * * * * * * * *
364:
365:        Open (15, File='15.x(i).txt')
366:        Do i = 1, link
367:          Write (15, *) i, x(i)
368:        End Do
369:
370:        Open (16, File='16.linktraveltime.txt')
371:        Do i = 1, link
372:          Write (16, *) i, c0(i)
373:        End Do
374:
375:        Open (17, File='17.sc.txt')
376:        Do i = 1, path
377:          Write (17, *) i, sc(i)
378:        End Do
379:
380:        Open (20, File='20.c(i),f(i).txt')
381:        Do i = 1, path
382:          Write (20, *) i, c(i), f(i)
383:        End Do
384:
385:        Open (23, File='23.Calculation time.txt')
386:        Call cpu_time(v)
387:        Write (23, *) 'time', v
388:
389:        Stop
390:
391:      End Program
```

F.2 The first algorithm of link-based STOCH3 sensitivity analysis approach for the semi-DTA model

```
1:    ! *******semi-dynamic traffic assignment for multinomial logit model**********
2:    ! **********first-order approach based on stoch3 and DFS algorithm **********
3:        Integer i, j, k, ii, ioi, io, count
4:        Integer, Parameter :: node = 6 !number of nodes
5:        Integer, Parameter :: link = 6 !number of links
6:        Integer, Parameter :: nod = 3 !number of OD pairs in time period 1
7:        Real *8, Parameter :: theta = 0.5D0 !theta parameter
8:        Real *8, Parameter :: alpha = 0.15D0 !BPR parameter
9:        Real *8, Parameter :: beta = 4.0D0 !BPR parameter
10:       Real *8, Parameter :: lengtht = 60D0 !length of period (minutes)
11:       Integer s(link), e(link), ls(node, node) !start and end node and link number
12:       Integer ori(nod), des(nod) !origin, destination of od pairs
13:       Real *8 yde(nod) !od demand of od pairs
14:       Real t(link), cap(link) !t:free-flow travel time,cap: capacity of link
15:       Real *8 erh(link), p(node, node) !erh: stoch3 parameter, p:shortest route
16:       Integer headnode(0:node), anode(link), alink(link) ! adjacent nodes and links
17:       Real *8 ps(link), pe(link)
18:   !ps,pe: shortest route from each origin to start and end node of the link
19:       Integer ss(node, link), ee(node, link)
20:   !ss: Start node,ee: end node in arranged shortest route
21:       Real *8 a(link), wl(link), wn(node)
22:   !a:link likelihood,w:link weight,wn:node weight
23:       Integer omega(node, link), mark(node) !Used to check STOCH3-efficient route
24:       Real *8 xo(link) !xo: link traffic flow at SUE
25:       Real *8 tt(link) !tt: link travel time at SUE
26:       Real *8 x(link), xx(node)
27:   !x: link travel is assigned, xx:total node travel in STOCH3's Algorithm
28:       Real *8 tx(link) !tx:derivative of t respect to x
29:   !Variable used to calculate derivative of u respect to t and Detivative of x respect to s
30:       Real *8 gradgt(link, link) !used to calculate derivative of u respect to t
31:       Real *8 part1(link, link), part2(link, link)
32:       Real *8 www, pivot, part3(link, 2*link)
33:       Real *8 grad(link, link) !Grad: Detivative of x respect to s
34:       Real *8 sij(link), yij(link), yijrs(link), b(node, node) !sij:total eliminated flow
35:       Real *8 xafter(link) !link travel flow after sensitivity analysis
36:       Real *8 xsueafter(link)
37:       Real *8 xijrs(link), xijgh1(link, link), xijgh(link, link) ! Used to calculate xijgh
38:
39:   !*************************************************
40:
41:       Open (1, File='1.network.txt', Action='read')
42:       Open (2, File='2.linkparameter.txt', Action='read')
43:       Open (3, File='3.demand.txt', Action='read')
44:       Open (11, File='19.CalculationTime(SA).txt', Action='write')
45:       Do i = 1, link
46:         Read (1, *) s(i), e(i), j !s(i):start node, e(i):end node
47:         Read (2, *) t(i), cap(i) !t(i):free flow traveltime, cap(i):Capacity of link i
48:         erh(i) = 1.5
49:       End Do
```

```
50:         Do i = 1, nod
51:           Read (3, *) ori(i), des(i), yde(i)
52:         End Do
53:     !make network
54:         ls = 0.0
55:         Do i = 1, link
56:           ls(s(i), e(i)) = i
57:         End Do
58:     ! create adjacent nodes and links
59:         headnode(0) = 0
60:         count = 0
61:         Do i = 1, node
62:           Do j = 1, link
63:             If (i==s(j)) Then
64:               count = count + 1
65:               anode(count) = e(j)
66:               alink(count) = j
67:             End If
68:           End Do
69:           headnode(i) = count
70:         End Do
71:
72:         Write (6, *) 'theta', theta
73:         Write (6, *) 'alpha', alpha
74:         Write (6, *) 'beta', beta
75:
76:         Open (12, File='12.xij.txt')
77:         Do i = 1, link
78:           Read (12, *) xo(i)
79:         End Do
80:         Do i = 1, link
81:           tt(i) = t(i)*(1+alpha*((xo(i)/cap(i))**beta)) !BPR calculation
82:         End Do
83:     !Calculate xijrs,xijgh (EACH OD PAIR) and gradgt
84:         omega = 0
85:         mark = 0
86:         gradgt = 0.0
87:         yij = 0.0
88:         sij = 0.0
89:         b = 0.0
90:         xijrs = 0.0
91:     !******** Dijkstra method calculation *****************
92:         Call dijikstra(p, t)
93:     !***** Calculation of link likeli-hood **********************
94:         a = 0.0
95:         Do i = 1, link
96:           a(i) = exp(-theta*(tt(i)))
97:         End Do
98:     !********STOCH3 algorithm with each OD pair*****************************
99:         Do ioi = 1, nod
100:
101:          io = ori(ioi)
```

```
102:         xx = 0.0
103:         xx(des(ioi)) = yde(ioi)
104:
105:
106:         If (mark(io)==0) Then
107:           mark(io) = 1
108:           Do i = 1, link
109:             ps(i) = p(io, s(i))
110:             pe(i) = p(io, e(i))
111:           End Do
112:           Do i = 1, link
113:             If (pe(i)>ps(i)) Then
114:               If ((1+erh(i))*(pe(i)-ps(i))>=t(i)) Then
115:                 omega(ori(ioi), i) = 1
116:               End If
117:             End If
118:           End Do
119:  !*****sort shortest path from current node*****************************
120:           Do i = 1, link
121:             k = minloc(pe, 1)
122:             pe(k) = 10000001.0
123:             ss(ori(ioi), i) = s(k)
124:             ee(ori(ioi), i) = e(k)
125:           End Do
126:
127:         End If
128:
129:  !***** Calculation of link weight and node weight*********************
130:         wl = 0.0
131:         wn = 0.0
132:         Do i = 1, link !Ps(i)Calculate in ascending sequence
133:           nn = ls(ss(ori(ioi),i), ee(ori(ioi),i)) !NN is the link number
134:           wn(io) = 1.0
135:           If (omega(ori(ioi),nn)==1) Then
136:             wl(nn) = wn(ss(ori(ioi),i))*a(nn) !Link weight
137:             wn(ee(ori(ioi),i)) = wn(ee(ori(ioi),i)) + wl(nn) !Node weight
138:           End If
139:         End Do
140:  !******** Assign trip volume************
141:         x = 0.0
142:         Do i = 1, link
143:           ii = link + 1 - i !Ps(i)Calculate in descending order
144:           nn = ls(ss(ori(ioi),ii), ee(ori(ioi),ii)) !NN is the link number
145:           If (wn(ee(ori(ioi),ii))==0) Go To 100 !Prevents division by 0
146:           If (omega(ori(ioi),nn)==1) Then
147:             x(nn) = xx(ee(ori(ioi),ii))*wl(nn)/wn(ee(ori(ioi),ii))
148:             xx(ss(ori(ioi),ii)) = xx(ss(ori(ioi),ii)) + x(nn)
149:           End If
150:  100    End Do
151:  !      write (*,*) 'finish_xijrs'
152:  !************************************************************
153:         xijrs = x
```

```
154:        xijgh1 = 0.0
155:  !************
156:        Do kk = 1, link
157:          If (xijrs(kk)/=0) Then
158:            io = e(kk)
159:  !***** Calculation of link weight and node weight*********************
160:            wl = 0.0
161:            wn = 0.0
162:            Do i = 1, link !Ps(i)Calculate in ascending sequence
163:              nn = ls(ss(ori(ioi),i), ee(ori(ioi),i)) !NN is the link number
164:              wn(io) = 1.0
165:              If (omega(ori(ioi),nn)==1) Then
166:                wl(nn) = wn(ss(ori(ioi),i))*a(nn) !Link weight
167:                wn(ee(ori(ioi),i)) = wn(ee(ori(ioi),i)) + wl(nn) !Node weight
168:              End If
169:            End Do
170:  !******** Assign trip volume*************
171:            x = 0.0
172:            xx = 0.0
173:            xx(des(ioi)) = xijrs(kk)
174:            Do i = 1, link
175:              ii = link + 1 - i !Ps(i)Calculate in descending order
176:              nn = ls(ss(ori(ioi),ii), ee(ori(ioi),ii)) !NN is the link number
177:              If (wn(ee(ori(ioi),ii))==0) Go To 110 !Prevents division by 0
178:              If (omega(ori(ioi),nn)==1) Then
179:                x(nn) = xx(ee(ori(ioi),ii))*wl(nn)/wn(ee(ori(ioi),ii))
180:                xx(ss(ori(ioi),ii)) = xx(ss(ori(ioi),ii)) + x(nn)
181:              End If
182:  110         End Do
183:  !***************************
184:            Do j = 1, link
185:              xijgh1(kk, j) = x(j)
186:            End Do
187:          End If
188:        End Do
189:        yijrs = 0.0
190:        Do i = 1, link
191:          yijrs(i) = yijrs(i) + xijrs(i)/lengtht*tt(i)
192:          yij(i) = yij(i) + yijrs(i)
193:          Do j = 1, link
194:            If (xijgh1(i,j)/=0) Then
195:              sij(j) = sij(j) + xijgh1(i, j)/lengtht*tt(i)
196:            End If
197:          End Do
198:        End Do
199:
200:        Do i = 1, link
201:          If (yijrs(i)/=0) Then
202:            b(e(i), des(ioi)) = b(e(i), des(ioi)) + yijrs(i)
203:          End If
204:        End Do
205:        xijgh = 0.0
```

```
206:          Do i = 1, link
207:            Do j = 1, link
208:              xijgh(i, j) = xijgh1(i, j) + xijgh1(j, i)
209:            End Do
210:            xijgh(i, i) = xijrs(i)
211:          End Do
212:
213:          Do i = 1, link
214:            Do j = 1, link
215:              gradgt(i, j) = gradgt(i, j) + theta*(xijrs(i)*xijrs(j)/yde(ioi)- &
216:                xijgh(i,j))
217:            End Do
218:          End Do
219:  !       write (*,*) 'finish_xijgh'
220:        End Do
221:
222:  !*******Record results*****
223:        Call cpu_time(v)
224:        Write (11, *) 'Calculation time 1 (second)', v
225:
226:        Open (22, File='22.gradgt.txt')
227:        Do i = 1, link
228:          Write (22, *) gradgt(i, :)
229:        End Do
230:        Write (*, *) 'finish_gradgt'
231:
232:        Open (23, File='23.sij.txt')
233:        Do i = 1, link
234:          Write (23, *) sij(i)
235:        End Do
236:        Open (24, File='24.yij.txt')
237:        Do i = 1, link
238:          Write (24, *) yij(i)
239:        End Do
240:  !***** Calculating derivitives of implicit BPR function*****************
241:        Do i = 1, link
242:          tx(i) = t(i)*beta*alpha*(xo(i)**(beta-1))/(cap(i)**beta)
243:        End Do
244:        Open (25, File='25.tx.txt')
245:        Do i = 1, link
246:          Write (25, *) tx(i)
247:        End Do
248:  !*******Calculation Gradient**********
249:        part1 = 0.0
250:        part2 = 0.0
251:        part3 = 0.0
252:        Do i = 1, link
253:          Do j = 1, link
254:            part1(i, j) = gradgt(i, j)*tx(j)
255:          End Do
256:        End Do
257:
```

```
258:        Do i = 1, link
259:          Do j = 1, link
260:            If (i==j) Then
261:              part2(i, j) = 1 - part1(i, j)
262:            Else
263:              part2(i, j) = -part1(i, j)
264:            End If
265:          End Do
266:        End Do
267:        Do i = 1, link
268:          Do j = 1, link
269:            part3(i, j) = part2(i, j)
270:          End Do
271:        End Do
272:
273:        Do i = 1, link
274:          Do j = 1, link
275:            part3(i, j+link) = 0
276:            part3(i, i+link) = 1
277:          End Do
278:        End Do
279:
280:        Do k = 1, link
281:          pivot = part3(k, k)
282:          If (pivot==0) Then
283:            i = k + 1
284:            Do While (pivot==0 .And. i<link)
285:              pivot = part3(i, k)
286:              i = i + 1
287:            End Do
288:            If (pivot==0) Then
289:              Stop
290:            Else
291:              Do j = 1, 2*link
292:                www = part3(k, j)
293:                i = i - 1
294:                part3(k, j) = part3(i, j)
295:                part3(i, j) = www
296:              End Do
297:            End If
298:          End If
299:          Do j = 1, link*2
300:            part3(k, j) = part3(k, j)/pivot
301:          End Do
302:
303:          Do i = 1, link
304:            If (i/=k) Then
305:              www = part3(i, k)
306:              Do j = 1, link*2
307:                part3(i, j) = part3(i, j) - www*part3(k, j)
308:              End Do
309:            End If
```

```
310:          End Do
311:        End Do
312:
313:        part2 = 0.0
314:        Do i = 1, link
315:          Do j = 1, link
316:            part2(i, j) = part3(i, j+link)
317:          End Do
318:        End Do
319:        grad = 0
320:        grad = -matmul(part2, part1)
321:        Open (26, File='26.grad.txt')
322:        Do i = 1, link
323:          Write (26, *) grad(i, :)
324:        End Do
325:        Write (*, *) 'finish_Gradient'
326:
327:  !*****Calculate link travel flow and link travel time after Sensitivity
      analysis******************
328:
329:        xsueafter = matmul(grad, sij)
330:        Do i = 1, link
331:          xsueafter(i) = xo(i) + xsueafter(i)
332:          xafter(i) = max((xsueafter(i)-sij(i)), 0D0)
333:        End Do
334:
335:        Open (37, File='37.xSUEafter.txt')
336:        Do i = 1, link
337:          Write (37, *) xsueafter(i)
338:        End Do
339:
340:        Open (38, File='38.xafter(subtracted).txt')
341:
342:        Do i = 1, link
343:          Write (38, *) xafter(i)
344:        End Do
345:
346:        Open (68, File='68.FlowPropagation.txt')
347:        Do i = 1, node
348:          b(i, i) = 0
349:          Write (68, *)(b(i,j), j=1, node)
350:        End Do
351:
352:        Call cpu_time(v)
353:        Write (11, *) 'Calculation time 2 (second)', v
354:
355:  !*****SUBROUTINE************************************************************
356:      Contains
357:        Subroutine dijikstra(p, t)
358:          Real, Intent (In) :: t(link)
359:          Real *8, Intent (Out) :: p(node, node)
360:          Real *8 :: c(node), cmin(node), fin_io(node)
```

```
361:
362:         Do j = 1, node
363:           io = j !io is the source node number
364:           c(:) = 10000000.0 !set partial route cost • ‡
365:           c(io) = 0.0 !set the cost of the source node to 0
366:           cmin(:) = 10000000.0
367:           fin_io(:) = 0 !array for storing the nodes for which calculation has been completed
368:           Do While (minval(cmin,1)<50000000)
369:             Do iii = headnode(io-1) + 1, headnode(io)
370:               i = ls(io, anode(iii))
371:               If (c(io)+t(i)<=c(e(i))) Then
372:                 c(e(i)) = c(io) + t(i) !update with the next link cost with the node
373:                 cmin(e(i)) = c(io) + t(i)
374:               End If
375:             End Do
376:             ncj = minloc(cmin, 1) !identify the location of cmin in the array
377:             cmin(ncj) = 50000000.0 !ineffectiveness of extracted cmin
378:   !**** calculation end judgment (end when all cmin are disabled) ***********
379:             If (minval(cmin,1)==50000000) Then
380:               Go To 100
381:             End If
382:   !******!move to the next node******************
383:             io = ncj
384:             If (fin_io(ncj)==1) Then !when calculation is completed, move to another node
385:               inot = minloc(fin_io, 1)
386:               io = inot
387:             End If
388:             fin_io(io) = 1
389:           End Do
390:   100       Continue
391:   !******record shortest route results************************************
392:           p(j, :) = c(:)
393:         End Do
394:
395:       End Subroutine
396:   !*****FINISH SUBROUTINE***********************************
397:     End Program
```

F.3 The second algorithm of link-based STOCH3 sensitivity analysis approach for the semi-DTA model

```
1:    ! *******semi-dynamic traffic assignment for multinomial logit model**********
2:    ! **********the second algorithm based on stoch3 and DFS algorithm **********
3:        Integer i, j, k, ii, ioi, io, count
4:        Integer, Parameter :: node = 6 !number of nodes
5:        Integer, Parameter :: link = 6 !number of links
6:        Integer, Parameter :: nod = 3 !number of OD pairs in time period 1
7:        Real *8, Parameter :: theta = 0.5D0 !theta parameter
8:        Real *8, Parameter :: alpha = 0.15D0 !BPR parameter
9:        Real *8, Parameter :: beta = 4.0D0 !BPR parameter
10:       Real *8, Parameter :: lengtht = 60D0 !length of period (minutes)
11:       Real *8, Parameter :: ro = 2D0 !sra parameter
12:       Real *8, Parameter :: gamma = 0.01D0 !sra parameter
13:       Integer s(link), e(link), ls(node, node) !start and end node and link number
14:       Integer ori(nod), des(nod) !origin, destination of od pairs
15:       Real *8 yde(nod) !od demand of od pairs
16:       Real t(link), cap(link) !t:free-flow travel time,cap: capacity of link
17:       Real *8 erh(link), p(node, node) !erh: stoch3 parameter, p:shortest route
18:       Integer headnode(0:node), anode(link), alink(link) ! adjacent nodes and links
19:       Real *8 ps(link), pe(link)
20:   !ps,pe: shortest route from each origin to start and end node of the link
21:       Integer ss(node, link), ee(node, link)
22:   !ss: Start node,ee: end node in arranged shortest route
23:       Real *8 a(link), wl(link), wn(node)
24:   !a:link likelihood,w:link weight,wn:node weight
25:       Integer omega(node, link), mark(node) !Used to check STOCH3-efficient route
26:       Real *8 xo(link) !xo: link traffic flow at SUE
27:       Real *8 tt(link) !tt: link travel time at SUE
28:       Real *8 x(link), xx(node)
29:   !x: link travel is assigned, xx:total node travel in STOCH3's Algorithm
30:       Real *8 tx(link) !tx:derivative of t respect to x
31:   !Variable used to calculate derivative of u respect to t and Detivative of x respect to s
32:       Real *8 gradgt(link, link) !used to calculate derivative of u respect to t
33:       Real *8 part1(link, link), part2(link, link)
34:       Real *8 www, pivot, part3(link, 2*link)
35:       Real *8 grad(link, link) !Grad: Detivative of x respect to s
36:       Real *8 sij(link), yij(link), yijrs(link), b(node, node) !sij:total eliminated flow
37:       Real *8 xafter(link) !link travel flow after sensitivity analysis
38:       Real *8 xsueafter(link)
39:       Real *8 xijrs(link), xijgh1(link, link), xijgh(link, link) ! Used to calculate xijgh
40:       Real *8 lamda, rg1, rg2, sij1(link), rg(link), maxrg
41:   !*****************************************************
42:
43:       Open (1, File='1.network.txt', Action='read')
44:       Open (2, File='2.linkparameter.txt', Action='read')
45:       Open (3, File='3.demand.txt', Action='read')
46:       Open (11, File='19.CalculationTime(SA).txt', Action='write')
47:       Do i = 1, link
48:         Read (1, *) s(i), e(i), j !s(i):start node, e(i):end node
49:         Read (2, *) t(i), cap(i) !t(i):free flow traveltime, cap(i):Capacity of link i
```

```
50:           erh(i) = 1.5
51:         End Do
52:         Do i = 1, nod
53:           Read (3, *) ori(i), des(i), yde(i)
54:         End Do
55:     !make network
56:         ls = 0.0
57:         Do i = 1, link
58:           ls(s(i), e(i)) = i
59:         End Do
60:     ! create adjacent nodes and links
61:         headnode(0) = 0
62:         count = 0
63:         Do i = 1, node
64:           Do j = 1, link
65:             If (i==s(j)) Then
66:               count = count + 1
67:               anode(count) = e(j)
68:               alink(count) = j
69:             End If
70:           End Do
71:           headnode(i) = count
72:         End Do
73:
74:         Write (6, *) 'theta', theta
75:         Write (6, *) 'alpha', alpha
76:         Write (6, *) 'beta', beta
77:
78:         Open (12, File='12.xij.txt')
79:         Do i = 1, link
80:           Read (12, *) xo(i)
81:         End Do
82:         Do i = 1, link
83:           tt(i) = t(i)*(1+alpha*((xo(i)/cap(i))**beta)) !BPR calculation
84:         End Do
85:     !STEP2
86:         omega = 0
87:         mark = 0
88:         gradgt = 0.0
89:         sij = 0.0
90:         xijrs = 0.0
91:         l = 0.0
92:     !******** Dijkstra method calculation ****************
93:         Call dijikstra(p, t)
94:     !***** Calculation of link likeli-hood *********************
95:         a = 0.0
96:         Do i = 1, link
97:           a(i) = exp(-theta*(tt(i)))
98:         End Do
99:     !********STOCH3 algorithm with each OD pair*****************************
100:        Do ioi = 1, nod
101:
```

```
102:          io = ori(ioi)
103:          xx = 0.0
104:          xx(des(ioi)) = yde(ioi)
105:
106:          If (mark(io)==0) Then
107:            mark(io) = 1
108:            Do i = 1, link
109:              ps(i) = p(io, s(i))
110:              pe(i) = p(io, e(i))
111:            End Do
112:            Do i = 1, link
113:              If (pe(i)>ps(i)) Then
114:                If ((1+erh(i))*(pe(i)-ps(i))>=t(i)) Then
115:                  omega(ori(ioi), i) = 1
116:                End If
117:              End If
118:            End Do
119:  !*****sort shortest path from current node*****************************
120:            Do i = 1, link
121:              k = minloc(pe, 1)
122:              pe(k) = 10000001.0
123:              ss(ori(ioi), i) = s(k)
124:              ee(ori(ioi), i) = e(k)
125:            End Do
126:
127:          End If
128:
129:  !***** Calculation of link weight and node weight*********************
130:          wl = 0.0
131:          wn = 0.0
132:          Do i = 1, link !Ps(i)Calculate in ascending sequence
133:            nn = ls(ss(ori(ioi),i), ee(ori(ioi),i)) !NN is the link number
134:            wn(io) = 1.0
135:            If (omega(ori(ioi),nn)==1) Then
136:              wl(nn) = wn(ss(ori(ioi),i))*a(nn) !Link weight
137:              wn(ee(ori(ioi),i)) = wn(ee(ori(ioi),i)) + wl(nn) !Node weight
138:            End If
139:          End Do
140:  !******** Assign trip volume************
141:          x = 0.0
142:          Do i = 1, link
143:            ii = link + 1 - i !Ps(i)Calculate in descending order
144:            nn = ls(ss(ori(ioi),ii), ee(ori(ioi),ii)) !NN is the link number
145:            If (wn(ee(ori(ioi),ii))==0) Go To 100 !Prevents division by 0
146:            If (omega(ori(ioi),nn)==1) Then
147:              x(nn) = xx(ee(ori(ioi),ii))*wl(nn)/wn(ee(ori(ioi),ii))
148:              xx(ss(ori(ioi),ii)) = xx(ss(ori(ioi),ii)) + x(nn)
149:            End If
150:  100     End Do
151:  !       write (*,*) 'finish_xijrs'
152:  !************************************************************
153:          xijrs = x
```

```
154:        xijgh1 = 0.0
155:  !************
156:        Do kk = 1, link
157:          If (xijrs(kk)/=0) Then
158:            io = e(kk)
159:  !***** Calculation of link weight and node weight*********************
160:            wl = 0.0
161:            wn = 0.0
162:            Do i = 1, link !Ps(i)Calculate in ascending sequence
163:              nn = ls(ss(ori(ioi),i), ee(ori(ioi),i)) !NN is the link number
164:              wn(io) = 1.0
165:              If (omega(ori(ioi),nn)==1) Then
166:                wl(nn) = wn(ss(ori(ioi),i))*a(nn) !Link weight
167:                wn(ee(ori(ioi),i)) = wn(ee(ori(ioi),i)) + wl(nn) !Node weight
168:              End If
169:            End Do
170:  !******** Assign trip volume*************
171:            x = 0.0
172:            xx = 0.0
173:            xx(des(ioi)) = xijrs(kk)
174:            Do i = 1, link
175:              ii = link + 1 - i !Ps(i)Calculate in descending order
176:              nn = ls(ss(ori(ioi),ii), ee(ori(ioi),ii)) !NN is the link number
177:              If (wn(ee(ori(ioi),ii))==0) Go To 110 !Prevents division by 0
178:              If (omega(ori(ioi),nn)==1) Then
179:                x(nn) = xx(ee(ori(ioi),ii))*wl(nn)/wn(ee(ori(ioi),ii))
180:                xx(ss(ori(ioi),ii)) = xx(ss(ori(ioi),ii)) + x(nn)
181:              End If
182:  110         End Do
183:  !***************************
184:            Do j = 1, link
185:              xijgh1(kk, j) = x(j)
186:            End Do
187:          End If
188:        End Do
189:
190:        Do i = 1, link
191:          Do j = 1, link
192:            If (xijgh1(i,j)/=0) Then
193:              sij(j) = sij(j) + xijgh1(i, j)/lengtht*tt(i)
194:            End If
195:          End Do
196:        End Do
197:        xijgh = 0.0
198:        Do i = 1, link
199:          Do j = 1, link
200:            xijgh(i, j) = xijgh1(i, j) + xijgh1(j, i)
201:          End Do
202:          xijgh(i, i) = xijrs(i)
203:        End Do
204:
205:        Do i = 1, link
```

```
206:            Do j = 1, link
207:              gradgt(i, j) = gradgt(i, j) + theta*(xijrs(i)*xijrs(j)/yde(ioi)- &
208:                xijgh(i,j))
209:            End Do
210:          End Do
211:  !       write (*,*) 'finish_xijgh'
212:        End Do
213:
214:  !*******Record results*****
215:        Call cpu_time(v)
216:        Write (11, *) 'Calculation time 1 (second)', v
217:
218:        Open (22, File='22.gradgt.txt')
219:        Do i = 1, link
220:          Write (22, *) gradgt(i, :)
221:        End Do
222:        Write (*, *) 'finish_gradgt'
223:
224:  !***** Calculating derivitives of implicit BPR function*****************
225:        Do i = 1, link
226:          tx(i) = t(i)*beta*alpha*(xo(i)**(beta-1))/(cap(i)**beta)
227:        End Do
228:        Open (25, File='25.tx.txt')
229:        Do i = 1, link
230:          Write (25, *) tx(i)
231:        End Do
232:  !*******Calculation Gradient**********
233:        part1 = 0.0
234:        part2 = 0.0
235:        part3 = 0.0
236:        Do i = 1, link
237:          Do j = 1, link
238:            part1(i, j) = gradgt(i, j)*tx(j)
239:          End Do
240:        End Do
241:
242:        Do i = 1, link
243:          Do j = 1, link
244:            If (i==j) Then
245:              part2(i, j) = 1 - part1(i, j)
246:            Else
247:              part2(i, j) = -part1(i, j)
248:            End If
249:          End Do
250:        End Do
251:        Do i = 1, link
252:          Do j = 1, link
253:            part3(i, j) = part2(i, j)
254:          End Do
255:        End Do
256:
257:        Do i = 1, link
```

```
258:          Do j = 1, link
259:            part3(i, j+link) = 0
260:            part3(i, i+link) = 1
261:          End Do
262:        End Do
263:
264:        Do k = 1, link
265:          pivot = part3(k, k)
266:          If (pivot==0) Then
267:            i = k + 1
268:            Do While (pivot==0 .And. i<link)
269:              pivot = part3(i, k)
270:              i = i + 1
271:            End Do
272:            If (pivot==0) Then
273:              Stop
274:            Else
275:              Do j = 1, 2*link
276:                www = part3(k, j)
277:                i = i - 1
278:                part3(k, j) = part3(i, j)
279:                part3(i, j) = www
280:              End Do
281:            End If
282:          End If
283:          Do j = 1, link*2
284:            part3(k, j) = part3(k, j)/pivot
285:          End Do
286:
287:          Do i = 1, link
288:            If (i/=k) Then
289:              www = part3(i, k)
290:              Do j = 1, link*2
291:                part3(i, j) = part3(i, j) - www*part3(k, j)
292:              End Do
293:            End If
294:          End Do
295:        End Do
296:
297:        part2 = 0.0
298:        Do i = 1, link
299:          Do j = 1, link
300:            part2(i, j) = part3(i, j+link)
301:          End Do
302:        End Do
303:        grad = 0
304:        grad = -matmul(part2, part1)
305:
306:        Open (26, File='26.gradxs.txt')
307:        Do i = 1, link
308:          Write (26, *) grad(i, :)
309:        End Do
```

```
310:
311:       Write (*, *) 'finish_Gradient'
312:   !******STEP3:  SOLVING  FIXED-PONT  PROBLEM  OF  ELIMINATED
       FLOW*****************
313:       l = 0
314:       maxrg = 1.0D0
315:       xijgh1 = 0.0
316:       Do While (maxrg>0.0001)
317:         l = l + 1
318:         xsueafter = 0.0D0
319:
320:         xsueafter = matmul(grad, sij)
321:         Do i = 1, link
322:           xsueafter(i) = xo(i) + xsueafter(i)
323:           tt(i) = t(i)*(1+alpha*(((xsueafter(i)-sij(i))/cap(i))**beta))
324:           a(i) = exp(-theta*(tt(i)))
325:         End Do
326:         sij1 = 0.0D0
327:         Do ioi = 1, nod
328:   !***** Calculation of link weight and node weight*********************
329:           wl = 0.0
330:           wn = 0.0
331:           Do i = 1, link !Ps(i)Calculate in ascending sequence
332:             nn = ls(ss(ori(ioi),i), ee(ori(ioi),i)) !NN is the link number
333:             wn(ori(ioi)) = 1.0
334:             If (omega(ori(ioi),nn)==1) Then
335:               wl(nn) = wn(ss(ori(ioi),i))*a(nn) !Link weight
336:               wn(ee(ori(ioi),i)) = wn(ee(ori(ioi),i)) + wl(nn) !Node weight
337:             End If
338:           End Do
339:   !******** Assign trip volume************
340:           x = 0.0
341:           xx = 0
342:           xx(des(ioi)) = yde(ioi)
343:   !                     write (*,*) 'finish_xijrs',xx(des(ioi))
344:           Do i = 1, link
345:             ii = link + 1 - i !Ps(i)Calculate in descending order
346:             nn = ls(ss(ori(ioi),ii), ee(ori(ioi),ii)) !NN is the link number
347:             If (wn(ee(ori(ioi),ii))==0) Go To 120 !Prevents division by 0
348:             If (omega(ori(ioi),nn)==1) Then
349:               x(nn) = xx(ee(ori(ioi),ii))*wl(nn)/wn(ee(ori(ioi),ii))
350:               xx(ss(ori(ioi),ii)) = xx(ss(ori(ioi),ii)) + x(nn)
351:             End If
352:   120      End Do
353:   !*************************************************************
354:           xijrs = x
355:           xijgh1 = 0.0
356:   !************
357:           Do kk = 1, link
358:             If (xijrs(kk)/=0) Then
359:               io = e(kk)
360:   !***** Calculation of link weight and node weight*********************
```

```
361:              wl = 0.0
362:              wn = 0.0
363:              Do i = 1, link !Ps(i)Calculate in ascending sequence
364:                nn = ls(ss(ori(ioi),i), ee(ori(ioi),i)) !NN is the link number
365:                wn(io) = 1.0
366:                If (omega(ori(ioi),nn)==1) Then
367:                  wl(nn) = wn(ss(ori(ioi),i))*a(nn) !Link weight
368:                  wn(ee(ori(ioi),i)) = wn(ee(ori(ioi),i)) + wl(nn) !Node weight
369:                End If
370:              End Do
371:  !******** Assign trip volume************
372:              x = 0.0
373:              xx = 0.0
374:              xx(des(ioi)) = xijrs(kk)
375:              Do i = 1, link
376:                ii = link + 1 - i !Ps(i)Calculate in descending order
377:                nn = ls(ss(ori(ioi),ii), ee(ori(ioi),ii)) !NN is the link number
378:                If (wn(ee(ori(ioi),ii))==0) Go To 130 !Prevents division by 0
379:                If (omega(ori(ioi),nn)==1) Then
380:                  x(nn) = xx(ee(ori(ioi),ii))*wl(nn)/wn(ee(ori(ioi),ii))
381:                  xx(ss(ori(ioi),ii)) = xx(ss(ori(ioi),ii)) + x(nn)
382:                End If
383:  130          End Do
384:  !**************************
385:              Do j = 1, link
386:                xijgh1(kk, j) = x(j)
387:              End Do
388:            End If
389:          End Do
390:          Do i = 1, link
391:            Do j = 1, link
392:              If (xijgh1(i,j)/=0) Then
393:                sij1(j) = sij1(j) + xijgh1(i, j)/lengtht*tt(i)
394:              End If
395:            End Do
396:          End Do
397:
398:        End Do
399:        If (l==1) Then
400:          lamda = 1.0D0
401:        Else
402:          rg2 = 0
403:          Do i = 1, link
404:            rg2 = rg2 + (sij(i)-sij1(i))**2.0
405:          End Do
406:          rg2 = sqrt(rg2)
407:          If (rg2>=rg1) Then
408:            lamda = lamda + ro
409:          Else
410:            lamda = lamda + gamma
411:          End If
412:        End If
```

```
413:          maxrg = 0.0D0
414:          Do i = 1, link
415:            rg(i) = sij1(i) - sij(i)
416:            sij1(i) = sij(i) + rg(i)/lamda
417:            If (abs(rg(i))>maxrg) Then
418:              maxrg = abs(rg(i))
419:            End If
420:          End Do
421:          rg1 = rg2
422:          sij = sij1
423:          Write (6, *) l, maxrg
424:        End Do
425:  !Calculate link travel flow, residual link flow and link travel time at the semi-DTA
426:
427:        xsueafter = matmul(grad, sij)
428:        Do i = 1, link
429:          xsueafter(i) = xo(i) + xsueafter(i)
430:          xafter(i) = max((xsueafter(i)-sij(i)), 0D0)
431:          tt(i) = t(i)*(1+alpha*(((xsueafter(i)-sij(i))/cap(i))**beta))
432:          a(i) = exp(-theta*(tt(i)))
433:        End Do
434:
435:        yij = 0.0
436:        b = 0.0
437:  !********STOCH3 algorithm with each OD pair*****************************
438:        Do ioi = 1, nod
439:          io = ori(ioi)
440:  !***** Calculation of link weight and node weight*********************
441:          wl = 0.0
442:          wn = 0.0
443:          Do i = 1, link !Ps(i)Calculate in ascending sequence
444:            nn = ls(ss(ori(ioi),i), ee(ori(ioi),i)) !NN is the link number
445:            wn(io) = 1.0
446:            If (omega(ori(ioi),nn)==1) Then
447:              wl(nn) = wn(ss(ori(ioi),i))*a(nn) !Link weight
448:              wn(ee(ori(ioi),i)) = wn(ee(ori(ioi),i)) + wl(nn) !Node weight
449:            End If
450:          End Do
451:  !******** Assign trip volume************
452:          x = 0.0
453:          xx = 0.0
454:          xx(des(ioi)) = yde(ioi)
455:          Do i = 1, link
456:            ii = link + 1 - i !Ps(i)Calculate in descending order
457:            nn = ls(ss(ori(ioi),ii), ee(ori(ioi),ii)) !NN is the link number
458:            If (wn(ee(ori(ioi),ii))==0) Go To 140 !Prevents division by 0
459:            If (omega(ori(ioi),nn)==1) Then
460:              x(nn) = xx(ee(ori(ioi),ii))*wl(nn)/wn(ee(ori(ioi),ii))
461:              xx(ss(ori(ioi),ii)) = xx(ss(ori(ioi),ii)) + x(nn)
462:            End If
463:  140     End Do
464:  !************************************************************
```

```
465:          xijrs = x
466:          yijrs = 0.0
467:          Do i = 1, link
468:            yijrs(i) = xijrs(i)/lengtht*tt(i)
469:            yij(i) = yij(i) + yijrs(i)
470:          End Do
471:
472:          Do i = 1, link
473:            If (yijrs(i)/=0) Then
474:              b(e(i), des(ioi)) = b(e(i), des(ioi)) + yijrs(i)
475:            End If
476:          End Do
477:        End Do
478:
479:        Open (28, File='28.sij.txt')
480:        Do i = 1, link
481:          Write (28, *) sij(i)
482:        End Do
483:        Open (29, File='29.yij.txt')
484:        Do i = 1, link
485:          Write (29, *) yij(i)
486:        End Do
487:        Open (37, File='37.xSUEafter.txt')
488:        Do i = 1, link
489:          Write (37, *) xsueafter(i)
490:        End Do
491:
492:        Open (38, File='38.xafter(subtracted).txt')
493:
494:        Do i = 1, link
495:          Write (38, *) xafter(i)
496:        End Do
497:
498:        Open (68, File='68.FlowPropagation.txt')
499:        Do i = 1, node
500:          b(i, i) = 0
501:          Write (68, *)(b(i,j), j=1, node)
502:        End Do
503:
504:        Call cpu_time(v)
505:        Write (11, *) 'Calculation time 2 (second)', v
506:
507:    !*****SUBROUTINE************************************************************
508:      Contains
509:        Subroutine dijikstra(p, t)
510:          Real, Intent (In) :: t(link)
511:          Real *8, Intent (Out) :: p(node, node)
512:          Real *8 :: c(node), cmin(node), fin_io(node)
513:
514:          Do j = 1, node
515:            io = j !io is the source node number
516:            c(:) = 10000000.0 !set partial route cost • ‡
```

```
517:          c(io) = 0.0 !set the cost of the source node to 0
518:          cmin(:) = 10000000.0
519:          fin_io(:) = 0 !array for storing the nodes for which calculation has been completed
520:          Do While (minval(cmin,1)<50000000)
521:            Do iii = headnode(io-1) + 1, headnode(io)
522:              i = ls(io, anode(iii))
523:              If (c(io)+t(i)<=c(e(i))) Then
524:                c(e(i)) = c(io) + t(i) !update with the next link cost with the node
525:                cmin(e(i)) = c(io) + t(i)
526:              End If
527:            End Do
528:            ncj = minloc(cmin, 1) !identify the location of cmin in the array
529:            cmin(ncj) = 50000000.0 !ineffectiveness of extracted cmin
530:  !**** calculation end judgment (end when all cmin are disabled) ***********
531:            If (minval(cmin,1)==50000000) Then
532:              Go To 100
533:            End If
534:  !******!move to the next node******************
535:            io = ncj
536:            If (fin_io(ncj)==1) Then !when calculation is completed, move to another node
537:              inot = minloc(fin_io, 1)
538:              io = inot
539:            End If
540:            fin_io(io) = 1
541:          End Do
542:  100       Continue
543:  !******record shortest route results***********************************
544:          p(j, :) = c(:)
545:        End Do
546:
547:      End Subroutine
548:  !*****FINISH SUBROUTINE***********************************
549:    End Program
```

Appendix G. Fortran Coding for Chapter 5

The following will detail the programs for the proposed algorithm applied to the small network in chaper 5.

G.1 Sensitivity analysis method for the CNL SUE model

```
1:      ! *******Sensitivity analysis for cross-nested logit model**********
2:      ! **************changing parameters of toll fare*********************
3:          Integer l, i, j, k, ioi, io, count
4:          Integer, Parameter :: node = 5 !number of node
5:          Integer, Parameter :: link = 7 !number of link
6:          Integer, Parameter :: nod = 1 !number of od pairs
7:          Real *8, Parameter :: eee = 1.0D-3 !loop termination condition
8:          Real *8, Parameter :: theta = 0.5D0 !logit parameter
9:          Real *8, Parameter :: alpha = 1.0D0 !bpr parameter
10:         Real *8, Parameter :: beta = 2.0D0 !bpr parameter
11:         Real *8, Parameter :: muy = 0.5D0 !cross-nested logit parameter
12:         Real *8, Parameter :: tv = 50D0 !The value of time
13:         Real *8, Parameter :: ro = 2D0 !sra parameter
14:         Real *8, Parameter :: gamma = 0.01D0 !sra parameter
15:         Integer s(link), e(link), ls(node, node) !start and end node and link number
16:         Integer ori(nod), des(nod) !origin, destination of od pairs
17:         Real *8 yde(nod) !od demand of od pairs
18:         Real *8 toll(link) !toll for each link
19:         Real t(link), cap(link) !t:free-flow travel time,cap: capacity of link
20:         Real *8 erh(link), p(node, node) !erh: stoch3 parameter, p:shortest route
21:         Integer headnode(0:node), anode(link), alink(link) ! adjacent nodes and links
22:         Real *8 ps(link), pe(link)
23:     !ps,pe: shortest path from each origin to start and end node of the link
24:         Real *8 a(link) !a:link likelihood
25:         Integer omega(nod, link), mark(link)
26:     !omega(nod,link): marking efficient links of each od pair
27:     !mark(link): marking a route at each time reaching the destination in dfs algorithm
28:         Real *8 xo(link), xn(link) !xo: link traffic flow after, xn:link traffic flow before
29:         Real *8 tt(link) !tt: link travel time after update
30:         Real *8 maxrg, rg(link) !used in msa method
31:         Real *8 x(link) !x: link travel is assigned
32:         Real *8 vn(node), crou, lamda, rg1, rg2
33:     !lamda,rg1,rg2: used in sra method;
34:     !vn(node): node weight, crou: travel cost of a route used in dfs algorithm.
35:         Real *8 u2ijrs(link), dnm, num(link), num1
36:     !num1: used to calculated num(link) (the nominator of gradht 5.21)
37:         Real *8 vijghrs(link, link)
38:         Real *8 grad(link, link)
39:         Real *8 p1ijghrs(link, link), p2ijghrs(link, link), gradht2(link)
40:     !p1ijghrs,p2ijghrs: the first and second parts of 5.28
41:     !gradht2(link,link): the second part of 5.22;
42:     !grad: The derivative of link flows with respect to eliminated flow
43:         Real *8 tx(link) !tx:derivative of t respect to x
```

```
44:         Real *8 gradht(link, link) !used to calculate derivative of g respect to t
45:         Real *8 part1(link, link), part2(link, link) ! used to calculate inverse matrix
46:         Real *8 www, pivot, part3(link, 2*link) ! used to calculate inverse matrix
47:     !Variables used to calculate second part of 5.46 for all other links (nests)
48:         Real *8 ctoll(link) !the change of toll fare for each link
49:         Real *8 xafter(link) !link travel flow after sensitivity analysis
50:
51:     !c*****************************************************
52:
53:         Open (1, File='1.network.txt', Action='read')
54:         Open (2, File='2.linkparameter.txt', Action='read')
55:         Open (3, File='3.demand.txt', Action='read')
56:         Open (11, File='11.calculationtime.txt', Action='write')
57:         Do i = 1, link
58:           Read (1, *) s(i), e(i), j !s(i):start node, e(i):end node
59:           Read (2, *) t(i), cap(i) !t(i):free flow traveltime, cap(i):capacity of link i
60:           erh(i) = 1.5
61:         End Do
62:         toll = 0.0D0
63:         toll(2) = 500D0
64:         Do i = 1, nod
65:           Read (3, *) ori(i), des(i), yde(i)
66:         End Do
67:     !make network
68:         ls = 0.0
69:         Do i = 1, link
70:           ls(s(i), e(i)) = i
71:         End Do
72:     ! create adjacent nodes and links
73:         headnode(0) = 0
74:         count = 0
75:         Do i = 1, node
76:           Do j = 1, link
77:             If (i==s(j)) Then
78:               count = count + 1
79:               anode(count) = e(j)
80:               alink(count) = j
81:             End If
82:           End Do
83:           headnode(i) = count
84:         End Do
85:
86:         Write (6, *) 'theta', theta
87:         Write (6, *) 'alpha', alpha
88:         Write (6, *) 'beta', beta
89:     ! computing static sue
90:     !********* dijkstra method calculation *****************
91:         Call dijikstra(p, t)
92:
93:     !****** dfs algorithm ******************************
94:         xo = 0
95:         omega = 0.0
```

```
96:   !****** calculation of link likeli-hood *********************
97:         a = 0.0
98:         tt = 0.0
99:         Do i = 1, link
100:          a(i) = exp(-theta*(t(i)))
101:          tt(i) = t(i)
102:        End Do
103:        Do ioi = 1, nod
104:          io = ori(ioi)
105:          Do i = 1, link
106:            ps(i) = p(io, s(i))
107:            pe(i) = p(io, e(i))
108:          End Do
109:
110:  !****** !check STOCH3-efficient routes ******************************
111:        Do i = 1, link
112:  !       If (ps(i)>pe(i)) Go To 100
113:  !       If ((1+erh(i))*(pe(i)-ps(i))>=t(i)) Then
114:          omega(ioi, i) = 1
115:  !       End If
116:        End Do
117:  !******running the dfs algorithm from origin node r to destination node s ******
118:        vn = 0.0
119:        vn(ori(ioi)) = 1
120:        j = ori(ioi)
121:        k = 0
122:        u2ijrs = 0
123:        Call calxij(1) !Running DFS for the First time
124:        dnm = 0
125:        Do i = 1, link
126:          dnm = dnm + u2ijrs(i)**muy
127:        End Do
128:        num = 0.0
129:        vn = 0.0
130:        vn(ori(ioi)) = 1
131:        j = ori(ioi)
132:        k = 0
133:        Call calxij1(1) !Running DFS for the Second time
134:  !************************************************************
135:  ! recording after loop termination
136:        x = 0.0D0
137:        Do i = 1, link
138:          omega(ioi, i) = 0
139:          x(i) = yde(ioi)*num(i)/dnm
140:          If (x(i)/=0) Then
141:            omega(ioi, i) = 1
142:          End If
143:          xo(i) = xo(i) + x(i)
144:        End Do
145:      End Do
146:  !********starting calculation round*************************************
147:      maxrg = 1
```

```
148:        l = 0
149:        mark = 0
150:        rg1 = 0
151:        Do While (maxrg>eee)
152:  !*****Link travel time and link impedances update**********************
153:          l = l + 1
154:          Do i = 1, link
155:            tt(i) = t(i)*(1+alpha*((xo(i)/cap(i))**beta)) + toll(i)/tv !bpr calculation
156:            a(i) = exp(-theta*(tt(i)))
157:          End Do
158:  !******Direction Finding******
159:          xn = 0.0
160:          Do ioi = 1, nod
161:            vn = 0.0
162:            vn(ori(ioi)) = 1
163:            j = ori(ioi)
164:            k = 0
165:            u2ijrs = 0
166:            Call calxij(1)
167:            dnm = 0
168:            Do i = 1, link
169:              dnm = dnm + u2ijrs(i)**muy
170:            End Do
171:            num = 0.0
172:            vn = 0.0
173:            vn(ori(ioi)) = 1
174:            j = ori(ioi)
175:            k = 0
176:            Call calxij1(1)
177:  !***************************
178:            Do i = 1, link
179:              xn(i) = xn(i) + yde(ioi)*num(i)/dnm
180:            End Do
181:  !****************************************
182:          End Do
183:  !****************************************
184:  ! convergence determination calculation
185:          If (l==1) Then
186:            lamda = 1.0D0
187:          Else
188:            rg2 = 0
189:            Do i = 1, link
190:              rg2 = rg2 + (xn(i)-xo(i))**2.0
191:            End Do
192:            rg2 = sqrt(rg2)
193:            If (rg2>=rg1) Then
194:              lamda = lamda + ro
195:            Else
196:              lamda = lamda + gamma
197:            End If
198:          End If
199:          rg = 0.0
```

```
200:          Do i = 1, link
201:            rg(i) = xn(i) - xo(i)
202:            xn(i) = xo(i) + rg(i)/dble(lamda)
203:          End Do
204:          xo = xn
205:          maxrg = maxval(abs(rg))
206:          rg1 = rg2
207:          Write (*, *) 'round', l, 'maxrg', maxrg
208:        End Do
209:    !***********results************************
210:        Write (11, *) 'round', l, 'computation gap', maxrg
211:        Call cpu_time(v)
212:        Write (11, *) 'time of calculation (second) ', v
213:        Open (12, File='12.xij.txt')
214:        Do i = 1, link
215:          Write (12, *) xo(i)
216:        End Do
217:        Do i = 1, link
218:          tt(i) = t(i)*(1+alpha*((xo(i)/cap(i))**beta)) + toll(i)/tv !bpr calculation
219:        End Do
220:        Open (13, File='13.tij.txt')
221:        Do i = 1, link
222:          Write (13, *) tt(i)
223:        End Do
224:        Open (10, File='10.pijrs.txt')
225:    ! Calculating gradient matrix
226:    !****** calculating derivitives of explicit bpr function*****************
227:        tx = 0.0D0
228:        Do i = 1, link
229:          tx(i) = t(i)*beta*alpha*(xo(i)**(beta-1))/(cap(i)**beta)
230:        End Do
231:        Open (21, File='21.tx.txt')
232:        Do i = 1, link
233:          Write (21, *) tx(i)
234:        End Do
235:        Write (*, *) 'finish_tx'
236:        gradht = 0.0
237:    !****** calculating derivitives of g with respect to t*****************
238:        a = 0.0
239:        Do i = 1, link
240:          a(i) = exp(-theta*(tt(i)))
241:        End Do
242:        Do ioi = 1, nod
243:          io = ori(ioi)
244:          vn = 0.0
245:          vn(ori(ioi)) = 1
246:          j = ori(ioi)
247:          k = 0
248:          u2ijrs = 0
249:          vijghrs = 0.0D0
250:          Call calxij2(1)
251:          dnm = 0
```

```
252:         Do i = 1, link
253:           dnm = dnm + u2ijrs(i)**muy
254:         End Do
255:         num = 0.0
256:         vn = 0.0
257:         vn(ori(ioi)) = 1
258:         j = ori(ioi)
259:         k = 0
260:         p1ijghrs = 0
261:         p2ijghrs = 0
262:         gradht2 = 0.0
263:         Call calxij3(1)
264:         Do i = 1, link
265:           Do j = 1, link
266:             gradht2(i) = gradht2(i) - theta*u2ijrs(j)**(muy-1)*vijghrs(j, i)
267:           End Do
268:         End Do
269:   ! recording after loop termination
270:   !*******calculating node traffic flow, link proportion, xijgh and gradht*****
271:         x = 0.0D0
272:         Do i = 1, link
273:           If (omega(ioi,i)/=0) Then
274:             Write (10, *) 'OD', ioi, 'Link', i, (num(i))/dnm
275:             x(i) = yde(ioi)*num(i)/dnm
276:             If (x(i)==0) Then
277:               omega(ioi, i) = 0
278:             End If
279:             Do j = 1, link
280:               If (omega(ioi,j)/=0 .And. dnm/=0) Then
281:                 gradht(i, j) = gradht(i, j) + yde(ioi)*((-theta/muy)*(p1ijghrs &
282:                   (i,j)+p2ijghrs(i,j))/dnm-gradht2(j)*(num(i)/dnm)/dnm)
283:               End If
284:             End Do
285:           End If
286:         End Do
287:
288:       End Do
289:
290:   !c***********Results************************
291:       Call cpu_time(v)
292:       Write (11, *) 'Calculation time 1 (second)', v
293:
294:   !*******Calculation Gradient**********
295:       part1 = 0.0
296:       part2 = 0.0
297:       part3 = 0.0
298:       Do i = 1, link
299:         Do j = 1, link
300:           part1(i, j) = gradht(i, j)*tx(j)
301:         End Do
302:       End Do
303:
```

```
304:        Do i = 1, link
305:          Do j = 1, link
306:            If (i==j) Then
307:              part2(i, j) = 1 - part1(i, j)
308:            Else
309:              part2(i, j) = -part1(i, j)
310:            End If
311:          End Do
312:        End Do
313:        Do i = 1, link
314:          Do j = 1, link
315:            part3(i, j) = part2(i, j)
316:          End Do
317:        End Do
318:
319:        Do i = 1, link
320:          Do j = 1, link
321:            part3(i, j+link) = 0
322:            part3(i, i+link) = 1
323:          End Do
324:        End Do
325:
326:        Do k = 1, link
327:          pivot = part3(k, k)
328:          If (pivot==0) Then
329:            i = k + 1
330:            Do While (pivot==0 .And. i<link)
331:              pivot = part3(i, k)
332:              i = i + 1
333:            End Do
334:            If (pivot==0) Then
335:              Stop
336:            Else
337:              Do j = 1, 2*link
338:                www = part3(k, j)
339:                i = i - 1
340:                part3(k, j) = part3(i, j)
341:                part3(i, j) = www
342:              End Do
343:            End If
344:          End If
345:          Do j = 1, link*2
346:            part3(k, j) = part3(k, j)/pivot
347:          End Do
348:
349:          Do i = 1, link
350:            If (i/=k) Then
351:              www = part3(i, k)
352:              Do j = 1, link*2
353:                part3(i, j) = part3(i, j) - www*part3(k, j)
354:              End Do
355:            End If
```

```
356:          End Do
357:        End Do
358:
359:        part2 = 0.0
360:        Do i = 1, link
361:          Do j = 1, link
362:            part2(i, j) = part3(i, j+link)
363:          End Do
364:        End Do
365:        part1 = 0.0D0
366:        Do i = 1, link
367:          Do j = 1, link
368:            If (toll(j)/=0) Then
369:              part1(i, j) = gradht(i, j)/tv
370:            End If
371:          End Do
372:        End Do
373:
374:        grad = 0
375:        grad = matmul(part2, part1)
376:        Open (26, File='26.grad.txt')
377:        Do i = 1, link
378:          Write (26, *)(grad(i,j), j=1, link)
379:        End Do
380:
381:        Write (*, *) 'finish_Gradient'
382:        Call cpu_time(v)
383:        Write (11, *) 'Calculation time 2 (second)', v
384:  !c******Calculate link travel flow and link travel time after Sensitivity
      analysis******************
385:        Open (36, File='36.xafter.txt')
386:        Do i = 1, 10
387:          ctoll = 0.0D0
388:          Do j = 1, link
389:            If (toll(j)==0) Then
390:              ctoll(j) = 0
391:            Else
392:              ctoll(j) = (100.0D0-toll(j)) + 100*(i-1)
393:            End If
394:          End Do
395:          xafter = matmul(grad, ctoll)
396:          Do j = 1, link
397:            xafter(j) = xo(j) + xafter(j)
398:          End Do
399:          Do j = 1, link
400:            Write (36, *) i*100, xafter(j)
401:          End Do
402:        End Do
403:
404:        Call cpu_time(v)
405:        Write (11, *) 'Calculation time 3 (second)', v
406:  !c******SUBROUTINE*************************************************************
```

```
407:      Contains
408:        Subroutine dijikstra(p, t)
409:          Real, Intent (In) :: t(link)
410:          Real *8, Intent (Out) :: p(node, node)
411:          Real *8 :: c(node), cmin(node), fin_io(node)
412:
413:          Do j = 1, node
414:            io = j !io is the source node number
415:            c(:) = 10000000.0 !set partial path cost • ‡
416:            c(io) = 0.0 !set the cost of the source node to 0
417:            cmin(:) = 10000000.0
418:            fin_io(:) = 0 !array for storing the nodes for which calculation has been completed
419:            Do While (minval(cmin,1)<50000000)
420:              Do iii = headnode(io-1) + 1, headnode(io)
421:                i = ls(io, anode(iii))
422:                If (c(io)+t(i)<=c(e(i))) Then
423:                  c(e(i)) = c(io) + t(i) !update with the next link cost with the node
424:                  cmin(e(i)) = c(io) + t(i)
425:                End If
426:              End Do
427:              ncj = minloc(cmin, 1) !identify the location of cmin in the array
428:              cmin(ncj) = 50000000.0 !ineffectiveness of extracted cmin
429:  !**** calculation end judgment (end when all cmin are disabled) ***********
430:              If (minval(cmin,1)==50000000) Then
431:                Go To 100
432:              End If
433:  !******!move to the next node******************
434:              io = ncj
435:              If (fin_io(ncj)==1) Then !when calculation is completed, move to another node
436:                inot = minloc(fin_io, 1)
437:                io = inot
438:              End If
439:              fin_io(io) = 1
440:            End Do
441:  100       Continue
442:  !******record shortest path results***********************************
443:            p(j, :) = c(:)
444:          End Do
445:
446:        End Subroutine
447:  ! calculate xij with dfs
448:        Recursive Subroutine calxij(dfs)
449:          Integer dfs, iii
450:
451:          If (j==des(ioi)) Then
452:            crou = 0.0
453:            Do i = 1, k
454:              crou = crou + t(mark(i))
455:            End Do
456:            Do i = 1, k
457:              u2ijrs(mark(i)) = u2ijrs(mark(i)) + (t(mark(i))/crou*vn(j))**( &
458:                1.0D0/muy)
```

```
459:           End Do
460:         Else
461:           Do iii = headnode(j-1) + 1, headnode(j)
462:             If (omega(ioi,alink(iii))/=0) Then
463:               j = e(alink(iii))
464:               k = k + 1
465:               vn(j) = a(alink(iii))*vn(s(alink(iii)))
466:               mark(k) = alink(iii)
467:               Call calxij(dfs+1)
468:               mark(k) = 0
469:               k = k - 1
470:             End If
471:           End Do
472:         End If
473:       End Subroutine
474:   ! calculate xij with dfs
475:       Recursive Subroutine calxij1(dfs)
476:         Integer dfs, iii
477:
478:         If (j==des(ioi)) Then
479:           crou = 0.0
480:           Do i = 1, k
481:             crou = crou + t(mark(i))
482:           End Do
483:           num1 = 0.0
484:           Do i = 1, k
485:             num1 = num1 + (t(mark(i))/crou*vn(j))**(1.0D0/muy)*(u2ijrs(mark(i) &
486:               )**(muy-1.0D0))
487:           End Do
488:           Do i = 1, k
489:             num(mark(i)) = num(mark(i)) + num1
490:           End Do
491:         Else
492:           Do iii = headnode(j-1) + 1, headnode(j)
493:             If (omega(ioi,alink(iii))/=0) Then
494:               j = e(alink(iii))
495:               k = k + 1
496:               vn(j) = a(alink(iii))*vn(s(alink(iii)))
497:               mark(k) = alink(iii)
498:               Call calxij1(dfs+1)
499:               mark(k) = 0
500:               k = k - 1
501:             End If
502:           End Do
503:         End If
504:       End Subroutine
505:   ! calculate gradient with dfs
506:       Recursive Subroutine calxij2(dfs)
507:         Integer dfs, iii
508:
509:         If (j==des(ioi)) Then
510:           crou = 0.0
```

```
511:          Do i = 1, k
512:            crou = crou + t(mark(i))
513:          End Do
514:          Do i = 1, k
515:            u2ijrs(mark(i)) = u2ijrs(mark(i)) + (t(mark(i))/crou*vn(j))**( &
516:              1.0D0/muy)
517:          End Do
518:          Do i = 1, k
519:            Do ii = 1, k
520:              vijghrs(mark(i), mark(ii)) = vijghrs(mark(i), mark(ii)) + &
521:                (t(mark(i))/crou*vn(j))**(1.0D0/muy)
522:            End Do
523:          End Do
524:
525:        Else
526:          Do iii = headnode(j-1) + 1, headnode(j)
527:            If (omega(ioi,alink(iii))/=0) Then
528:              j = e(alink(iii))
529:              k = k + 1
530:              vn(j) = a(alink(iii))*vn(s(alink(iii)))
531:              mark(k) = alink(iii)
532:              Call calxij2(dfs+1)
533:              mark(k) = 0
534:              k = k - 1
535:            End If
536:          End Do
537:        End If
538:      End Subroutine
539:  ! calculate p1ijghrs,p1ijghrs and grad4 with dfs
540:      Recursive Subroutine calxij3(dfs)
541:        Integer dfs, iii
542:
543:        If (j==des(ioi)) Then
544:          crou = 0.0
545:          Do i = 1, k
546:            crou = crou + t(mark(i))
547:          End Do
548:          num1 = 0.0
549:          Do i = 1, k
550:            num1 = num1 + (t(mark(i))/crou*vn(j))**(1.0D0/muy)*(u2ijrs(mark(i) &
551:              )**(muy-1.0D0))
552:          End Do
553:          Do i = 1, k
554:            num(mark(i)) = num(mark(i)) + num1
555:          End Do
556:
557:          Do i = 1, k
558:            Do ii = 1, k
559:  !p1ijghrs(ij,mn) for link mn is also an element of the route including ij and gh
560:              Do iii = 1, k
561:                p1ijghrs(mark(i), mark(iii)) = p1ijghrs(mark(i), mark(iii)) + &
562:                  (t(mark(ii))/crou*vn(j))**(1.0D0/muy)* &
```

```
563:                 (u2ijrs(mark(ii))**(muy-1.0D0))
564:               End Do
565:  !p1ijghrs(ij,mn) for link mn is not an element of the route including ij but vijghrs(gh, mn).ne.0
566:               Do iii = 1, link
567:                 If (vijghrs(mark(ii),iii)/=0 .And. u2ijrs(mark(ii))/=0) Then
568:                   p2ijghrs(mark(i), iii) = p2ijghrs(mark(i), iii) + &
569:                     (muy-1)*(t(mark(ii))/crou*vn(j))**(1.0D0/muy)* &
570:                     (u2ijrs(mark(ii))**(muy-2.0D0))*vijghrs(mark(ii), iii)
571:                 End If
572:               End Do
573:
574:             End Do
575:           End Do
576:
577:         Else
578:           Do iii = headnode(j-1) + 1, headnode(j)
579:             If (omega(ioi,alink(iii))/=0) Then
580:               j = e(alink(iii))
581:               k = k + 1
582:               vn(j) = a(alink(iii))*vn(s(alink(iii)))
583:               mark(k) = alink(iii)
584:               Call calxij3(dfs+1)
585:               mark(k) = 0
586:               k = k - 1
587:             End If
588:           End Do
589:         End If
590:       End Subroutine
591:
592:     End Program
```

G.2 The q-generalized logit SUE with the implicit route-based approach

```
1:      ! *******static traffic assignment for q-generalized logit model**********
2:      ! ***Based on stoch3-efficient route, dfs algorithm and the sra method ***
3:          Integer l, i, j, k, ioi, io, count
4:          Integer, Parameter :: node = 3 !number of node
5:          Integer, Parameter :: link = 3 !number of link
6:          Integer, Parameter :: nod = 2 !number of OD pairs
7:          Real *8, Parameter :: eee = 1.0D-3 !loop termination condition
8:          Real *8, Parameter :: theta = 2.0D0 !theta parameter
9:          Real *8, Parameter :: alpha = 1.0D0 !BPR parameter
10:         Real *8, Parameter :: beta = 2.0D0 !BPR parameter
11:         Real *8, Parameter :: kiu = 0.5D0 !q parameter
12:         Real *8, Parameter :: ro = 2D0 !sra parameter
13:         Real *8, Parameter :: gamma = 0.01D0 !sra parameter
14:         Integer s(link), e(link), ls(node, node) !start and end node and link number
15:         Integer ori(nod), des(nod) !origin, destination of od pairs
16:         Real *8 yde(nod) !od demand of od pairs
17:         Real t(link), cap(link) !t:free-flow travel time,cap: capacity of link
18:         Real *8 erh(link), p(node, node) !erh: stoch3 parameter, p:shortest route
19:         Integer headnode(0:node), anode(link), alink(link) ! adjacent nodes and links
20:         Real *8 ps(link), pe(link)
21:     !ps,pe: shortest path from each origin to start and end node of the link
22:         Integer omega(nod, link), mark(link)
23:     !omega(nod,link): marking efficient links of each od pair
24:     !mark(link): marking a route at each time reaching the destination in dfs algorithm
25:         Real *8 xo(link), xn(link) !xo: link traffic flow after, xn:link traffic flow before
26:         Real *8 tt(link) !tt: link travel time after update
27:         Real *8 maxrg, g(link) !used in msa method
28:         Real *8 x(link) !x: link travel is assigned
29:         Real *8 num(link), dnm, vn, crou, lamda, rg1, rg2, v
30:     !!******************************************************
31:
32:         Open (1, File='1.network.txt', Action='read')
33:         Open (2, File='2.linkparameter.txt', Action='read')
34:         Open (3, File='3.demand.txt', Action='read')
35:         Open (11, File='11.CalculationTime(SUE).txt', Action='write')
36:         Do i = 1, link
37:           Read (1, *) s(i), e(i), j !s(i):start node, e(i):end node
38:           Read (2, *) t(i), cap(i) !t(i):free flow traveltime, cap(i):Capacity of link i
39:           erh(i) = 1.5
40:         End Do
41:         Do i = 1, nod
42:           Read (3, *) ori(i), des(i), yde(i)
43:         End Do
44:     !make network
45:         ls = 0.0
46:         Do i = 1, link
47:           ls(s(i), e(i)) = i
48:         End Do
49:     ! create adjacent nodes and links
50:         headnode(0) = 0
51:         count = 0
```

```
52:         Do i = 1, node
53:           Do j = 1, link
54:             If (i==s(j)) Then
55:               count = count + 1
56:               anode(count) = e(j)
57:               alink(count) = j
58:             End If
59:           End Do
60:           headnode(i) = count
61:         End Do
62:
63:         Write (6, *) 'theta', theta
64:         Write (6, *) 'alpha', alpha
65:         Write (6, *) 'beta', beta
66:     !step 1: computing static sue
67:     !********* dijkstra method calculation ****************
68:         Call dijikstra(p, t)
69:
70:     !****** dfs algorithm *****************************
71:         xo = 0
72:         omega = 0.0
73:     !***** Creating initial solution based on free-flow travel time***********
74:         tt = 0.0
75:         Do i = 1, link
76:           tt(i) = t(i)
77:         End Do
78:         Do ioi = 1, nod
79:           io = ori(ioi)
80:           Do i = 1, link
81:             ps(i) = p(io, s(i))
82:             pe(i) = p(io, e(i))
83:           End Do
84:
85:     !****** !check STOCH3-efficient routes ****************************
86:           Do i = 1, link
87:             If (ps(i)>pe(i)) Go To 100
88:             If ((1+erh(i))*(pe(i)-ps(i))>=t(i)) Then
89:               omega(ioi, i) = 1
90:             End If
91:     100     End Do
92:     !******running the dfs algorithm from origin node r to destination node s ******
93:           num = 0.0
94:           vn = 0.0
95:           j = ori(ioi)
96:           k = 0
97:           dnm = 0
98:           Call calxij(1)
99:     !************************************************************
100:    !**************************
101:    !Recording after loop termination
102:          Do i = 1, link
103:            omega(ioi, i) = 0
```

```
104:            x(i) = yde(ioi)*num(i)/dnm
105:            If (x(i)/=0) Then
106:              omega(ioi, i) = 1
107:            End If
108:            xo(i) = xo(i) + x(i)
109:          End Do
110:        End Do
111:    !*******starting calculation round*************************************
112:        xn = 0.0
113:        g = 0.0
114:        maxrg = 1
115:        l = 0
116:        mark = 0
117:        Do While (maxrg>eee)
118:    !****Update link travel time
119:          l = l + 1
120:          Do i = 1, link
121:            tt(i) = t(i)*(1+alpha*((xo(i)/cap(i))**beta)) !BPR calculation
122:          End Do
123:    !******Direction Finding******
124:          xn = 0.0
125:          Do ioi = 1, nod
126:            num = 0.0
127:            vn = 0.0
128:            j = ori(ioi)
129:            k = 0
130:            dnm = 0
131:            Call calxij(1)
132:    !****************************
133:            Do i = 1, link
134:              xn(i) = xn(i) + yde(ioi)*num(i)/dnm
135:            End Do
136:    !************************************
137:          End Do
138:    !************************************
139:    !Convergence determination calculation
140:          If (l==1) Then
141:            lamda = 1.0D0
142:          Else
143:            rg2 = 0
144:            Do i = 1, link
145:              rg2 = rg2 + (xn(i)-xo(i))**2.0
146:            End Do
147:            rg2 = sqrt(rg2)
148:            If (rg2>=rg1) Then
149:              lamda = lamda + ro
150:            Else
151:              lamda = lamda + gamma
152:            End If
153:          End If
154:          g = 0.0
155:          Do i = 1, link
```

```
156:            g(i) = xn(i) - xo(i)
157:            xn(i) = xo(i) + g(i)/dble(lamda)
158:          End Do
159:          xo = xn
160:          maxrg = maxval(abs(g))
161:          rg1 = rg2
162:          Write (*, *) 'round', l, 'maxrg', maxrg
163:        End Do
164:    !**********Results************************
165:        Write (11, *) 'Round', l, 'Computation gap', maxrg
166:        Call cpu_time(v)
167:        Write (11, *) 'time of calculation (second) ', v
168:        Open (12, File='12.xij.txt')
169:        Do i = 1, link
170:          Write (12, *) xo(i)
171:        End Do
172:        Do i = 1, link
173:          tt(i) = t(i)*(1+alpha*((xo(i)/cap(i))**beta)) !BPR calculation
174:        End Do
175:        Open (13, File='13.tij.txt')
176:        Do i = 1, link
177:          Write (13, *) tt(i)
178:        End Do
179:
180:    !*****SUBROUTINE*************************************************************
181:      Contains
182:        Subroutine dijikstra(p, t)
183:          Real, Intent (In) :: t(link)
184:          Real *8, Intent (Out) :: p(node, node)
185:          Real *8 :: c(node), cmin(node), fin_io(node)
186:
187:          Do j = 1, node
188:            io = j !io is the source node number
189:            c(:) = 10000000.0 !set partial path cost • ‡
190:            c(io) = 0.0 !set the cost of the source node to 0
191:            cmin(:) = 10000000.0
192:            fin_io(:) = 0 !array for storing the nodes for which calculation has been completed
193:            Do While (minval(cmin,1)<50000000)
194:              Do iii = headnode(io-1) + 1, headnode(io)
195:                i = ls(io, anode(iii))
196:                If (c(io)+t(i)<=c(e(i))) Then
197:                  c(e(i)) = c(io) + t(i) !update with the next link cost with the node
198:                  cmin(e(i)) = c(io) + t(i)
199:                End If
200:              End Do
201:              ncj = minloc(cmin, 1) !identify the location of cmin in the array
202:              cmin(ncj) = 50000000.0 !ineffectiveness of extracted cmin
203:    !**** calculation end judgment (end when all cmin are disabled) ***********
204:              If (minval(cmin,1)==50000000) Then
205:                Go To 100
206:              End If
207:    !******!move to the next node******************
```

```
208:              io = ncj
209:              If (fin_io(ncj)==1) Then !when calculation is completed, move to another node
210:                inot = minloc(fin_io, 1)
211:                io = inot
212:              End If
213:              fin_io(io) = 1
214:            End Do
215:    100       Continue
216:    !******record shortest path results************************************
217:            p(j, :) = c(:)
218:          End Do
219:
220:        End Subroutine
221:
222:    !   Calculate xij with DFS
223:        Recursive Subroutine calxij(dfs)
224:          Integer dfs, iii
225:
226:          If (j==des(ioi)) Then
227:            crou = 0.0
228:            Do i = 1, k
229:              crou = crou + tt(mark(i))
230:            End Do
231:            vn = (1.0D0+(kiu-1.0D0)*(-theta*crou))**(1.0D0/(kiu-1.0D0))
232:            dnm = dnm + vn
233:            Do i = 1, k
234:              num(mark(i)) = num(mark(i)) + vn
235:            End Do
236:          Else
237:            Do iii = headnode(j-1) + 1, headnode(j)
238:              If (omega(ioi,alink(iii))/=0) Then
239:                j = e(alink(iii))
240:                k = k + 1
241:                mark(k) = alink(iii)
242:                Call calxij(dfs+1)
243:                mark(k) = 0
244:                k = k - 1
245:              End If
246:            End Do
247:          End If
248:        End Subroutine
249:
250:      End Program
```

G.3 Sensitivity analysis method for the q-generalized logit SUE model

```
1:    ! *******static traffic assignment for q-generalized logit model**********
2:    ! ***Based on stoch3-efficient route, dfs algorithm and the sra method ***
3:        Integer l, i, j, k, ioi, io, count
4:        Integer, Parameter :: node = 3 !number of node
5:        Integer, Parameter :: link = 3 !number of link
6:        Integer, Parameter :: nod = 2 !number of OD pairs
7:        Real *8, Parameter :: eee = 1.0D-3 !loop termination condition
8:        Real *8, Parameter :: theta = 2.0D0 !theta parameter
9:        Real *8, Parameter :: alpha = 1.0D0 !BPR parameter
10:       Real *8, Parameter :: beta = 2.0D0 !BPR parameter
11:       Real *8, Parameter :: kiu = 0.5D0 !q parameter
12:       Real *8, Parameter :: ro = 2D0 !sra parameter
13:       Real *8, Parameter :: gamma = 0.01D0 !sra parameter
14:       Integer s(link), e(link), ls(node, node) !start and end node and link number
15:       Integer ori(nod), des(nod) !origin, destination of od pairs
16:       Real *8 yde(nod) !od demand of od pairs
17:       Real t(link), cap(link) !t:free-flow travel time,cap: capacity of link
18:       Real *8 erh(link), p(node, node) !erh: stoch3 parameter, p:shortest route
19:       Integer headnode(0:node), anode(link), alink(link) ! adjacent nodes and links
20:       Real *8 ps(link), pe(link)
21:   !ps,pe: shortest path from each origin to start and end node of the link
22:       Integer omega(nod, link), mark(link)
23:   !omega(nod,link): marking efficient links of each od pair
24:   !mark(link): marking a route at each time reaching the destination in dfs algorithm
25:       Real *8 xo(link), xn(link) !xo: link traffic flow after, xn:link traffic flow before
26:       Real *8 tt(link) !tt: link travel time after update
27:       Real *8 maxrg, g(link) !used in msa method
28:       Real *8 x(link) !x: link travel is assigned
29:       Real *8 num(link), dnm, vn, crou, lamda, rg1, rg2, v
30:   !!******************************************************
31:
32:       Open (1, File='1.network.txt', Action='read')
33:       Open (2, File='2.linkparameter.txt', Action='read')
34:       Open (3, File='3.demand.txt', Action='read')
35:       Open (11, File='11.CalculationTime(SUE).txt', Action='write')
36:       Do i = 1, link
37:         Read (1, *) s(i), e(i), j !s(i):start node, e(i):end node
38:         Read (2, *) t(i), cap(i) !t(i):free flow traveltime, cap(i):Capacity of link i
39:         erh(i) = 1.5
40:       End Do
41:       Do i = 1, nod
42:         Read (3, *) ori(i), des(i), yde(i)
43:       End Do
44:   !make network
45:       ls = 0.0
46:       Do i = 1, link
47:         ls(s(i), e(i)) = i
48:       End Do
49:   ! create adjacent nodes and links
50:       headnode(0) = 0
51:       count = 0
```

```
52:         Do i = 1, node
53:           Do j = 1, link
54:             If (i==s(j)) Then
55:               count = count + 1
56:               anode(count) = e(j)
57:               alink(count) = j
58:             End If
59:           End Do
60:           headnode(i) = count
61:         End Do
62:
63:         Write (6, *) 'theta', theta
64:         Write (6, *) 'alpha', alpha
65:         Write (6, *) 'beta', beta
66:     !step 1: computing static sue
67:     !********* dijkstra method calculation ****************
68:         Call dijikstra(p, t)
69:
70:     !****** dfs algorithm *****************************
71:         xo = 0
72:         omega = 0.0
73:     !***** Creating initial solution based on free-flow travel time***********
74:         tt = 0.0
75:         Do i = 1, link
76:           tt(i) = t(i)
77:         End Do
78:         Do ioi = 1, nod
79:           io = ori(ioi)
80:           Do i = 1, link
81:             ps(i) = p(io, s(i))
82:             pe(i) = p(io, e(i))
83:           End Do
84:
85:     !****** !check STOCH3-efficient routes *****************************
86:           Do i = 1, link
87:             If (ps(i)>pe(i)) Go To 100
88:             If ((1+erh(i))*(pe(i)-ps(i))>=t(i)) Then
89:               omega(ioi, i) = 1
90:             End If
91:     100     End Do
92:     !******running the dfs algorithm from origin node r to destination node s ******
93:           num = 0.0
94:           vn = 0.0
95:           j = ori(ioi)
96:           k = 0
97:           dnm = 0
98:           Call calxij(1)
99:     !************************************************************
100:    !*************************
101:    !Recording after loop termination
102:          Do i = 1, link
103:            omega(ioi, i) = 0
```

```
104:          x(i) = yde(ioi)*num(i)/dnm
105:          If (x(i)/=0) Then
106:            omega(ioi, i) = 1
107:          End If
108:          xo(i) = xo(i) + x(i)
109:        End Do
110:      End Do
111:  !*******starting calculation round************************************
112:      xn = 0.0
113:      g = 0.0
114:      maxrg = 1
115:      l = 0
116:      mark = 0
117:      Do While (maxrg>eee)
118:  !****Update link travel time
119:        l = l + 1
120:        Do i = 1, link
121:          tt(i) = t(i)*(1+alpha*((xo(i)/cap(i))**beta)) !BPR calculation
122:        End Do
123:  !******Direction Finding******
124:        xn = 0.0
125:        Do ioi = 1, nod
126:          num = 0.0
127:          vn = 0.0
128:          j = ori(ioi)
129:          k = 0
130:          dnm = 0
131:          Call calxij(1)
132:  !***************************
133:          Do i = 1, link
134:            xn(i) = xn(i) + yde(ioi)*num(i)/dnm
135:          End Do
136:  !************************************
137:        End Do
138:  !************************************
139:  !Convergence determination calculation
140:        If (l==1) Then
141:          lamda = 1.0D0
142:        Else
143:          rg2 = 0
144:          Do i = 1, link
145:            rg2 = rg2 + (xn(i)-xo(i))**2.0
146:          End Do
147:          rg2 = sqrt(rg2)
148:          If (rg2>=rg1) Then
149:            lamda = lamda + ro
150:          Else
151:            lamda = lamda + gamma
152:          End If
153:        End If
154:        g = 0.0
155:        Do i = 1, link
```

```
156:          g(i) = xn(i) - xo(i)
157:          xn(i) = xo(i) + g(i)/dble(lamda)
158:        End Do
159:        xo = xn
160:        maxrg = maxval(abs(g))
161:        rg1 = rg2
162:        Write (*, *) 'round', l, 'maxrg', maxrg
163:      End Do
164:  !**********Results************************
165:      Write (11, *) 'Round', l, 'Computation gap', maxrg
166:      Call cpu_time(v)
167:      Write (11, *) 'time of calculation (second) ', v
168:      Open (12, File='12.xij.txt')
169:      Do i = 1, link
170:        Write (12, *) xo(i)
171:      End Do
172:      Do i = 1, link
173:        tt(i) = t(i)*(1+alpha*((xo(i)/cap(i))**beta)) !BPR calculation
174:      End Do
175:      Open (13, File='13.tij.txt')
176:      Do i = 1, link
177:        Write (13, *) tt(i)
178:      End Do
179:
180:  !*****SUBROUTINE*************************************************************
181:    Contains
182:      Subroutine dijikstra(p, t)
183:        Real, Intent (In) :: t(link)
184:        Real *8, Intent (Out) :: p(node, node)
185:        Real *8 :: c(node), cmin(node), fin_io(node)
186:
187:        Do j = 1, node
188:          io = j !io is the source node number
189:          c(:) = 10000000.0 !set partial path cost • ‡
190:          c(io) = 0.0 !set the cost of the source node to 0
191:          cmin(:) = 10000000.0
192:          fin_io(:) = 0 !array for storing the nodes for which calculation has been completed
193:          Do While (minval(cmin,1)<50000000)
194:            Do iii = headnode(io-1) + 1, headnode(io)
195:              i = ls(io, anode(iii))
196:              If (c(io)+t(i)<=c(e(i))) Then
197:                c(e(i)) = c(io) + t(i) !update with the next link cost with the node
198:                cmin(e(i)) = c(io) + t(i)
199:              End If
200:            End Do
201:            ncj = minloc(cmin, 1) !identify the location of cmin in the array
202:            cmin(ncj) = 50000000.0 !ineffectiveness of extracted cmin
203:  !**** calculation end judgment (end when all cmin are disabled) ***********
204:            If (minval(cmin,1)==50000000) Then
205:              Go To 100
206:            End If
207:  !******!move to the next node******************
```

```
208:            io = ncj
209:            If (fin_io(ncj)==1) Then !when calculation is completed, move to another node
210:              inot = minloc(fin_io, 1)
211:              io = inot
212:            End If
213:            fin_io(io) = 1
214:          End Do
215:  100      Continue
216:  !******record shortest path results***********************************
217:          p(j, :) = c(:)
218:        End Do
219:
220:      End Subroutine
221:
222:  !   Calculate xij with DFS
223:      Recursive Subroutine calxij(dfs)
224:        Integer dfs, iii
225:
226:        If (j==des(ioi)) Then
227:          crou = 0.0
228:          Do i = 1, k
229:            crou = crou + tt(mark(i))
230:          End Do
231:          vn = (1.0D0+(kiu-1.0D0)*(-theta*crou))**(1.0D0/(kiu-1.0D0))
232:          dnm = dnm + vn
233:          Do i = 1, k
234:            num(mark(i)) = num(mark(i)) + vn
235:          End Do
236:        Else
237:          Do iii = headnode(j-1) + 1, headnode(j)
238:            If (omega(ioi,alink(iii))/=0) Then
239:              j = e(alink(iii))
240:              k = k + 1
241:              mark(k) = alink(iii)
242:              Call calxij(dfs+1)
243:              mark(k) = 0
244:              k = k - 1
245:            End If
246:          End Do
247:        End If
248:      End Subroutine
249:
250:    End Program
251:  ! *******static traffic assignment for q-generalized logit model**********
252:  ! ***Based on stoch3-efficient route, dfs algorithm and the sra method ***
253:      Integer l, i, j, k, ioi, io, count
254:      Integer, Parameter :: node = 3 !number of node
255:      Integer, Parameter :: link = 3 !number of link
256:      Integer, Parameter :: nod = 2 !number of OD pairs
257:      Real *8, Parameter :: eee = 1.0D-3 !loop termination condition
258:      Real *8, Parameter :: theta = 2.0D0 !theta parameter
259:      Real *8, Parameter :: alpha = 1.0D0 !BPR parameter
```

```
260:        Real *8, Parameter :: beta = 2.0D0 !BPR parameter
261:        Real *8, Parameter :: kiu = 0.5D0 !q parameter
262:        Real *8, Parameter :: ro = 2D0 !sra parameter
263:        Real *8, Parameter :: gamma = 0.01D0 !sra parameter
264:        Integer s(link), e(link), ls(node, node) !start and end node and link number
265:        Integer ori(nod), des(nod) !origin, destination of od pairs
266:        Real *8 yde(nod) !od demand of od pairs
267:        Real t(link), cap(link) !t:free-flow travel time,cap: capacity of link
268:        Real *8 erh(link), p(node, node) !erh: stoch3 parameter, p:shortest route
269:        Integer headnode(0:node), anode(link), alink(link) ! adjacent nodes and links
270:        Real *8 ps(link), pe(link)
271:   !ps,pe: shortest path from each origin to start and end node of the link
272:        Integer omega(nod, link), mark(link)
273:   !omega(nod,link): marking efficient links of each od pair
274:   !mark(link): marking a route at each time reaching the destination in dfs algorithm
275:        Real *8 xo(link), xn(link) !xo: link traffic flow after, xn:link traffic flow before
276:        Real *8 tt(link) !tt: link travel time after update
277:        Real *8 maxrg, g(link) !used in msa method
278:        Real *8 x(link) !x: link travel is assigned
279:        Real *8 num(link), dnm, vn, crou, lamda, rg1, rg2, v
280:   !!*****************************************************
281:
282:        Open (1, File='1.network.txt', Action='read')
283:        Open (2, File='2.linkparameter.txt', Action='read')
284:        Open (3, File='3.demand.txt', Action='read')
285:        Open (11, File='11.CalculationTime(SUE).txt', Action='write')
286:        Do i = 1, link
287:          Read (1, *) s(i), e(i), j !s(i):start node, e(i):end node
288:          Read (2, *) t(i), cap(i) !t(i):free flow traveltime, cap(i):Capacity of link i
289:          erh(i) = 1.5
290:        End Do
291:        Do i = 1, nod
292:          Read (3, *) ori(i), des(i), yde(i)
293:        End Do
294:   !make network
295:        ls = 0.0
296:        Do i = 1, link
297:          ls(s(i), e(i)) = i
298:        End Do
299:   ! create adjacent nodes and links
300:        headnode(0) = 0
301:        count = 0
302:        Do i = 1, node
303:          Do j = 1, link
304:            If (i==s(j)) Then
305:              count = count + 1
306:              anode(count) = e(j)
307:              alink(count) = j
308:            End If
309:          End Do
310:          headnode(i) = count
311:        End Do
```

```
312:
313:       Write (6, *) 'theta', theta
314:       Write (6, *) 'alpha', alpha
315:       Write (6, *) 'beta', beta
316:   !step 1: computing static sue
317:   !********* dijkstra method calculation ****************
318:       Call dijikstra(p, t)
319:
320:   !****** dfs algorithm ******************************
321:       xo = 0
322:       omega = 0.0
323:   !***** Creating initial solution based on free-flow travel time***********
324:       tt = 0.0
325:       Do i = 1, link
326:         tt(i) = t(i)
327:       End Do
328:       Do ioi = 1, nod
329:         io = ori(ioi)
330:         Do i = 1, link
331:           ps(i) = p(io, s(i))
332:           pe(i) = p(io, e(i))
333:         End Do
334:
335:   !****** !check STOCH3-efficient routes ******************************
336:         Do i = 1, link
337:           If (ps(i)>pe(i)) Go To 100
338:           If ((1+erh(i))*(pe(i)-ps(i))>=t(i)) Then
339:             omega(ioi, i) = 1
340:           End If
341:   100     End Do
342:   !******running the dfs algorithm from origin node r to destination node s ******
343:         num = 0.0
344:         vn = 0.0
345:         j = ori(ioi)
346:         k = 0
347:         dnm = 0
348:         Call calxij(1)
349:   !************************************************************
350:   !**************************
351:   !Recording after loop termination
352:         Do i = 1, link
353:           omega(ioi, i) = 0
354:           x(i) = yde(ioi)*num(i)/dnm
355:           If (x(i)/=0) Then
356:             omega(ioi, i) = 1
357:           End If
358:           xo(i) = xo(i) + x(i)
359:         End Do
360:       End Do
361:   !*******starting calculation round************************************
362:       xn = 0.0
363:       g = 0.0
```

```
364:        maxrg = 1
365:        l = 0
366:        mark = 0
367:        Do While (maxrg>eee)
368:  !****Update link travel time
369:          l = l + 1
370:          Do i = 1, link
371:            tt(i) = t(i)*(1+alpha*((xo(i)/cap(i))**beta)) !BPR calculation
372:          End Do
373:  !******Direction Finding******
374:          xn = 0.0
375:          Do ioi = 1, nod
376:            num = 0.0
377:            vn = 0.0
378:            j = ori(ioi)
379:            k = 0
380:            dnm = 0
381:            Call calxij(1)
382:  !***************************
383:            Do i = 1, link
384:              xn(i) = xn(i) + yde(ioi)*num(i)/dnm
385:            End Do
```

G.4 Solving the double-looped fixed-point problem of the semi-DTA CNL model

```
1:      !* * SEMI-DYNAMIC TRAFFIC ASSIGNMENT FOR CROSS-NESTED LOGIT MODEL*
2:      ! * * * * * * * * * * *SOLVING DOUBLE-LOOPED FIXED-POINT PROBLEM* * * * *
3:          Integer i, j, ij, k, o, ij1, l, m
4:          Integer, Parameter :: round = 5000000 !the number of computation
5:          Real *8, Parameter :: eee = 1.0D-3 !error in computation
6:
7:          Integer, Parameter :: node = 5 !number of node
8:          Integer, Parameter :: link = 6 !number of link
9:          Integer, Parameter :: nod = 1 !number of OD pairs
10:
11:         Integer, Parameter :: path = 3 !the number of paths
12:         Integer, Parameter :: nmax = 6 !maximum number of links on routes+1 in route-set
13:         Real *8, Parameter :: theta = 0.2D0 !theta parameter
14:         Real *8, Parameter :: alpha = 0.15D0 !BPR parameter
15:         Real *8, Parameter :: beta = 4.0D0 !BPR parameter
16:         Real *8, Parameter :: length = 60.0D0 !time length
17:         Real *8, Parameter :: muy = 0.5D0 !cross-nested logit model parameter
18:
19:     !*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
20:
21:         Integer delta(path, nmax), kk(nod), sum, n(100000), pn(path, nmax), &
22:           en(link)
23:         Integer b(nmax, nmax), des(nod)
24:         Real *8 t0(link), capa(link), x(link), c0(link), c00(link), rgap(link)
25:         Real *8 q(nod)
26:         Real *8 c(path), sc(path), p(path), f(path), fgap(path)
27:         Real *8 gapmax, dij
28:         Real *8 od(node, node)
29:         Real *8 y(link), yij(100000), sij(100000, 1), s(link)
30:         Real *8 z(link), inco(link, path), pm(link), npm(link), tnpm, &
31:           pk(link, path), tnpk(link), pp(path)
32:
33:     !* * *  * * * * * * * * * * * * * * * * * * * * * * * * *
34:
35:         Open (1, File='1.network.txt')
36:         Do i = 1, link
37:           Read (1, *) k, en(i), j
38:         End Do
39:
40:         Open (2, File='2.linkparameter.txt')
41:         Do i = 1, link
42:           Read (2, *) t0(i), capa(i)
43:         End Do
44:
45:         Open (3, File='3.demand.txt')
46:         Do i = 1, nod
47:           Read (3, *) o, des(i), q(i)
48:         End Do
49:
50:         Open (66, File='6.NumberofPaths.txt')
51:         Do i = 1, nod
```

```
52:           Read (66, *) kk(i)
53:         End Do
54:
55:         Open (7, File='7.pass_link.txt')
56:         Do i = 1, path
57:           Read (7, *)(delta(i,j), j=1, nmax)
58:         End Do
59:
60:     !*********count passed link number******
61:         n(:) = 0.0D0
62:         sum = 0.0D0
63:
64:         Open (46, File='n(i).txt')
65:         Do i = 1, path
66:           Do j = 1, nmax
67:             If (delta(i,j)==0) Go To 100
68:             n(i) = n(i) + 1
69:           End Do
70:     100   End Do
71:
72:         Do i = 1, path
73:           sum = sum + n(i)
74:         End Do
75:
76:         Do i = 1, path
77:           Write (46, *) n(i), sum
78:         End Do
79:
80:     !********* passed node number ******
81:         pn(:, :) = 0.0D0
82:         Open (47, File='pn.txt')
83:
84:         Do i = 1, path
85:           Do j = 1, n(i)
86:             If (delta(i,j)==0) Go To 110
87:             pn(i, j) = delta(i, j)
88:           End Do
89:     110   End Do
90:
91:         Do i = 1, path
92:           Do j = 1, n(i)
93:             Write (47, *) pn(i, j)
94:           End Do
95:         End Do
96:
97:     !*********** B  *************
98:         b(:, :) = 0
99:
100:        Do i = 1, nmax
101:          Do j = 1, nmax
102:
103:            If (i<=j) Then
```

```
          b(i, j) = 0
        Else
          b(i, j) = 1
        End If

      End Do
    End Do

    Open (48, File='b(i,j).txt')
    Do i = 1, nmax
      Write (48, *)(b(i,j), j=1, nmax)
    End Do

!* *  * Solving fixed point problem with reference route flows* * * * * * * * * * * * * * * * * * * * * * *

!* *  * * * creating initial solution of reference route flows * * * * * * * * * * * * * * * * * * *
    m = 0
    Do i = 1, nod
      Do j = 1, kk(i)
        m = m + 1
        f(m) = q(i)/kk(i)
      End Do
    End Do

    Do ij = 1, round
!* * *  * * * *Calculating reference link traffic flow * * * * * * * * * *

      x(:) = 0.0D0
      Do i = 1, path
        Do j = 1, nmax
          If (delta(i,j)==0) Go To 120
          in = delta(i, j)
          x(in) = x(in) + f(i)
        End Do
120     End Do

!* * *  Solving fixed point problem with link travel time* * * * * * * * * * * * * * * * * * * * *
!* *  * * * creating initial solution of link travel time * * * * * * * * * * * * * * * * * * *
      c0 = t0
!* *  * * * solving the problem with MSA method * * * * * * * * * * * * * * * * * * *
      Do ij1 = 1, round

!*******************  yij ***********************
        yij(:) = 0.0D0
        k = 0
        Do i = 1, path
          Do j = 1, nmax
            If (delta(i,j)==0) Go To 130
            k = k + 1
            o = delta(i, j)
```

```
154:              yij(k) = f(i)*c0(o)/length
155:            End Do
156:  130       End Do
157:
158:  !********** sij=b*y  matrix calculation  ****************
159:
160:          sij(:, :) = 0.0D0
161:          m = 0.0D0
162:          Do i = 1, path
163:            sij(m+1:m+n(i), 1) = matmul(b(1:n(i),1:n(i)), yij(m+1:m+n(i)))
164:            m = m + n(i)
165:          End Do
166:
167:  !********** Calculating total eliminated flow  *******************************
168:          s(:) = 0.0D0
169:          k = 0.0D0
170:
171:          Do i = 1, path
172:            Do j = 1, n(i)
173:              k = k + 1
174:              in = pn(i, j)
175:              s(in) = s(in) + sij(k, 1)
176:            End Do
177:          End Do
178:  !********** Calculating adjusted link flows  *******************************
179:          z(:) = 0
180:          Do i = 1, link
181:            z(i) = x(i) - s(i)
182:            If (z(i)<0) Then
183:              z(i) = 0
184:            End If
185:          End Do
186:  !**********  Calculating  auxiliary  link  travel  time  and  relation  gap
      *********************************
187:          c00 = 0.0D0
188:          rgap = 0.0D0
189:          Do i = 1, link
190:            c00(i) = t0(i)*(1+alpha*z(i)**beta/capa(i)**beta)
191:            rgap(i) = c0(i) - c00(i)
192:          End Do
193:
194:          dij = dble(ij1)
195:  !********** MSA method *******************************
196:          Do i = 1, link
197:            c0(i) = c0(i) - rgap(i)*(1.0D0/dij)
198:          End Do
199:          gapmax = 0.0D0
200:
201:          Do i = 1, link
202:            If (abs(rgap(i))>gapmax) Then
203:              gapmax = abs(rgap(i))
204:            End If
```

```
205:          End Do
206:  !        Write (*, *) 'Computation number (tij)', ij1, gapmax
207:          If (gapmax<eee) Exit
208:        End Do
209:  !********** Calculating auxiliary route flow ******************************
210:  !*********** path_travel_time * * * * * * * * * * *
211:        pp = 0.0D0
212:        mm = 0
213:        mm1 = 0
214:        mmm = 0
215:        mmmm = 0
216:
217:        Do i = 1, nod
218:          c(:) = 0.0D0
219:          Do j = 1, kk(i)
220:            mm1 = mm1 + 1
221:            Do k = 1, nmax
222:              If (delta(mm1,k)==0) Go To 140
223:              in = delta(mm1, k)
224:              c(j) = c(j) + t0(in)
225:            End Do
226:  140      End Do
227:
228:          inco = 0.0
229:          Do j = 1, kk(i)
230:            mmmm = mmmm + 1
231:            Do k = 1, nmax
232:              If (delta(mmmm,k)==0) Go To 150
233:              in = delta(mmmm, k)
234:              inco(in, j) = t0(in)/c(j)
235:            End Do
236:  150      End Do
237:
238:          c = 0.0D0
239:          Do j = 1, kk(i)
240:            mmm = mmm + 1
241:            Do k = 1, nmax
242:              If (delta(mmm,k)==0) Go To 160
243:              in = delta(mmm, k)
244:              c(j) = c(j) + c0(in)
245:            End Do
246:  160      End Do
247:
248:  !* * * * * * * * * * * * * * * * * * * * * * * * * * * * *
249:
250:          sc(:) = 0.0D0
251:          Do j = 1, kk(i)
252:            sc(j) = exp(-theta*c(j))
253:          End Do
254:
255:          npm = 0.0D0
256:          tnpm = 0.0D0
```

```
257:          Do ii = 1, link
258:            Do j = 1, kk(i)
259:              If (inco(ii,j)/=0) Then
260:                npm(ii) = npm(ii) + (inco(ii,j)*sc(j))**(1.0D0/muy)
261:              End If
262:            End Do
263:            tnpm = tnpm + npm(ii)**muy
264:          End Do
265:  !* * * * * *choice_p* * * * * * * * * * * * * * * *
266:          pm = 0.0D0
267:          Do ii = 1, link
268:            pm(ii) = npm(ii)**muy/tnpm
269:          End Do
270:
271:          tnpk = 0.0D0
272:          Do ii = 1, link
273:            Do j = 1, kk(i)
274:              If (inco(ii,j)/=0) Then
275:                tnpk(ii) = tnpk(ii) + (inco(ii,j)*sc(j))**(1.0D0/muy)
276:              End If
277:            End Do
278:          End Do
279:
280:          pk = 0.0D0
281:          Do ii = 1, link
282:            Do k = 1, kk(i)
283:              If (inco(ii,k)/=0) Then
284:                pk(ii, k) = (inco(ii,k)*sc(k))**(1.0D0/muy)/tnpk(ii)
285:              End If
286:            End Do
287:          End Do
288:          p = 0.0D0
289:          Do ii = 1, kk(i)
290:            mm = mm + 1
291:            Do j = 1, link
292:              If (inco(j,ii)/=0) Then
293:                p(ii) = p(ii) + pm(j)*pk(j, ii)
294:              End If
295:            End Do
296:            pp(mm) = p(ii)
297:          End Do
298:
299:        End Do
300:
301:  !* * *  * * MSA method for solving fixed-point problem with route flows* * * * * * * * * *
      * * * * * * * *
302:
303:        m = 0
304:        Do i = 1, nod
305:          Do j = 1, kk(i)
306:            m = m + 1
307:            fgap(m) = f(m) - q(i)*pp(m)
```

```
308:          End Do
309:        End Do
310:
311:        dij = dble(ij)
312:
313:        Do i = 1, path
314:          f(i) = f(i) - fgap(i)*(1.0D0/dij)
315:        End Do
316:
317:  !* * * * Checking convergence* * * * * * * * * * * * * * * * * * * * * * * * * *
318:
319:        fgapmax = 0.0D0
320:        Do i = 1, path
321:          If (abs(fgap(i))>fgapmax) Then
322:            fgapmax = abs(fgap(i))
323:          End If
324:        End Do
325:        Write (*, *) 'Computation number (fk)', ij, fgapmax
326:        If (fgapmax<eee) Exit
327:
328:      End Do
329:  !* * * * Results* * * * * * * * * * * * * * * * * * * * * * * * *
330:  !* *  * * Saving some varibales * * * * * * * * * *
331:      Open (50, File='round,gapmax,c00,Rgap.txt')
332:      Write (50, *) 'round', ij, 'gapmax', gapmax
333:      Do i = 1, link
334:        Write (50, *) c00(i), rgap(i)
335:      End Do
336:
337:      Open (51, File='fgapmax,fgap.txt')
338:      Write (51, *) 'fgapmax', fgapmax
339:      Do i = 1, path
340:        Write (51, *) fgap(i)
341:      End Do
342:
343:  !* *  * * link residual flow * * * * * * * * * *
344:  !*******************  yij*************************
345:      Open (8, File='8.yij.txt')
346:      Open (68, File='68.FlowPropagation.txt')
347:      yij(:) = 0.0D0
348:      k = 0
349:      od = 0.0D0
350:      i = 0
351:      Do l = 1, nod
352:        Do m = 1, kk(l)
353:          i = i + 1
354:          Do j = 1, n(i)
355:            k = k + 1
356:            o = delta(i, j)
357:            yij(k) = f(i)*c0(o)/length
358:            od(en(o), des(l)) = od(en(o), des(l)) + yij(k)
359:            Write (8, *) o, yij(k)
```

```
360:           End Do
361:         End Do
362:       End Do
363:       Do i = 1, node
364:         Write (68, *)(od(i,j), j=1, node)
365:       End Do
366:  !**sum**
367:
368:       y(:) = 0.0D0
369:
370:       Do i = 1, path
371:         Do j = 1, nmax
372:           If (delta(i,j)==0) Go To 170
373:           in = delta(i, j)
374:           y(in) = y(in) + f(i)*c0(in)/length
375:         End Do
376:  170   End Do
377:
378:       Open (11, File='11.y(i).txt')
379:
380:       Do i = 1, link
381:         Write (11, *) i, y(i)
382:       End Do
383:
384:  !*********** sij=b*y  matrix calculation  *****************
385:       Open (12, File='12.sij.txt')
386:
387:       sij(:, :) = 0.0D0
388:       m = 0.0D0
389:
390:       Do i = 1, path
391:         sij(m+1:m+n(i), 1) = matmul(b(1:n(i),1:n(i)), yij(m+1:m+n(i)))
392:         m = m + n(i)
393:       End Do
394:       Do i = 1, sum
395:         Write (12, *) i, sij(i, 1)
396:       End Do
397:
398:  !*********** s=sum_sij  *********************************
399:       Open (13, File='13.s(i).txt')
400:       s(:) = 0.0D0
401:       k = 0.0D0
402:
403:       Do i = 1, path
404:         Do j = 1, n(i)
405:           k = k + 1
406:           in = pn(i, j)
407:           s(in) = s(in) + sij(k, 1)
408:         End Do
409:       End Do
410:
411:       Do i = 1, link
```

```
412:          Write (13, *) i, s(i)
413:        End Do
414:
415:   !************ z=x(delta_f)-s   ******************************
416:        Open (14, File='14.z(i).txt')
417:        z(:) = 0
418:
419:        Do i = 1, link
420:          z(i) = x(i) - s(i)
421:        End Do
422:
423:        Do i = 1, link
424:          Write (14, *) i, z(i)
425:        End Do
426:
427:   !***********************************************
428:        Write (*, *) 'alpha :', alpha
429:        Write (*, *) 'beta :', beta
430:        Write (*, *) 'theta :', theta
431:        Write (*, *) 'Maximum of the gap', fgapmax
432:
433:   !* ** * * * * * * * * * * * * * * * * *
434:        Open (15, File='15.x(i).txt')
435:        Do i = 1, link
436:          Write (15, *) i, x(i)
437:        End Do
438:
439:        Open (16, File='16.linktraveltime.txt')
440:        Do i = 1, link
441:          Write (16, *) i, c0(i)
442:        End Do
443:
444:        Open (17, File='17.sc.txt')
445:        Do i = 1, path
446:          Write (17, *) i, sc(i)
447:        End Do
448:
449:        Open (18, File='18.inco.txt')
450:        Do i = 1, link
451:          Write (18, *)(inco(i,j), j=1, path)
452:        End Do
453:
454:        Open (19, File='19.p(i).txt')
455:        Do i = 1, path
456:          Write (19, *) p(i), pp(i)
457:        End Do
458:
459:        Open (20, File='20.c(i),f(i).txt')
460:        Do i = 1, path
461:          Write (20, *) i, c(i), f(i)
462:        End Do
463:        Open (21, File='21.pm,npm,tnpk.txt')
```

```
464:        Do i = 1, link
465:          Write (21, *) pm(i), npm(i), tnpk(i)
466:        End Do
467:        Open (22, File='22.pk.txt')
468:        Do i = 1, link
469:          Write (22, *)(pk(i,j), j=1, path)
470:        End Do
471:        Open (23, File='23.Calculation time.txt')
472:        Call cpu_time(v)
473:        Write (23, *) 'time', v
474:
475:        Stop
476:
477:      End Program
```

G.5 Solving the double-looped fixed-point problem of the semi-DTA q-generalized model

```
1:      !* SEMI-DYNAMIC TRAFFIC ASSIGNMENT FOR q-GENERALIZED LOGIT MODEL
2:      ! * * * * * * * * *SOLVING DOUBLE-LOOPED FIXED-POINT PROBLEM* * * * *
3:          Integer i, j, ij, k, o, ij1, l, m
4:          Integer, Parameter :: round = 5000000 !the number of computation
5:          Real *8, Parameter :: eee = 1.0D-3 !error in computation
6:
7:          Integer, Parameter :: node = 5 !number of node
8:          Integer, Parameter :: link = 6 !number of link
9:          Integer, Parameter :: nod = 1 !number of OD pairs
10:
11:         Integer, Parameter :: path = 3 !the number of paths
12:         Integer, Parameter :: nmax = 4 !maximum number of links on routes+1 in route-set
13:         Real *8, Parameter :: theta = 0.2D0 !theta parameter
14:         Real *8, Parameter :: alpha = 0.15D0 !BPR parameter
15:         Real *8, Parameter :: beta = 4.0D0 !BPR parameter
16:         Real *8, Parameter :: length = 60.0D0 !time length
17:         Real *8, Parameter :: kiu = 0.5D0 !q-generalized logit model parameter
18:     !*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
19:
20:         Integer delta(path, nmax), kk(nod), sum, n(100000), pn(path, nmax), &
21:           en(link)
22:         Integer b(nmax, nmax), des(nod)
23:         Real *8 t0(link), capa(link), x(link), c0(link), c00(link), rgap(link)
24:         Real *8 q(nod), sgm(nod)
25:         Real *8 c(path), sc(path), p(path), f(path), fgap(path)
26:         Real *8 gapmax, dij
27:         Real *8 od(node, node)
28:         Real *8 y(link), yij(100000), sij(100000, 1), s(link)
29:         Real *8 z(link)
30:
31:     !* * *  * * * * * * * * * * * * * * * * * * * * * * * * *
32:
33:         Open (1, File='1.network.txt')
34:         Do i = 1, link
35:           Read (1, *) k, en(i), j
36:         End Do
37:
38:         Open (2, File='2.linkparameter.txt')
39:         Do i = 1, link
40:           Read (2, *) t0(i), capa(i)
41:         End Do
42:
43:         Open (3, File='3.demand.txt')
44:         Do i = 1, nod
45:           Read (3, *) o, des(i), q(i)
46:         End Do
47:
48:         Open (66, File='6.NumberofPaths.txt')
49:         Do i = 1, nod
```

```
50:           Read (66, *) kk(i)
51:         End Do
52:
53:         Open (7, File='7.pass_link.txt')
54:         Do i = 1, path
55:           Read (7, *)(delta(i,j), j=1, nmax)
56:         End Do
57:
58:     !*********count passed link number******
59:         n(:) = 0.0D0
60:         sum = 0.0D0
61:
62:         Open (46, File='n(i).txt')
63:         Do i = 1, path
64:           Do j = 1, nmax
65:             If (delta(i,j)==0) Go To 100
66:             n(i) = n(i) + 1
67:           End Do
68:     100   End Do
69:
70:         Do i = 1, path
71:           sum = sum + n(i)
72:         End Do
73:
74:         Do i = 1, path
75:           Write (46, *) n(i), sum
76:         End Do
77:
78:     !********* passed node number       ******
79:         pn(:, :) = 0.0D0
80:         Open (47, File='pn.txt')
81:
82:         Do i = 1, path
83:           Do j = 1, n(i)
84:             If (delta(i,j)==0) Go To 110
85:             pn(i, j) = delta(i, j)
86:           End Do
87:     110   End Do
88:
89:         Do i = 1, path
90:           Do j = 1, n(i)
91:             Write (47, *) pn(i, j)
92:           End Do
93:         End Do
94:
95:     !********** B  ************
96:         b(:, :) = 0
97:         Do i = 1, nmax
98:           Do j = 1, nmax
99:             If (i<=j) Then
100:              b(i, j) = 0
101:            Else
```

```
102:            b(i, j) = 1
103:          End If
104:
105:        End Do
106:      End Do
107:
108:      Open (48, File='b(i,j).txt')
109:      Do i = 1, nmax
110:        Write (48, *)(b(i,j), j=1, nmax)
111:      End Do
112:
113:  !* *  * Solving fixed point problem with reference route flows* * * * * * * * * * * * * * * * * * * * * * * * * *
114:  !* *  * * * creating initial solution of reference route flows * * * * * * * * * * * * * * * * * * * * * *
115:      m = 0
116:      Do i = 1, nod
117:        Do j = 1, kk(i)
118:          m = m + 1
119:          f(m) = q(i)/kk(i)
120:        End Do
121:      End Do
122:
123:      Do ij = 1, round
124:  !* * *  * * * *Calculating reference link traffic flow * * * * * * * * * *
125:
126:        x(:) = 0.0D0
127:        Do i = 1, path
128:          Do j = 1, nmax
129:            If (delta(i,j)==0) Go To 120
130:            in = delta(i, j)
131:            x(in) = x(in) + f(i)
132:          End Do
133:  120     End Do
134:
135:  !* * *  Solving fixed point problem with link travel time* * * * * * * * * * * * * * * * * * * * * * * * * * *
136:  !* *  * * * creating initial solution of link travel time * * * * * * * * * * * * * * * * * * * *
137:        c0 = t0
138:  !* *  * * * solving the problem with MSA method * * * * * * * * * * * * * * * * * * * *
139:        Do ij1 = 1, round
140:
141:  !********************  yij************************
142:          yij(:) = 0.0D0
143:          k = 0
144:          Do i = 1, path
145:            Do j = 1, nmax
146:              If (delta(i,j)==0) Go To 130
147:              k = k + 1
148:              o = delta(i, j)
149:              yij(k) = f(i)*c0(o)/length
150:            End Do
151:  130      End Do
```

```
152:
153:  !********** sij=b*y  matrix calculation  ***************
154:          sij(:, :) = 0.0D0
155:          m = 0.0D0
156:
157:          Do i = 1, path
158:            sij(m+1:m+n(i), 1) = matmul(b(1:n(i),1:n(i)), yij(m+1:m+n(i)))
159:            m = m + n(i)
160:          End Do
161:
162:  !********** Calculating total eliminated flow  ******************************
163:          s(:) = 0.0D0
164:          k = 0.0D0
165:
166:          Do i = 1, path
167:            Do j = 1, n(i)
168:              k = k + 1
169:              in = pn(i, j)
170:              s(in) = s(in) + sij(k, 1)
171:            End Do
172:          End Do
173:
174:  !********** Calculating adjusted link flows  ******************************
175:          z(:) = 0
176:          Do i = 1, link
177:            z(i) = x(i) - s(i)
178:            If (z(i)<0) Then
179:              z(i) = 0
180:            End If
181:          End Do
182:
183:  !********** Calculating auxiliary link travel time and relation gap
      ******************************
184:          c00 = 0.0D0
185:          rgap = 0.0D0
186:          Do i = 1, link
187:            c00(i) = t0(i)*(1+alpha*z(i)**beta/capa(i)**beta)
188:            rgap(i) = c0(i) - c00(i)
189:          End Do
190:
191:          dij = dble(ij1)
192:  !********** MSA method ******************************
193:          Do i = 1, link
194:            c0(i) = c0(i) - rgap(i)*(1.0D0/dij)
195:          End Do
196:          gapmax = 0.0D0
197:
198:          Do i = 1, link
199:            If (abs(rgap(i))>gapmax) Then
200:              gapmax = abs(rgap(i))
201:            End If
202:          End Do
```

```
203:  !         write(*,*)'Computation number (tij)',ij1,gapmax
204:           If (gapmax<eee) Exit
205:         End Do
206:  !*********** Calculating auxiliary route flow ******************************
207:  !************ path_travel_time * * * * * * * * * * *
208:         c(:) = 0.0D0
209:
210:         Do i = 1, path
211:           Do k = 1, nmax
212:             If (delta(i,k)==0) Go To 140
213:             in = delta(i, k)
214:             c(i) = c(i) + c0(in)
215:           End Do
216:  140     End Do
217:
218:  !* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
219:         sc(:) = 0.0D0
220:         sgm = 0.0D0
221:         m = 0
222:         Do i = 1, nod
223:           Do j = 1, kk(i)
224:             m = m + 1
225:             sc(m) = (1.0D0+(kiu-1.0D0)*(-theta*c(m)))**(1.0D0/(kiu-1.0D0))
226:             sgm(i) = sgm(i) + sc(m)
227:           End Do
228:         End Do
229:
230:  !* * *  * * MSA method for solving fixed-point problem with route flows* * * * * * * * * *
      * * * * * * * *
231:         m = 0
232:         Do i = 1, nod
233:           Do j = 1, kk(i)
234:             m = m + 1
235:             fgap(m) = f(m) - q(i)*sc(m)/sgm(i)
236:           End Do
237:         End Do
238:
239:         dij = dble(ij)
240:
241:         Do i = 1, path
242:           f(i) = f(i) - fgap(i)*(1.0D0/dij)
243:         End Do
244:
245:  !* * * * * Checking convergence* * * * * * * * * * * * * * * * * * * * * * * * * * *
246:         fgapmax = 0.0D0
247:         Do i = 1, path
248:           If (abs(fgap(i))>fgapmax) Then
249:             fgapmax = abs(fgap(i))
250:           End If
251:         End Do
252:         Write (*, *) 'Computation number (fk)', ij, fgapmax
253:         If (fgapmax<eee) Exit
```

```
254:
255:       End Do
256:   !* * * * * Results* * * * * * * * * * * * * * * * * * * * * * * * * * *
257:   !* *  * * Saving some varibales * * * * * * * * * *
258:       Open (50, File='round,gapmax,c00,Rgap.txt')
259:       Write (50, *) 'round', ij, 'gapmax', gapmax
260:       Do i = 1, link
261:         Write (50, *) c00(i), rgap(i)
262:       End Do
263:
264:       Open (51, File='fgapmax,fgap.txt')
265:       Write (51, *) 'fgapmax', fgapmax
266:       Do i = 1, path
267:         Write (51, *) fgap(i)
268:       End Do
269:
270:   !* *  * * link residual flow * * * * * * * * *
271:   !*******************  yij************************
272:       Open (8, File='8.yij.txt')
273:       Open (68, File='68.FlowPropagation.txt')
274:       yij(:) = 0.0D0
275:       k = 0
276:       od = 0.0D0
277:       i = 0
278:       Do l = 1, nod
279:         Do m = 1, kk(l)
280:           i = i + 1
281:           Do j = 1, n(i)
282:             k = k + 1
283:             o = delta(i, j)
284:             yij(k) = f(i)*c0(o)/length
285:             od(en(o), des(l)) = od(en(o), des(l)) + yij(k)
286:             Write (8, *) o, yij(k)
287:           End Do
288:         End Do
289:       End Do
290:       Do i = 1, node
291:         Write (68, *)(od(i,j), j=1, node)
292:       End Do
293:   !**sum**
294:
295:       y(:) = 0.0D0
296:
297:       Do i = 1, path
298:         Do j = 1, nmax
299:           If (delta(i,j)==0) Go To 150
300:           in = delta(i, j)
301:           y(in) = y(in) + f(i)*c0(in)/length
302:         End Do
303:   150   End Do
304:
305:       Open (11, File='11.y(i).txt')
```

```
306:
307:        Do i = 1, link
308:          Write (11, *) y(i)
309:        End Do
310:
311:    !*********** sij=b*y  matrix calculation  ***************
312:        Open (12, File='12.sij.txt')
313:        sij(:, :) = 0.0D0
314:        m = 0.0D0
315:
316:        Do i = 1, path
317:          sij(m+1:m+n(i), 1) = matmul(b(1:n(i),1:n(i)), yij(m+1:m+n(i)))
318:          m = m + n(i)
319:        End Do
320:        Do i = 1, sum
321:          Write (12, *) i, sij(i, 1)
322:        End Do
323:
324:    !*********** s=sum_sij  *********************************
325:        Open (13, File='13.s(i).txt')
326:        s(:) = 0.0D0
327:        k = 0.0D0
328:
329:        Do i = 1, path
330:          Do j = 1, n(i)
331:            k = k + 1
332:            in = pn(i, j)
333:            s(in) = s(in) + sij(k, 1)
334:          End Do
335:        End Do
336:
337:        Do i = 1, link
338:          Write (13, *) s(i)
339:        End Do
340:
341:    !************ z=x(delta_f)-s   ******************************
342:        Open (14, File='14.z(i).txt')
343:        z(:) = 0
344:
345:        Do i = 1, link
346:          z(i) = x(i) - s(i)
347:        End Do
348:
349:        Do i = 1, link
350:          Write (14, *) z(i)
351:        End Do
352:
353:    !***********************************************
354:
355:        Write (*, *) 'alpha :', alpha
356:        Write (*, *) 'beta :', beta
357:        Write (*, *) 'theta :', theta
```

```
358:        Write (*, *) 'Maximum of the gap', fgapmax
359:
360:    !* ** * * * * * * * * * * * * * * * * * *
361:
362:        Open (15, File='15.x(i).txt')
363:        Do i = 1, link
364:          Write (15, *) i, x(i)
365:        End Do
366:
367:        Open (16, File='16.linktraveltime.txt')
368:        Do i = 1, link
369:          Write (16, *) i, c0(i)
370:        End Do
371:
372:        Open (17, File='17.sc.txt')
373:        Do i = 1, path
374:          Write (17, *) i, sc(i)
375:        End Do
376:
377:        Open (20, File='20.c(i),f(i).txt')
378:        Do i = 1, path
379:          Write (20, *) i, c(i), f(i)
380:        End Do
381:
382:        Open (23, File='23.Calculation time.txt')
383:        Call cpu_time(v)
384:        Write (23, *) 'time', v
385:
386:        Stop
387:
388:      End Program
```

G.6 Solving the semi-DTA MNL model with the sensitivity analysis

```
1:    ! *******semi-dynamic traffic assignment for multinomial logit model**********
2:    ! **********second algorithm based on stoch3-efficient route ************
3:    ! **************solving one fixed-point problem method ************************
4:        Integer i, j, k, ioi, io, count
5:        Integer, Parameter :: node = 5 !number of nodes
6:        Integer, Parameter :: link = 6 !number of links
7:        Integer, Parameter :: nod = 1 !number of od pairs
8:        Real *8, Parameter :: eee = 1.0D-3 !loop termination condition
9:        Real *8, Parameter :: theta = 0.2D0 !logit parameter
10:       Real *8, Parameter :: alpha = 0.15D0 !bpr parameter
11:       Real *8, Parameter :: beta = 4.0D0 !bpr parameter
12:       Real *8, Parameter :: lengtht = 60D0 !length of period (minutes)
13:       Real *8, Parameter :: ro = 2D0 !sra parameter
14:       Real *8, Parameter :: gamma = 0.01D0 !sra parameter
15:       Integer s(link), e(link), ls(node, node) !start and end node and link number
16:       Integer ori(nod), des(nod) !origin, destination of od pairs
17:       Real *8 yde(nod) !od demand of od pairs
18:       Real t(link), cap(link) !t:free-flow travel time,cap: capacity of link
19:       Real *8 erh(link), p(node, node) !erh: stoch3 parameter, p:shortest route
20:       Integer headnode(0:node), anode(link), alink(link) ! adjacent nodes and links
21:       Real *8 ps(link), pe(link)
22:   !ps,pe: shortest path from each origin to start and end node of the link
23:       Real *8 a(link), num(link), vn(node), dnm
24:   !a:link likelihood
25:       Integer omega(nod, link), mark(link) !Used to check STOCH3-efficient path
26:       Real *8 xo(link), xn(link) !xo: link traffic flow after, xn:link traffic flow before
27:       Real *8 tt(link) !tt: link travel time after update,
28:       Real *8 x(link)
29:   !x: link travel is assigned
30:       Real *8 tx(link) !tx:derivative of t respect to x
31:   !Variable used to calculate derivative of g respect to t and Detivative of x respect to s
32:       Real *8 gradgt(link, link) !used to calculate derivative of u respect to t
33:       Real *8 part1(link, link), part2(link, link)
34:       Real *8 www, pivot, part3(link, 2*link)
35:       Real *8 grad(link, link) !Grad: Detivative of x respect to s
36:       Real *8 sij(link), yij(link), yijrs(link), b(node, node) !sij:total eliminated flow
37:       Real *8 xafter(link) !link travel flow after sensitivity analysis
38:       Real *8 xsueafter(link), dijgh(link, link)
39:       Real *8 xijrs(link), xijgh1(link, link), xijgh(link, link) ! Used to calculate xijgh
40:       Real *8 lamda, rg1, rg2, sij1(link), rg(link), maxrg
41:   !*****************************************************
42:
43:       Open (1, File='1.network.txt', Action='read')
44:       Open (2, File='2.linkparameter.txt', Action='read')
45:       Open (3, File='3.demand.txt', Action='read')
46:       Open (11, File='11.CalculationTime.txt', Action='write')
47:
48:       Do i = 1, link
49:         Read (1, *) s(i), e(i), j !s(i):start node, e(i):end node
50:         Read (2, *) t(i), cap(i) !t(i):free flow traveltime, cap(i):Capacity of link i
51:         erh(i) = 1.5
```

```
52:         End Do
53:         Do i = 1, nod
54:           Read (3, *) ori(i), des(i), yde(i)
55:         End Do
56:     !make network
57:         ls = 0.0
58:         Do i = 1, link
59:           ls(s(i), e(i)) = i
60:         End Do
61:     ! create adjacent nodes and links
62:         headnode(0) = 0
63:         count = 0
64:         Do i = 1, node
65:           Do j = 1, link
66:             If (i==s(j)) Then
67:               count = count + 1
68:               anode(count) = e(j)
69:               alink(count) = j
70:             End If
71:           End Do
72:           headnode(i) = count
73:         End Do
74:
75:         Write (6, *) 'theta', theta
76:         Write (6, *) 'alpha', alpha
77:         Write (6, *) 'beta', beta
78:
79:     !********* Dijkstra method calculation *****************
80:         Call dijikstra(p, t)
81:
82:     !****** dfs algorithm ******************************
83:         xo = 0
84:         omega = 0.0
85:     !***** Creating initial solution based on free-flow travel time***********
86:         a = 0.0
87:         tt = 0.0
88:         Do i = 1, link
89:           tt(i) = t(i)
90:           a(i) = exp(-theta*tt(i))
91:         End Do
92:         Do ioi = 1, nod
93:           io = ori(ioi)
94:           Do i = 1, link
95:             ps(i) = p(io, s(i))
96:             pe(i) = p(io, e(i))
97:           End Do
98:
99:     !****** !check STOCH3-efficient routes ******************************
100:        Do i = 1, link
101:          If (ps(i)>pe(i)) Go To 100
102:          If ((1+erh(i))*(pe(i)-ps(i))>=t(i)) Then
103:            omega(ioi, i) = 1
```

```
104:          End If
105:  100     End Do
106:  !******running the dfs algorithm from origin node r to destination node s ******
107:        num = 0.0
108:        vn = 0.0D0
109:        vn(ori(ioi)) = 1
110:        j = ori(ioi)
111:        k = 0
112:        dnm = 0
113:        Call calxij(1)
114:  !************************************************************
115:  ! recording after loop termination
116:        Do i = 1, link
117:          omega(ioi, i) = 0
118:          x(i) = yde(ioi)*num(i)/dnm
119:          If (x(i)/=0) Then
120:            omega(ioi, i) = 1
121:          End If
122:          xo(i) = xo(i) + x(i)
123:        End Do
124:      End Do
125:  !*******starting calculation round*************************************
126:      xn = 0.0
127:
128:      maxrg = 1
129:      l = 0
130:      Do While (maxrg>eee)
131:  !*****Link travel time and link impedances update**********************
132:        l = l + 1
133:        a = 0.0D0
134:        Do i = 1, link
135:          tt(i) = t(i)*(1+alpha*((xo(i)/cap(i))**beta)) !BPR calculation
136:          a(i) = exp(-theta*(tt(i)))
137:        End Do
138:  !***** STOCH 3 algorithm ******************************
139:        xn = 0.0
140:        Do ioi = 1, nod
141:          num = 0.0
142:          vn = 0.0D0
143:          vn(ori(ioi)) = 1
144:          j = ori(ioi)
145:          k = 0
146:          dnm = 0
147:          Call calxij(1)
148:  !**************************
149:          Do i = 1, link
150:            xn(i) = xn(i) + yde(ioi)*num(i)/dnm
151:          End Do
152:  !************************************
153:        End Do
154:  !************************************
155:  !c Convergence determination calculation
```

```
156:
157:          If (l==1) Then
158:            lamda = 1.0D0
159:          Else
160:            rg2 = 0
161:            Do i = 1, link
162:              rg2 = rg2 + (xn(i)-xo(i))**2.0
163:            End Do
164:            rg2 = sqrt(rg2)
165:            If (rg2>=rg1) Then
166:              lamda = lamda + ro
167:            Else
168:              lamda = lamda + gamma
169:            End If
170:          End If
171:
172:          Do i = 1, link
173:            rg(i) = xn(i) - xo(i)
174:            xn(i) = xo(i) + rg(i)/lamda
175:          End Do
176:
177:    !**********Step 8: Link flow update*******************************
178:          xo = xn
179:          rg1 = rg2
180:          maxrg = maxval(rg)
181:          Write (*, *) 'Round', l, 'maxRG', maxrg
182:        End Do
183:
184:    !**********RECORDING THE RESULTS*********************************
185:        Write (11, *) 'Round of Static SUE', l, 'Computation gap', maxrg
186:        Call cpu_time(v)
187:        Write (11, *) 'Calculation time 1 (second) ', v
188:        Open (12, File='12.xij.txt')
189:        Do i = 1, link
190:          Write (12, *) xo(i)
191:        End Do
192:        Do i = 1, link
193:          tt(i) = t(i)*(1+alpha*((xo(i)/cap(i))**beta)) !BPR calculation
194:        End Do
195:        Open (13, File='13.tij.txt')
196:        Do i = 1, link
197:          Write (13, *) tt(i)
198:        End Do
199:        Open (16, File='16.omega.txt')
200:        Do i = 1, nod
201:          Write (16, *)(omega(i,j), j=1, link)
202:        End Do
203:        Open (17, File='17.x,xn,a,num.txt')
204:        Do i = 1, link
205:          Write (17, *) x(i), xn(i), a(i), num(i)
206:        End Do
207:        Open (18, File='18.vn.txt')
```

```
208:        Do i = 1, node
209:          Write (18, *) vn(i)
210:        End Do
211:
212:   !****** STEP2: Calculate Gradient*****************************
213:        mark = 0
214:        gradgt = 0.0
215:        sij = 0.0
216:        xijrs = 0.0
217:        l = 0.0
218:
219:   !***** Calculation of link likeli-hood *********************
220:        a = 0.0
221:        Do i = 1, link
222:          a(i) = exp(-theta*(tt(i)))
223:        End Do
224:
225:   !****** calculating derivitives of g with respect to t*****************
226:        Do ioi = 1, nod
227:          io = ori(ioi)
228:          num = 0.0D0
229:          vn = 0.0D0
230:          vn(ori(ioi)) = 1
231:          j = ori(ioi)
232:          k = 0
233:          dnm = 0.0D0
234:          dijgh = 0.0
235:          Call calxij2(1)
236:          xijgh1 = 0.0D0
237:          xijrs = 0.0
238:   !      write (*,*) 'finish_xijrs'
239:          Do i = 1, link
240:            xijrs(i) = yde(ioi)*num(i)/dnm
241:            Do j = 1, link
242:              xijgh1(i, j) = yde(ioi)*dijgh(i, j)/dnm
243:              If (xijgh1(i,j)/=0) Then
244:                sij(j) = sij(j) + xijgh1(i, j)/lengtht*tt(i)
245:              End If
246:            End Do
247:          End Do
248:          xijgh = 0.0
249:          Do i = 1, link
250:            Do j = 1, link
251:              xijgh(i, j) = xijgh1(i, j) + xijgh1(j, i)
252:            End Do
253:            xijgh(i, i) = xijrs(i)
254:          End Do
255:
256:          Do i = 1, link
257:            Do j = 1, link
258:              gradgt(i, j) = gradgt(i, j) + theta*(xijrs(i)*xijrs(j)/yde(ioi)- &
259:                xijgh(i,j))
```

```
260:           End Do
261:         End Do
262:       End Do
263:   !*************************************************************
264:   !*******Record results*****
265:       Call cpu_time(v)
266:       Write (11, *) 'Calculation time 1 (second)', v
267:
268:       Open (22, File='22.gradgt.txt')
269:       Do i = 1, link
270:         Write (22, *) gradgt(i, :)
271:       End Do
272:       Write (*, *) 'finish_gradgt'
273:
274:   !***** Calculating derivitives of implicit BPR function*****************
275:       Do i = 1, link
276:         tx(i) = t(i)*beta*alpha*(xo(i)**(beta-1))/(cap(i)**beta)
277:       End Do
278:       Open (25, File='25.tx.txt')
279:       Do i = 1, link
280:         Write (25, *) tx(i)
281:       End Do
282:   !*******Calculation Gradient**********
283:       part1 = 0.0
284:       part2 = 0.0
285:       part3 = 0.0
286:       Do i = 1, link
287:         Do j = 1, link
288:           part1(i, j) = gradgt(i, j)*tx(j)
289:         End Do
290:       End Do
291:
292:       Do i = 1, link
293:         Do j = 1, link
294:           If (i==j) Then
295:             part2(i, j) = 1 - part1(i, j)
296:           Else
297:             part2(i, j) = -part1(i, j)
298:           End If
299:         End Do
300:       End Do
301:       Do i = 1, link
302:         Do j = 1, link
303:           part3(i, j) = part2(i, j)
304:         End Do
305:       End Do
306:
307:       Do i = 1, link
308:         Do j = 1, link
309:           part3(i, j+link) = 0
310:           part3(i, i+link) = 1
311:         End Do
```

```
312:        End Do
313:
314:        Do k = 1, link
315:          pivot = part3(k, k)
316:          If (pivot==0) Then
317:            i = k + 1
318:            Do While (pivot==0 .And. i<link)
319:              pivot = part3(i, k)
320:              i = i + 1
321:            End Do
322:            If (pivot==0) Then
323:              Stop
324:            Else
325:              Do j = 1, 2*link
326:                www = part3(k, j)
327:                i = i - 1
328:                part3(k, j) = part3(i, j)
329:                part3(i, j) = www
330:              End Do
331:            End If
332:          End If
333:          Do j = 1, link*2
334:            part3(k, j) = part3(k, j)/pivot
335:          End Do
336:
337:          Do i = 1, link
338:            If (i/=k) Then
339:              www = part3(i, k)
340:              Do j = 1, link*2
341:                part3(i, j) = part3(i, j) - www*part3(k, j)
342:              End Do
343:            End If
344:          End Do
345:        End Do
346:
347:        part2 = 0.0
348:        Do i = 1, link
349:          Do j = 1, link
350:            part2(i, j) = part3(i, j+link)
351:          End Do
352:        End Do
353:        grad = 0
354:        grad = -matmul(part2, part1)
355:
356:        Open (26, File='26.gradxs.txt')
357:        Do i = 1, link
358:          Write (26, *) grad(i, :)
359:        End Do
360:
361:        Write (*, *) 'finish_Gradient'
362:  !******STEP3: SOLVING FIXED-PONT PROBLEM OF ELIMINATED
      FLOW******************
```

```
363:      l = 0
364:      maxrg = 1.0D0
365:      xijgh1 = 0.0
366:      Do While (maxrg>0.0001)
367:        l = l + 1
368:        xsueafter = 0.0D0
369:        xsueafter = matmul(grad, sij)
370:        Do i = 1, link
371:          xsueafter(i) = xo(i) + xsueafter(i)
372:          tt(i) = t(i)*(1+alpha*(((xsueafter(i)-sij(i))/cap(i))**beta))
373:          a(i) = exp(-theta*(tt(i)))
374:        End Do
375:        sij1 = 0.0D0
376:        Do ioi = 1, nod
377:          io = ori(ioi)
378:          num = 0.0D0
379:          vn = 0.0D0
380:          vn(ori(ioi)) = 1
381:          j = ori(ioi)
382:          k = 0
383:          dnm = 0.0D0
384:          dijgh = 0.0D0
385:          Call calxij2(1)
386:          xijgh1 = 0.0D0
387:          Do i = 1, link
388:            Do j = 1, link
389:              xijgh1(i, j) = yde(ioi)*dijgh(i, j)/dnm
390:              If (xijgh1(i,j)/=0) Then
391:                sij1(j) = sij1(j) + xijgh1(i, j)/lengtht*tt(i)
392:              End If
393:            End Do
394:          End Do
395:        End Do
396:        If (l==1) Then
397:          lamda = 1.0D0
398:        Else
399:          rg2 = 0
400:          Do i = 1, link
401:            rg2 = rg2 + (sij(i)-sij1(i))**2.0
402:          End Do
403:          rg2 = sqrt(rg2)
404:          If (rg2>=rg1) Then
405:            lamda = lamda + ro
406:          Else
407:            lamda = lamda + gamma
408:          End If
409:        End If
410:        maxrg = 0.0D0
411:        Do i = 1, link
412:          rg(i) = sij1(i) - sij(i)
413:          sij(i) = sij(i) + rg(i)/lamda
414:          If (abs(rg(i))>maxrg) Then
```

```
415:              maxrg = abs(rg(i))
416:            End If
417:          End Do
418:          rg1 = rg2
419:          Write (6, *) l, maxrg
420:        End Do
421:    !Calculate link travel flow, residual link flow and link travel time at the semi-DTA
422:
423:        xsueafter = matmul(grad, sij)
424:        Do i = 1, link
425:          xsueafter(i) = xo(i) + xsueafter(i)
426:          xafter(i) = max((xsueafter(i)-sij(i)), 0D0)
427:          tt(i) = t(i)*(1+alpha*(((xsueafter(i)-sij(i))/cap(i))**beta))
428:          a(i) = exp(-theta*(tt(i)))
429:        End Do
430:
431:        yij = 0.0
432:        b = 0.0
433:    !********DFS algorithm with each OD pair*****************************
434:        Do ioi = 1, nod
435:          io = ori(ioi)
436:          num = 0.0D0
437:          vn = 0.0D0
438:          vn(ori(ioi)) = 1
439:          j = ori(ioi)
440:          k = 0
441:          dnm = 0.0D0
442:          Call calxij(1)
443:          Do i = 1, link
444:            xijrs(i) = yde(ioi)*num(i)/dnm
445:          End Do
446:          yijrs = 0.0
447:          Do i = 1, link
448:            yijrs(i) = xijrs(i)/lengtht*tt(i)
449:            yij(i) = yij(i) + yijrs(i)
450:          End Do
451:
452:          Do i = 1, link
453:            If (yijrs(i)/=0) Then
454:              b(e(i), des(ioi)) = b(e(i), des(ioi)) + yijrs(i)
455:            End If
456:          End Do
457:        End Do
458:
459:        Open (28, File='28.sij.txt')
460:        Do i = 1, link
461:          Write (28, *) sij(i)
462:        End Do
463:        Open (29, File='29.yij.txt')
464:        Do i = 1, link
465:          Write (29, *) yij(i)
466:        End Do
```

```
467:       Open (37, File='37.xSUEafter.txt')
468:       Do i = 1, link
469:         Write (37, *) xsueafter(i)
470:       End Do
471:
472:       Open (38, File='38.xafter(subtracted).txt')
473:
474:       Do i = 1, link
475:         Write (38, *) xafter(i)
476:       End Do
477:
478:       Open (68, File='68.FlowPropagation.txt')
479:       Do i = 1, node
480:         b(i, i) = 0
481:         Write (68, *)(b(i,j), j=1, node)
482:       End Do
483:
484:       Call cpu_time(v)
485:       Write (11, *) 'Calculation time 2 (second)', v
486:   !******SUBROUTINE************************************************************
487:     Contains
488:       Subroutine dijikstra(p, t)
489:         Real, Intent (In) :: t(link)
490:         Real *8, Intent (Out) :: p(node, node)
491:         Real *8 :: c(node), cmin(node), fin_io(node)
492:
493:         Do j = 1, node
494:           io = j !io is the source node number
495:           c(:) = 10000000.0 !set partial path cost • ‡
496:           c(io) = 0.0 !set the cost of the source node to 0
497:           cmin(:) = 10000000.0
498:           fin_io(:) = 0 !array for storing the nodes for which calculation has been completed
499:           Do While (minval(cmin,1)<50000000)
500:             Do iii = headnode(io-1) + 1, headnode(io)
501:               i = ls(io, anode(iii))
502:               If (c(io)+t(i)<=c(e(i))) Then
503:                 c(e(i)) = c(io) + t(i) !update with the next link cost with the node
504:                 cmin(e(i)) = c(io) + t(i)
505:               End If
506:             End Do
507:             ncj = minloc(cmin, 1) !identify the location of cmin in the array
508:             cmin(ncj) = 50000000.0 !ineffectiveness of extracted cmin
509:   !**** calculation end judgment (end when all cmin are disabled) ***********
510:             If (minval(cmin,1)==50000000) Then
511:               Go To 100
512:             End If
513:   !******!move to the next node******************
514:             io = ncj
515:             If (fin_io(ncj)==1) Then !when calculation is completed, move to another node
516:               inot = minloc(fin_io, 1)
517:               io = inot
518:             End If
```

```
519:              fin_io(io) = 1
520:            End Do
521:  100       Continue
522:  !******record shortest path results**********************************
523:            p(j, :) = c(:)
524:          End Do
525:
526:        End Subroutine
527:  !******STOP************************************************************
528:  !     First Calculate xij with DFS
529:        Recursive Subroutine calxij(dfs)
530:          Integer dfs, iii
531:
532:          If (j==des(ioi)) Then
533:            dnm = dnm + vn(j)
534:            Do i = 1, k
535:              num(mark(i)) = num(mark(i)) + vn(j)
536:            End Do
537:          Else
538:            Do iii = headnode(j-1) + 1, headnode(j)
539:              If (omega(ioi,alink(iii))/=0) Then
540:                j = e(alink(iii))
541:                k = k + 1
542:                vn(j) = a(alink(iii))*vn(s(alink(iii)))
543:                mark(k) = alink(iii)
544:                Call calxij(dfs+1)
545:                mark(k) = 0
546:                k = k - 1
547:              End If
548:            End Do
549:          End If
550:        End Subroutine
551:  !     Calculate xijrs and xijghrs with DFS
552:        Recursive Subroutine calxij2(dfs)
553:          Integer dfs, iii
554:
555:          If (j==des(ioi)) Then
556:            dnm = dnm + vn(j)
557:            Do i = 1, k - 1
558:              num(mark(i)) = num(mark(i)) + vn(j)
559:              Do ii = i + 1, k
560:                dijgh(mark(i), mark(ii)) = dijgh(mark(i), mark(ii)) + vn(j)
561:              End Do
562:            End Do
563:            num(mark(k)) = num(mark(k)) + vn(j)
564:          Else
565:            Do iii = headnode(j-1) + 1, headnode(j)
566:              If (omega(ioi,alink(iii))/=0) Then
567:                j = e(alink(iii))
568:                k = k + 1
569:                vn(j) = a(alink(iii))*vn(s(alink(iii)))
570:                mark(k) = alink(iii)
```

```
571:              Call calxij2(dfs+1)
572:              mark(k) = 0
573:              k = k - 1
574:            End If
575:          End Do
576:        End If
577:      End Subroutine
578:  !******FINISH SUBROUTINE**********************************
579:    End Program
```

G.7 Solving the semi-DTA CNL model with the sensitivity analysis

```
1:    ! *******semi-dynamic traffic assignment for cross-nested logit model**********
2:    ! **********second algorithm based on stoch3-efficient route ************
3:    ! **************solving one fixed-point problem method ******
4:        Integer i, j, k, ioi, io, count
5:        Integer, Parameter :: node = 5 !number of nodes
6:        Integer, Parameter :: link = 6 !number of links
7:        Integer, Parameter :: nod = 1 !number of od pairs
8:        Real *8, Parameter :: eee = 1.0D-3 !loop termination condition
9:        Real *8, Parameter :: theta = 0.2D0 !logit parameter
10:       Real *8, Parameter :: alpha = 0.15D0 !bpr parameter
11:       Real *8, Parameter :: beta = 4.0D0 !bpr parameter
12:       Real *8, Parameter :: muy = 0.5D0 !cross-nested logit parameter
13:       Real *8, Parameter :: lengtht = 60D0 !length of period (minutes)
14:       Real *8, Parameter :: ro = 2D0 !sra parameter
15:       Real *8, Parameter :: gamma = 0.01D0 !sra parameter
16:       Integer s(link), e(link), ls(node, node) !start and end node and link number
17:       Integer ori(nod), des(nod) !origin, destination of od pairs
18:       Real *8 yde(nod) !od demand of od pairs
19:       Real t(link), cap(link) !t:free-flow travel time,cap: capacity of link
20:       Real *8 erh(link), p(node, node) !erh: stoch3 parameter, p:shortest route
21:       Integer headnode(0:node), anode(link), alink(link) ! adjacent nodes and links
22:       Real *8 ps(link), pe(link)
23:   !ps,pe: shortest path from each origin to start and end node of the link
24:       Real *8 a(link), vn(node)
25:   !a:link likelihood
26:       Integer omega(nod, link), mark(link) !Used to check STOCH3-efficient path
27:       Real *8 xo(link), xn(link) !xo: link traffic flow after, xn:link traffic flow before
28:       Real *8 tt(link) !tt: link travel time after update,
29:       Real *8 x(link)
30:   !x: link travel is assigned
31:       Real *8 tx(link) !tx:derivative of t respect to x
32:   !Variable used to calculate derivative of g respect to t and Detivative of x respect to
      s
33:       Real *8 gradht(link, link) !used to calculate derivative of h respect to t
34:       Real *8 part1(link, link), part2(link, link)
35:       Real *8 www, pivot, part3(link, 2*link)
36:       Real *8 grad(link, link) !Grad: Detivative of x respect to s
37:       Real *8 sij(link), yij(link), yijrs(link), b(node, node) !sij:total eliminated flow
38:       Real *8 xafter(link) !link travel flow after sensitivity analysis
39:       Real *8 xsueafter(link), dijgh(link, link)
40:       Real *8 xijrs(link), xijgh1(link, link) ! Used to calculate xijgh
41:       Real *8 lamda, rg1, rg2, sij1(link), rg(link), maxrg
42:       Real *8 u2ijrs(link), dnm, num(link), num1
43:   !num1: used to calculated num(link) (the nominator of gradht 5.21)
44:       Real *8 vijghrs(link, link)
45:       Real *8 p1ijghrs(link, link), p2ijghrs(link, link), gradht2(link)
46:   !*******************************************************
47:
48:       Open (1, File='1.network.txt', Action='read')
49:       Open (2, File='2.linkparameter.txt', Action='read')
50:       Open (3, File='3.demand.txt', Action='read')
```

```
51:         Open (11, File='11.CalculationTime.txt', Action='write')
52:
53:         Do i = 1, link
54:           Read (1, *) s(i), e(i), j !s(i):start node, e(i):end node
55:           Read (2, *) t(i), cap(i) !t(i):free flow traveltime, cap(i):Capacity of link i
56:           erh(i) = 1.5
57:         End Do
58:         Do i = 1, nod
59:           Read (3, *) ori(i), des(i), yde(i)
60:         End Do
61:     !make network
62:         ls = 0.0
63:         Do i = 1, link
64:           ls(s(i), e(i)) = i
65:         End Do
66:     ! create adjacent nodes and links
67:         headnode(0) = 0
68:         count = 0
69:         Do i = 1, node
70:           Do j = 1, link
71:             If (i==s(j)) Then
72:               count = count + 1
73:               anode(count) = e(j)
74:               alink(count) = j
75:             End If
76:           End Do
77:           headnode(i) = count
78:         End Do
79:
80:         Write (6, *) 'theta', theta
81:         Write (6, *) 'alpha', alpha
82:         Write (6, *) 'beta', beta
83:
84:     !********* Dijkstra method calculation *****************
85:         Call dijikstra(p, t)
86:
87:     !****** dfs algorithm ******************************
88:         xo = 0
89:         omega = 0.0
90:     !***** Creating initial solution based on free-flow travel time***********
91:         a = 0.0
92:         tt = 0.0
93:         Do i = 1, link
94:           tt(i) = t(i)
95:           a(i) = exp(-theta*tt(i))
96:         End Do
97:         Do ioi = 1, nod
98:           io = ori(ioi)
99:           Do i = 1, link
100:            ps(i) = p(io, s(i))
101:            pe(i) = p(io, e(i))
102:          End Do
```

```
103:
104:  !****** !check STOCH3-efficient routes ****************************
105:        Do i = 1, link
106:          If (ps(i)>pe(i)) Go To 100
107:          If ((1+erh(i))*(pe(i)-ps(i))>=t(i)) Then
108:            omega(ioi, i) = 1
109:          End If
110:  100    End Do
111:  !******running the dfs algorithm from origin node r to destination node s ******
112:        vn = 0.0
113:        vn(ori(ioi)) = 1
114:        j = ori(ioi)
115:        k = 0
116:        u2ijrs = 0
117:        Call calxij(1) !Running DFS for the First time
118:        dnm = 0
119:        Do i = 1, link
120:          dnm = dnm + u2ijrs(i)**muy
121:        End Do
122:        num = 0.0
123:        vn = 0.0
124:        vn(ori(ioi)) = 1
125:        j = ori(ioi)
126:        k = 0
127:        Call calxij1(1) !Running DFS for the Second time
128:  !************************************************************
129:  ! recording after loop termination
130:        Do i = 1, link
131:          omega(ioi, i) = 0
132:          x(i) = yde(ioi)*num(i)/dnm
133:          If (x(i)/=0) Then
134:            omega(ioi, i) = 1
135:          End If
136:          xo(i) = xo(i) + x(i)
137:        End Do
138:      End Do
139:  !*******starting calculation round*************************************
140:      xn = 0.0
141:
142:      maxrg = 1
143:      l = 0
144:      Do While (maxrg>eee)
145:  !*****Link travel time and link impedances update**********************
146:        l = l + 1
147:        a = 0.0D0
148:        Do i = 1, link
149:          tt(i) = t(i)*(1+alpha*((xo(i)/cap(i))**beta)) !BPR calculation
150:          a(i) = exp(-theta*(tt(i)))
151:        End Do
152:  !******Direction Finding******
153:        xn = 0.0
154:        Do ioi = 1, nod
```

```
155:  !******running the dfs algorithm from origin node r to destination node s ******
156:          vn = 0.0
157:          vn(ori(ioi)) = 1
158:          j = ori(ioi)
159:          k = 0
160:          u2ijrs = 0
161:          Call calxij(1) !Running DFS for the First time
162:          dnm = 0
163:          Do i = 1, link
164:            dnm = dnm + u2ijrs(i)**muy
165:          End Do
166:          num = 0.0
167:          vn = 0.0
168:          vn(ori(ioi)) = 1
169:          j = ori(ioi)
170:          k = 0
171:          Call calxij1(1) !Running DFS for the Second time
172:  !*************************************************************
173:
174:          Do i = 1, link
175:            xn(i) = xn(i) + yde(ioi)*num(i)/dnm
176:          End Do
177:  !************************************
178:        End Do
179:  !************************************
180:  !c Convergence determination calculation
181:
182:        If (l==1) Then
183:          lamda = 1.0D0
184:        Else
185:          rg2 = 0
186:          Do i = 1, link
187:            rg2 = rg2 + (xn(i)-xo(i))**2.0
188:          End Do
189:          rg2 = sqrt(rg2)
190:          If (rg2>=rg1) Then
191:            lamda = lamda + ro
192:          Else
193:            lamda = lamda + gamma
194:          End If
195:        End If
196:
197:        Do i = 1, link
198:          rg(i) = xn(i) - xo(i)
199:          xn(i) = xo(i) + rg(i)/lamda
200:        End Do
201:
202:  !*********Step 8: Link flow update*****************************
203:        xo = xn
204:        rg1 = rg2
205:        maxrg = maxval(rg)
206:        Write (*, *) 'Round', l, 'maxRG', maxrg
```

```
207:        End Do
208:
209:   !*********RECORDING THE RESULTS*****************************
210:        Write (11, *) 'Round of Static SUE', l, 'Computation gap', maxrg
211:        Call cpu_time(v)
212:        Write (11, *) 'Calculation time 1 (second) ', v
213:        Open (12, File='12.xij.txt')
214:        Do i = 1, link
215:          Write (12, *) xo(i)
216:        End Do
217:        Do i = 1, link
218:          tt(i) = t(i)*(1+alpha*((xo(i)/cap(i))**beta)) !BPR calculation
219:        End Do
220:        Open (13, File='13.tij.txt')
221:        Do i = 1, link
222:          Write (13, *) tt(i)
223:        End Do
224:        Open (16, File='16.omega.txt')
225:        Do i = 1, nod
226:          Write (16, *)(omega(i,j), j=1, link)
227:        End Do
228:        Open (17, File='17.x,xn,a,num.txt')
229:        Do i = 1, link
230:          Write (17, *) x(i), xn(i), a(i), num(i)
231:        End Do
232:        Open (18, File='18.vn.txt')
233:        Do i = 1, node
234:          Write (18, *) vn(i)
235:        End Do
236:
237:   !****** STEP2: Calculate Gradient*****************************
238:        mark = 0
239:        gradht = 0.0
240:        sij = 0.0
241:        xijrs = 0.0
242:        l = 0.0
243:
244:   !***** Calculation of link likeli-hood ********************
245:        a = 0.0
246:        Do i = 1, link
247:          a(i) = exp(-theta*(tt(i)))
248:        End Do
249:
250:   !****** calculating derivitives of g with respect to t*****************
251:        Do ioi = 1, nod
252:
253:          io = ori(ioi)
254:          vn = 0.0
255:          vn(ori(ioi)) = 1
256:          j = ori(ioi)
257:          k = 0
258:          u2ijrs = 0
```

```
259:          vijghrs = 0.0D0
260:          Call calxij2(1)
261:          dnm = 0
262:          Do i = 1, link
263:            dnm = dnm + u2ijrs(i)**muy
264:          End Do
265:          num = 0.0
266:          vn = 0.0
267:          vn(ori(ioi)) = 1
268:          j = ori(ioi)
269:          k = 0
270:          p1ijghrs = 0
271:          p2ijghrs = 0
272:          gradht2 = 0.0
273:          Call calxij3(1)
274:          Do i = 1, link
275:            Do j = 1, link
276:              If (u2ijrs(j)/=0) Then
277:                gradht2(i) = gradht2(i) - theta*u2ijrs(j)**(muy-1)*vijghrs(j, i)
278:              End If
279:            End Do
280:          End Do
281:    ! recording after loop termination
282:    !*******calculating node traffic flow, link proportion, xijgh and gradht*****
283:          Do i = 1, link
284:            If (omega(ioi,i)/=0) Then
285:              Do j = 1, link
286:                If (omega(ioi,j)/=0 .And. dnm/=0) Then
287:                  gradht(i, j) = gradht(i, j) + yde(ioi)*((-theta/muy)*(p1ijghrs &
288:                    (i,j)+p2ijghrs(i,j))/dnm-gradht2(j)*(num(i)/dnm)/dnm)
289:                End If
290:              End Do
291:            End If
292:          End Do
293:          vn = 0.0D0
294:          vn(ori(ioi)) = 1
295:          j = ori(ioi)
296:          k = 0
297:          dijgh = 0.0
298:          Call caldijgh(1)
299:          xijgh1 = 0.0D0
300:    !     write (*,*) 'finish_xijrs'
301:          Do i = 1, link
302:            Do j = 1, link
303:              xijgh1(i, j) = yde(ioi)*dijgh(i, j)/dnm
304:              If (xijgh1(i,j)/=0) Then
305:                sij(j) = sij(j) + xijgh1(i, j)/lengtht*tt(i)
306:              End If
307:            End Do
308:          End Do
309:        End Do
310:    !*************************************************************
```

```
311:  !*******Record results*****
312:        Call cpu_time(v)
313:        Write (11, *) 'Calculation time 1 (second)', v
314:
315:        Open (22, File='22.gradht.txt')
316:        Do i = 1, link
317:          Write (22, *) gradht(i, :)
318:        End Do
319:        Write (*, *) 'finish_gradht'
320:
321:  !***** Calculating derivitives of implicit BPR function*****************
322:        Do i = 1, link
323:          tx(i) = t(i)*beta*alpha*(xo(i)**(beta-1))/(cap(i)**beta)
324:        End Do
325:        Open (25, File='25.tx.txt')
326:        Do i = 1, link
327:          Write (25, *) tx(i)
328:        End Do
329:  !*******Calculation Gradient**********
330:        part1 = 0.0
331:        part2 = 0.0
332:        part3 = 0.0
333:        Do i = 1, link
334:          Do j = 1, link
335:            part1(i, j) = gradht(i, j)*tx(j)
336:          End Do
337:        End Do
338:
339:        Do i = 1, link
340:          Do j = 1, link
341:            If (i==j) Then
342:              part2(i, j) = 1 - part1(i, j)
343:            Else
344:              part2(i, j) = -part1(i, j)
345:            End If
346:          End Do
347:        End Do
348:        Do i = 1, link
349:          Do j = 1, link
350:            part3(i, j) = part2(i, j)
351:          End Do
352:        End Do
353:
354:        Do i = 1, link
355:          Do j = 1, link
356:            part3(i, j+link) = 0
357:            part3(i, i+link) = 1
358:          End Do
359:        End Do
360:
361:        Do k = 1, link
362:          pivot = part3(k, k)
```

```
363:          If (pivot==0) Then
364:            i = k + 1
365:            Do While (pivot==0 .And. i<link)
366:              pivot = part3(i, k)
367:              i = i + 1
368:            End Do
369:            If (pivot==0) Then
370:              Stop
371:            Else
372:              Do j = 1, 2*link
373:                www = part3(k, j)
374:                i = i - 1
375:                part3(k, j) = part3(i, j)
376:                part3(i, j) = www
377:              End Do
378:            End If
379:          End If
380:          Do j = 1, link*2
381:            part3(k, j) = part3(k, j)/pivot
382:          End Do
383:
384:          Do i = 1, link
385:            If (i/=k) Then
386:              www = part3(i, k)
387:              Do j = 1, link*2
388:                part3(i, j) = part3(i, j) - www*part3(k, j)
389:              End Do
390:            End If
391:          End Do
392:        End Do
393:
394:        part2 = 0.0
395:        Do i = 1, link
396:          Do j = 1, link
397:            part2(i, j) = part3(i, j+link)
398:          End Do
399:        End Do
400:        grad = 0
401:        grad = -matmul(part2, part1)
402:
403:        Open (26, File='26.gradxs.txt')
404:        Do i = 1, link
405:          Write (26, *) grad(i, :)
406:        End Do
407:
408:        Write (*, *) 'finish_Gradient'
409:    !******STEP3: SOLVING FIXED-PONT PROBLEM OF ELIMINATED
        FLOW*****************
410:        l = 0
411:        maxrg = 1.0D0
412:        xijgh1 = 0.0
413:        Do While (maxrg>0.0001)
```

```
414:        l = l + 1
415:        xsueafter = 0.0D0
416:        xsueafter = matmul(grad, sij)
417:        Do i = 1, link
418:          xsueafter(i) = xo(i) + xsueafter(i)
419:          tt(i) = t(i)*(1+alpha*(((xsueafter(i)-sij(i))/cap(i))**beta))
420:          a(i) = exp(-theta*(tt(i)))
421:        End Do
422:        sij1 = 0.0D0
423:        Do ioi = 1, nod
424:
425:          vn = 0.0
426:          vn(ori(ioi)) = 1
427:          j = ori(ioi)
428:          k = 0
429:          u2ijrs = 0
430:          Call calxij(1) !Running DFS for the First time
431:          dnm = 0
432:          Do i = 1, link
433:            dnm = dnm + u2ijrs(i)**muy
434:          End Do
435:
436:          vn = 0.0D0
437:          vn(ori(ioi)) = 1
438:          j = ori(ioi)
439:          k = 0
440:          dijgh = 0.0D0
441:          Call caldijgh(1)
442:          xijgh1 = 0.0D0
443:          Do i = 1, link
444:            Do j = 1, link
445:              xijgh1(i, j) = yde(ioi)*dijgh(i, j)/dnm
446:              If (xijgh1(i,j)/=0) Then
447:                sij1(j) = sij1(j) + xijgh1(i, j)/lengtht*tt(i)
448:              End If
449:            End Do
450:          End Do
451:        End Do
452:        If (l==1) Then
453:          lamda = 1.0D0
454:        Else
455:          rg2 = 0
456:          Do i = 1, link
457:            rg2 = rg2 + (sij(i)-sij1(i))**2.0
458:          End Do
459:          rg2 = sqrt(rg2)
460:          If (rg2>=rg1) Then
461:            lamda = lamda + ro
462:          Else
463:            lamda = lamda + gamma
464:          End If
465:        End If
```

```
466:          maxrg = 0.0D0
467:          Do i = 1, link
468:            rg(i) = sij1(i) - sij(i)
469:            sij(i) = sij(i) + rg(i)/lamda
470:            If (abs(rg(i))>maxrg) Then
471:              maxrg = abs(rg(i))
472:            End If
473:          End Do
474:          rg1 = rg2
475:          Write (6, *) l, maxrg
476:        End Do
477:   !Calculate link travel flow, residual link flow and link travel time at the semi-DTA
478:
479:        xsueafter = matmul(grad, sij)
480:        Do i = 1, link
481:          xsueafter(i) = xo(i) + xsueafter(i)
482:          xafter(i) = max((xsueafter(i)-sij(i)), 0D0)
483:          tt(i) = t(i)*(1+alpha*(((xsueafter(i)-sij(i))/cap(i))**beta))
484:          a(i) = exp(-theta*(tt(i)))
485:        End Do
486:
487:        yij = 0.0
488:        b = 0.0
489:   !********DFS algorithm with each OD pair*****************************
490:        Do ioi = 1, nod
491:   !******running the dfs algorithm from origin node r to destination node s ******
492:          vn = 0.0
493:          vn(ori(ioi)) = 1
494:          j = ori(ioi)
495:          k = 0
496:          u2ijrs = 0
497:          Call calxij(1) !Running DFS for the First time
498:          dnm = 0
499:          Do i = 1, link
500:            dnm = dnm + u2ijrs(i)**muy
501:          End Do
502:          num = 0.0
503:          vn = 0.0
504:          vn(ori(ioi)) = 1
505:          j = ori(ioi)
506:          k = 0
507:          Call calxij1(1) !Running DFS for the Second time
508:   !************************************************************
509:
510:          xijrs = 0.0D0
511:          Do i = 1, link
512:            xijrs(i) = yde(ioi)*num(i)/dnm
513:          End Do
514:          yijrs = 0.0
515:          Do i = 1, link
516:            yijrs(i) = xijrs(i)/lengtht*tt(i)
517:            yij(i) = yij(i) + yijrs(i)
```

```
518:         End Do
519:
520:         Do i = 1, link
521:           If (yijrs(i)/=0) Then
522:             b(e(i), des(ioi)) = b(e(i), des(ioi)) + yijrs(i)
523:           End If
524:         End Do
525:       End Do
526:
527:       Open (28, File='28.sij.txt')
528:       Do i = 1, link
529:         Write (28, *) sij(i)
530:       End Do
531:       Open (29, File='29.yij.txt')
532:       Do i = 1, link
533:         Write (29, *) yij(i)
534:       End Do
535:       Open (37, File='37.xSUEafter.txt')
536:       Do i = 1, link
537:         Write (37, *) xsueafter(i)
538:       End Do
539:
540:       Open (38, File='38.xafter(subtracted).txt')
541:
542:       Do i = 1, link
543:         Write (38, *) xafter(i)
544:       End Do
545:
546:       Open (68, File='68.FlowPropagation.txt')
547:       Do i = 1, node
548:         b(i, i) = 0
549:         Write (68, *)(b(i,j), j=1, node)
550:       End Do
551:
552:       Call cpu_time(v)
553:       Write (11, *) 'Calculation time 2 (second)', v
554:   !******SUBROUTINE**********************************************************
      *******
555:     Contains
556:       Subroutine dijikstra(p, t)
557:         Real, Intent (In) :: t(link)
558:         Real *8, Intent (Out) :: p(node, node)
559:         Real *8 :: c(node), cmin(node), fin_io(node)
560:
561:         Do j = 1, node
562:           io = j !io is the source node number
563:           c(:) = 10000000.0 !set partial path cost □‡
564:           c(io) = 0.0 !set the cost of the source node to 0
565:           cmin(:) = 10000000.0
566:           fin_io(:) = 0 !array for storing the nodes for which calculation has been
      completed
567:           Do While (minval(cmin,1)<50000000)
```

```
568:            Do iii = headnode(io-1) + 1, headnode(io)
569:              i = ls(io, anode(iii))
570:              If (c(io)+t(i)<=c(e(i))) Then
571:                c(e(i)) = c(io) + t(i) !update with the next link cost with the node
572:                cmin(e(i)) = c(io) + t(i)
573:              End If
574:            End Do
575:            ncj = minloc(cmin, 1) !identify the location of cmin in the array
576:            cmin(ncj) = 50000000.0 !ineffectiveness of extracted cmin
577:  !**** calculation end judgment (end when all cmin are disabled) ***********
578:            If (minval(cmin,1)==50000000) Then
579:              Go To 100
580:            End If
581:  !******!move to the next node*****************
582:            io = ncj
583:            If (fin_io(ncj)==1) Then !when calculation is completed, move to another
      node
584:              inot = minloc(fin_io, 1)
585:              io = inot
586:            End If
587:            fin_io(io) = 1
588:          End Do
589:  100       Continue
590:  !******record shortest path results***********************************
591:          p(j, :) = c(:)
592:        End Do
593:
594:      End Subroutine
595:  !******STOP************************************************************
596:  ! calculate xij with dfs
597:      Recursive Subroutine calxij(dfs)
598:        Integer dfs, iii
599:
600:        If (j==des(ioi)) Then
601:          crou = 0.0
602:          Do i = 1, k
603:            crou = crou + t(mark(i))
604:          End Do
605:          Do i = 1, k
606:            u2ijrs(mark(i)) = u2ijrs(mark(i)) + (t(mark(i))/crou*vn(j))**( &
607:              1.0D0/muy)
608:          End Do
609:        Else
610:          Do iii = headnode(j-1) + 1, headnode(j)
611:            If (omega(ioi,alink(iii))/=0) Then
612:              j = e(alink(iii))
613:              k = k + 1
614:              vn(j) = a(alink(iii))*vn(s(alink(iii)))
615:              mark(k) = alink(iii)
616:              Call calxij(dfs+1)
617:              mark(k) = 0
618:              k = k - 1
```

```
619:              End If
620:            End Do
621:          End If
622:        End Subroutine
623:   ! calculate xij with dfs
624:        Recursive Subroutine calxij1(dfs)
625:          Integer dfs, iii
626:
627:          If (j==des(ioi)) Then
628:            crou = 0.0
629:            Do i = 1, k
630:              crou = crou + t(mark(i))
631:            End Do
632:            num1 = 0.0
633:            Do i = 1, k
634:              num1 = num1 + (t(mark(i))/crou*vn(j))**(1.0D0/muy)*(u2ijrs(mark(i) &
635:                )**(muy-1.0D0))
636:            End Do
637:            Do i = 1, k
638:              num(mark(i)) = num(mark(i)) + num1
639:            End Do
640:          Else
641:            Do iii = headnode(j-1) + 1, headnode(j)
642:              If (omega(ioi,alink(iii))/=0) Then
643:                j = e(alink(iii))
644:                k = k + 1
645:                vn(j) = a(alink(iii))*vn(s(alink(iii)))
646:                mark(k) = alink(iii)
647:                Call calxij1(dfs+1)
648:                mark(k) = 0
649:                k = k - 1
650:              End If
651:            End Do
652:          End If
653:        End Subroutine
654:   ! calculate gradient with dfs
655:        Recursive Subroutine calxij2(dfs)
656:          Integer dfs, iii
657:
658:          If (j==des(ioi)) Then
659:            crou = 0.0
660:            Do i = 1, k
661:              crou = crou + t(mark(i))
662:            End Do
663:            Do i = 1, k
664:              u2ijrs(mark(i)) = u2ijrs(mark(i)) + (t(mark(i))/crou*vn(j))**( &
665:                1.0D0/muy)
666:            End Do
667:            Do i = 1, k
668:              Do ii = 1, k
669:                vijghrs(mark(i), mark(ii)) = vijghrs(mark(i), mark(ii)) + &
670:                  (t(mark(i))/crou*vn(j))**(1.0D0/muy)
```

```
671:             End Do
672:           End Do
673:
674:         Else
675:           Do iii = headnode(j-1) + 1, headnode(j)
676:             If (omega(ioi,alink(iii))/=0) Then
677:               j = e(alink(iii))
678:               k = k + 1
679:               vn(j) = a(alink(iii))*vn(s(alink(iii)))
680:               mark(k) = alink(iii)
681:               Call calxij2(dfs+1)
682:               mark(k) = 0
683:               k = k - 1
684:             End If
685:           End Do
686:         End If
687:       End Subroutine
688:  ! calculate p1ijghrs,p1ijghrs and grad4 with dfs
689:       Recursive Subroutine calxij3(dfs)
690:         Integer dfs, iii
691:
692:         If (j==des(ioi)) Then
693:           crou = 0.0
694:           Do i = 1, k
695:             crou = crou + t(mark(i))
696:           End Do
697:           num1 = 0.0
698:           Do i = 1, k
699:             num1 = num1 + (t(mark(i))/crou*vn(j))**(1.0D0/muy)*(u2ijrs(mark(i) &
700:               )**(muy-1.0D0))
701:           End Do
702:           Do i = 1, k
703:             num(mark(i)) = num(mark(i)) + num1
704:           End Do
705:
706:           Do i = 1, k
707:             Do ii = 1, k
708:  !p1ijghrs(ij,mn) for link mn is also an element of the route including ij and gh
709:               Do iii = 1, k
710:                 p1ijghrs(mark(i), mark(iii)) = p1ijghrs(mark(i), mark(iii)) + &
711:                   (t(mark(ii))/crou*vn(j))**(1.0D0/muy)* &
712:                   (u2ijrs(mark(ii))**(muy-1.0D0))
713:               End Do
714:  !p1ijghrs(ij,mn) for link mn is not an element of the route including ij but
      vijghrs(gh, mn).ne.0
715:               Do iii = 1, link
716:                 If (vijghrs(mark(ii),iii)/=0 .And. u2ijrs(mark(ii))/=0) Then
717:                   p2ijghrs(mark(i), iii) = p2ijghrs(mark(i), iii) + &
718:                     (muy-1)*(t(mark(ii))/crou*vn(j))**(1.0D0/muy)* &
719:                     (u2ijrs(mark(ii))**(muy-2.0D0))*vijghrs(mark(ii), iii)
720:                 End If
721:               End Do
```

```
722:
723:              End Do
724:            End Do
725:
726:          Else
727:            Do iii = headnode(j-1) + 1, headnode(j)
728:              If (omega(ioi,alink(iii))/=0) Then
729:                j = e(alink(iii))
730:                k = k + 1
731:                vn(j) = a(alink(iii))*vn(s(alink(iii)))
732:                mark(k) = alink(iii)
733:                Call calxij3(dfs+1)
734:                mark(k) = 0
735:                k = k - 1
736:              End If
737:            End Do
738:          End If
739:        End Subroutine
740:   ! calculate dijgh
741:        Recursive Subroutine caldijgh(dfs)
742:          Integer dfs, iii
743:
744:          If (j==des(ioi)) Then
745:            crou = 0.0
746:            Do i = 1, k
747:              crou = crou + t(mark(i))
748:            End Do
749:            num1 = 0.0
750:            Do i = 1, k
751:              num1 = num1 + (t(mark(i))/crou*vn(j))**(1.0D0/muy)*(u2ijrs(mark(i) &
752:                )**(muy-1.0D0))
753:            End Do
754:            Do i = 1, k - 1
755:              Do j = i + 1, k
756:                dijgh(mark(i), mark(j)) = dijgh(mark(i), mark(j)) + num1
757:              End Do
758:            End Do
759:          Else
760:            Do iii = headnode(j-1) + 1, headnode(j)
761:              If (omega(ioi,alink(iii))/=0) Then
762:                j = e(alink(iii))
763:                k = k + 1
764:                vn(j) = a(alink(iii))*vn(s(alink(iii)))
765:                mark(k) = alink(iii)
766:                Call caldijgh(dfs+1)
767:                mark(k) = 0
768:                k = k - 1
769:              End If
770:            End Do
771:          End If
772:        End Subroutine
773:
```

```
774:   !******FINISH SUBROUTINE**********************************
775:       End Program
```

G.8 Solving the semi-DTA q-generalized model with the sensitivity analysis

```
1:      ! *******semi-dynamic traffic assignment for q-generalized logit model**********
2:      ! **********second algorithm based on stoch3-efficient route ************
3:      ! **************solving one fixed-point problem method ************************
4:          Integer i, j, k, ioi, io, count
5:          Integer, Parameter :: node = 5 !number of nodes
6:          Integer, Parameter :: link = 6 !number of links
7:          Integer, Parameter :: nod = 1 !number of od pairs
8:          Real *8, Parameter :: eee = 1.0D-3 !loop termination condition
9:          Real *8, Parameter :: theta = 0.2D0 !logit parameter
10:         Real *8, Parameter :: alpha = 0.15D0 !bpr parameter
11:         Real *8, Parameter :: beta = 4.0D0 !bpr parameter
12:         Real *8, Parameter :: kiu = 0.5D0 !q parameter
13:         Real *8, Parameter :: lengtht = 60D0 !length of period (minutes)
14:         Real *8, Parameter :: ro = 2D0 !sra parameter
15:         Real *8, Parameter :: gamma = 0.01D0 !sra parameter
16:         Integer s(link), e(link), ls(node, node) !start and end node and link number
17:         Integer ori(nod), des(nod) !origin, destination of od pairs
18:         Real *8 yde(nod) !od demand of od pairs
19:         Real t(link), cap(link) !t:free-flow travel time,cap: capacity of link
20:         Real *8 erh(link), p(node, node) !erh: stoch3 parameter, p:shortest route
21:         Integer headnode(0:node), anode(link), alink(link) ! adjacent nodes and links
22:         Real *8 ps(link), pe(link)
23:     !ps,pe: shortest path from each origin to start and end node of the link
24:         Real *8 a(link), num(link), dnm
25:     !a:link likelihood
26:         Integer omega(nod, link), mark(link) !Used to check STOCH3-efficient path
27:         Real *8 xo(link), xn(link) !xo: link traffic flow after, xn:link traffic flow before
28:         Real *8 tt(link) !tt: link travel time after update,
29:         Real *8 x(link)
30:     !x: link travel is assigned
31:         Real *8 tx(link) !tx:derivative of t respect to x
32:     !Variable used to calculate derivative of g respect to t and Detivative of x respect to s
33:         Real *8 b2gh(link), b1ijgh(link, link)
34:         Real *8 gradgt(link, link) !used to calculate derivative of u respect to t
35:         Real *8 part1(link, link), part2(link, link)
36:         Real *8 www, pivot, part3(link, 2*link)
37:         Real *8 grad(link, link) !Grad: Detivative of x respect to s
38:         Real *8 sij(link), yij(link), yijrs(link), b(node, node) !sij:total eliminated flow
39:         Real *8 xafter(link) !link travel flow after sensitivity analysis
40:         Real *8 xsueafter(link), dijgh(link, link)
41:         Real *8 xijrs(link), xijgh1(link, link)
42:         Real *8 lamda, rg1, rg2, sij1(link), rg(link), maxrg
43:     !*****************************************************
44:
45:         Open (1, File='1.network.txt', Action='read')
46:         Open (2, File='2.linkparameter.txt', Action='read')
47:         Open (3, File='3.demand.txt', Action='read')
48:         Open (11, File='11.CalculationTime.txt', Action='write')
49:
50:         Do i = 1, link
51:           Read (1, *) s(i), e(i), j !s(i):start node, e(i):end node
```

```
52:         Read (2, *) t(i), cap(i) !t(i):free flow traveltime, cap(i):Capacity of link i
53:         erh(i) = 1.5
54:       End Do
55:       Do i = 1, nod
56:         Read (3, *) ori(i), des(i), yde(i)
57:       End Do
58:   !make network
59:       ls = 0.0
60:       Do i = 1, link
61:         ls(s(i), e(i)) = i
62:       End Do
63:   ! create adjacent nodes and links
64:       headnode(0) = 0
65:       count = 0
66:       Do i = 1, node
67:         Do j = 1, link
68:           If (i==s(j)) Then
69:             count = count + 1
70:             anode(count) = e(j)
71:             alink(count) = j
72:           End If
73:         End Do
74:         headnode(i) = count
75:       End Do
76:
77:       Write (6, *) 'theta', theta
78:       Write (6, *) 'alpha', alpha
79:       Write (6, *) 'beta', beta
80:
81:   !********* Dijkstra method calculation ****************
82:       Call dijikstra(p, t)
83:
84:   !****** dfs algorithm ******************************
85:       xo = 0
86:       omega = 0.0
87:   !***** Creating initial solution based on free-flow travel time***********
88:       a = 0.0
89:       tt = 0.0
90:       Do i = 1, link
91:         tt(i) = t(i)
92:         a(i) = exp(-theta*tt(i))
93:       End Do
94:       Do ioi = 1, nod
95:         io = ori(ioi)
96:         Do i = 1, link
97:           ps(i) = p(io, s(i))
98:           pe(i) = p(io, e(i))
99:         End Do
100:
101:  !****** !check STOCH3-efficient routes ****************************
102:        Do i = 1, link
103:          If (ps(i)>pe(i)) Go To 100
```

```
104:          If ((1+erh(i))*(pe(i)-ps(i))>=t(i)) Then
105:            omega(ioi, i) = 1
106:          End If
107:  100     End Do
108:  !******running the dfs algorithm from origin node r to destination node s ******
109:        num = 0.0
110:        j = ori(ioi)
111:        k = 0
112:        dnm = 0
113:        Call calxij(1)
114:  !************************************************************
115:  ! recording after loop termination
116:        Do i = 1, link
117:          omega(ioi, i) = 0
118:          x(i) = yde(ioi)*num(i)/dnm
119:          If (x(i)/=0) Then
120:            omega(ioi, i) = 1
121:          End If
122:          xo(i) = xo(i) + x(i)
123:        End Do
124:      End Do
125:  !*******starting calculation round*************************************
126:      xn = 0.0
127:
128:      maxrg = 1
129:      l = 0
130:      Do While (maxrg>eee)
131:  !*****Link travel time and link impedances update**********************
132:        l = l + 1
133:        a = 0.0D0
134:        Do i = 1, link
135:          tt(i) = t(i)*(1+alpha*((xo(i)/cap(i))**beta)) !BPR calculation
136:          a(i) = exp(-theta*(tt(i)))
137:        End Do
138:  !***** STOCH 3 algorithm ******************************
139:        xn = 0.0
140:        Do ioi = 1, nod
141:          num = 0.0
142:          j = ori(ioi)
143:          k = 0
144:          dnm = 0
145:          Call calxij(1)
146:  !*************************
147:          Do i = 1, link
148:            xn(i) = xn(i) + yde(ioi)*num(i)/dnm
149:          End Do
150:  !************************************
151:        End Do
152:  !************************************
153:  !c Convergence determination calculation
154:
155:        If (l==1) Then
```

```
156:          lamda = 1.0D0
157:        Else
158:          rg2 = 0
159:          Do i = 1, link
160:            rg2 = rg2 + (xn(i)-xo(i))**2.0
161:          End Do
162:          rg2 = sqrt(rg2)
163:          If (rg2>=rg1) Then
164:            lamda = lamda + ro
165:          Else
166:            lamda = lamda + gamma
167:          End If
168:        End If
169:
170:        Do i = 1, link
171:          rg(i) = xn(i) - xo(i)
172:          xn(i) = xo(i) + rg(i)/lamda
173:        End Do
174:
175:  !*********Step 8: Link flow update*****************************
176:        xo = xn
177:        rg1 = rg2
178:        maxrg = maxval(rg)
179:        Write (*, *) 'Round', l, 'maxRG', maxrg
180:      End Do
181:
182:  !*********RECORDING THE RESULTS*****************************
183:      Write (11, *) 'Round of Static SUE', l, 'Computation gap', maxrg
184:      Call cpu_time(v)
185:      Write (11, *) 'Calculation time 1 (second) ', v
186:      Open (12, File='12.xij.txt')
187:      Do i = 1, link
188:        Write (12, *) xo(i)
189:      End Do
190:      Do i = 1, link
191:        tt(i) = t(i)*(1+alpha*((xo(i)/cap(i))**beta)) !BPR calculation
192:      End Do
193:      Open (13, File='13.tij.txt')
194:      Do i = 1, link
195:        Write (13, *) tt(i)
196:      End Do
197:      Open (16, File='16.omega.txt')
198:      Do i = 1, nod
199:        Write (16, *)(omega(i,j), j=1, link)
200:      End Do
201:      Open (17, File='17.x,xn,a,num.txt')
202:      Do i = 1, link
203:        Write (17, *) x(i), xn(i), a(i), num(i)
204:      End Do
205:  !****** STEP2: Calculate Gradient*****************************
206:      mark = 0
207:      gradgt = 0.0
```

```
208:        sij = 0.0
209:        l = 0.0
210:    !***** Calculation of link likeli-hood ********************
211:        a = 0.0
212:        Do i = 1, link
213:          a(i) = exp(-theta*(tt(i)))
214:        End Do
215:
216:    !****** calculating derivitives of g with respect to t*****************
217:        Do ioi = 1, nod
218:          io = ori(ioi)
219:          num = 0.0D0
220:          j = ori(ioi)
221:          k = 0
222:          dnm = 0.0D0
223:          dijgh = 0.0
224:          Call caldijgh(1)
225:
226:          xijgh1 = 0.0D0
227:          Do i = 1, link
228:            Do j = 1, link
229:              xijgh1(i, j) = yde(ioi)*dijgh(i, j)/dnm
230:              If (xijgh1(i,j)/=0) Then
231:                sij(j) = sij(j) + xijgh1(i, j)/lengtht*tt(i)
232:              End If
233:            End Do
234:          End Do
235:          num = 0.0D0
236:          vn = 0.0D0
237:          j = ori(ioi)
238:          k = 0
239:          b1ijgh = 0.0D0
240:          b2gh = 0.0D0
241:          Call calxij2(1)
242:          Do i = 1, link
243:            If (omega(ioi,i)/=0) Then
244:              Do j = 1, link
245:                If (omega(ioi,j)/=0) Then
246:                  gradgt(i, j) = gradgt(i, j) - theta*yde(ioi)*(b1ijgh(i,j)*dnm- &
247:                    b2gh(j)*num(i))/dnm**2
248:                End If
249:              End Do
250:            End If
251:          End Do
252:        End Do
253:    !*************************************************************
254:    !*******Record results*****
255:        Call cpu_time(v)
256:        Write (11, *) 'Calculation time 1 (second)', v
257:
258:        Open (22, File='22.gradgt.txt')
259:        Do i = 1, link
```

```
260:          Write (22, *) gradgt(i, :)
261:        End Do
262:        Write (*, *) 'finish_gradgt'
263:
264:    !***** Calculating derivitives of implicit BPR function*****************
265:        Do i = 1, link
266:          tx(i) = t(i)*beta*alpha*(xo(i)**(beta-1))/(cap(i)**beta)
267:        End Do
268:        Open (25, File='25.tx.txt')
269:        Do i = 1, link
270:          Write (25, *) tx(i)
271:        End Do
272:    !*******Calculation Gradient**********
273:        part1 = 0.0
274:        part2 = 0.0
275:        part3 = 0.0
276:        Do i = 1, link
277:          Do j = 1, link
278:            part1(i, j) = gradgt(i, j)*tx(j)
279:          End Do
280:        End Do
281:
282:        Do i = 1, link
283:          Do j = 1, link
284:            If (i==j) Then
285:              part2(i, j) = 1 - part1(i, j)
286:            Else
287:              part2(i, j) = -part1(i, j)
288:            End If
289:          End Do
290:        End Do
291:        Do i = 1, link
292:          Do j = 1, link
293:            part3(i, j) = part2(i, j)
294:          End Do
295:        End Do
296:
297:        Do i = 1, link
298:          Do j = 1, link
299:            part3(i, j+link) = 0
300:            part3(i, i+link) = 1
301:          End Do
302:        End Do
303:
304:        Do k = 1, link
305:          pivot = part3(k, k)
306:          If (pivot==0) Then
307:            i = k + 1
308:            Do While (pivot==0 .And. i<link)
309:              pivot = part3(i, k)
310:              i = i + 1
311:            End Do
```

```
312:          If (pivot==0) Then
313:            Stop
314:          Else
315:            Do j = 1, 2*link
316:              www = part3(k, j)
317:              i = i - 1
318:              part3(k, j) = part3(i, j)
319:              part3(i, j) = www
320:            End Do
321:          End If
322:        End If
323:        Do j = 1, link*2
324:          part3(k, j) = part3(k, j)/pivot
325:        End Do
326:
327:        Do i = 1, link
328:          If (i/=k) Then
329:            www = part3(i, k)
330:            Do j = 1, link*2
331:              part3(i, j) = part3(i, j) - www*part3(k, j)
332:            End Do
333:          End If
334:        End Do
335:      End Do
336:
337:      part2 = 0.0
338:      Do i = 1, link
339:        Do j = 1, link
340:          part2(i, j) = part3(i, j+link)
341:        End Do
342:      End Do
343:      grad = 0
344:      grad = -matmul(part2, part1)
345:
346:      Open (26, File='26.gradxs.txt')
347:      Do i = 1, link
348:        Write (26, *) grad(i, :)
349:      End Do
350:
351:      Write (*, *) 'finish_Gradient'
352: !******STEP3: SOLVING FIXED-PONT PROBLEM OF ELIMINATED
     FLOW******************
353:      l = 0
354:      maxrg = 1.0D0
355:      Do While (maxrg>0.0001)
356:        l = l + 1
357:        xsueafter = 0.0D0
358:        xsueafter = matmul(grad, sij)
359:        Do i = 1, link
360:          xsueafter(i) = xo(i) + xsueafter(i)
361:          tt(i) = t(i)*(1+alpha*(((xsueafter(i)-sij(i))/cap(i))**beta))
362:          a(i) = exp(-theta*(tt(i)))
```

```
363:          End Do
364:          sij1 = 0.0D0
365:          Do ioi = 1, nod
366:            io = ori(ioi)
367:            num = 0.0D0
368:            j = ori(ioi)
369:            k = 0
370:            vn = 0.0D0
371:            dnm = 0.0D0
372:            dijgh = 0.0D0
373:            Call caldijgh(1)
374:            xijgh1 = 0.0D0
375:            Do i = 1, link
376:              Do j = 1, link
377:                xijgh1(i, j) = yde(ioi)*dijgh(i, j)/dnm
378:                If (xijgh1(i,j)/=0) Then
379:                  sij1(j) = sij1(j) + xijgh1(i, j)/lengtht*tt(i)
380:                End If
381:              End Do
382:            End Do
383:          End Do
384:          If (l==1) Then
385:            lamda = 1.0D0
386:          Else
387:            rg2 = 0
388:            Do i = 1, link
389:              rg2 = rg2 + (sij(i)-sij1(i))**2.0
390:            End Do
391:            rg2 = sqrt(rg2)
392:            If (rg2>=rg1) Then
393:              lamda = lamda + ro
394:            Else
395:              lamda = lamda + gamma
396:            End If
397:          End If
398:          maxrg = 0.0D0
399:          Do i = 1, link
400:            rg(i) = sij1(i) - sij(i)
401:            sij(i) = sij(i) + rg(i)/lamda
402:            If (abs(rg(i))>maxrg) Then
403:              maxrg = abs(rg(i))
404:            End If
405:          End Do
406:          rg1 = rg2
407:          Write (6, *) l, maxrg
408:        End Do
409:   !Calculate link travel flow, residual link flow and link travel time at the semi-DTA
410:
411:        xsueafter = matmul(grad, sij)
412:        Do i = 1, link
413:          xsueafter(i) = xo(i) + xsueafter(i)
414:          xafter(i) = max((xsueafter(i)-sij(i)), 0D0)
```

```
415:          tt(i) = t(i)*(1+alpha*(((xsueafter(i)-sij(i))/cap(i))**beta))
416:          a(i) = exp(-theta*(tt(i)))
417:        End Do
418:
419:        yij = 0.0
420:        b = 0.0
421:  !********DFS algorithm with each OD pair*****************************
422:        Do ioi = 1, nod
423:          io = ori(ioi)
424:          num = 0.0D0
425:          j = ori(ioi)
426:          k = 0
427:          dnm = 0.0D0
428:          Call calxij(1)
429:          Do i = 1, link
430:            xijrs(i) = yde(ioi)*num(i)/dnm
431:          End Do
432:          yijrs = 0.0
433:          Do i = 1, link
434:            yijrs(i) = xijrs(i)/lengtht*tt(i)
435:            yij(i) = yij(i) + yijrs(i)
436:          End Do
437:
438:          Do i = 1, link
439:            If (yijrs(i)/=0) Then
440:              b(e(i), des(ioi)) = b(e(i), des(ioi)) + yijrs(i)
441:            End If
442:          End Do
443:        End Do
444:
445:        Open (28, File='28.sij.txt')
446:        Do i = 1, link
447:          Write (28, *) sij(i)
448:        End Do
449:        Open (29, File='29.yij.txt')
450:        Do i = 1, link
451:          Write (29, *) yij(i)
452:        End Do
453:        Open (37, File='37.xSUEafter.txt')
454:        Do i = 1, link
455:          Write (37, *) xsueafter(i)
456:        End Do
457:
458:        Open (38, File='38.xafter(subtracted).txt')
459:
460:        Do i = 1, link
461:          Write (38, *) xafter(i)
462:        End Do
463:
464:        Open (68, File='68.FlowPropagation.txt')
465:        Do i = 1, node
466:          b(i, i) = 0
```

```
467:         Write (68, *)(b(i,j), j=1, node)
468:       End Do
469:
470:       Call cpu_time(v)
471:       Write (11, *) 'Calculation time 2 (second)', v
472:  !******SUBROUTINE************************************************************
473:     Contains
474:       Subroutine dijikstra(p, t)
475:         Real, Intent (In) :: t(link)
476:         Real *8, Intent (Out) :: p(node, node)
477:         Real *8 :: c(node), cmin(node), fin_io(node)
478:
479:         Do j = 1, node
480:           io = j !io is the source node number
481:           c(:) = 10000000.0 !set partial path cost • ‡
482:           c(io) = 0.0 !set the cost of the source node to 0
483:           cmin(:) = 10000000.0
484:           fin_io(:) = 0 !array for storing the nodes for which calculation has been completed
485:           Do While (minval(cmin,1)<50000000)
486:             Do iii = headnode(io-1) + 1, headnode(io)
487:               i = ls(io, anode(iii))
488:               If (c(io)+t(i)<=c(e(i))) Then
489:                 c(e(i)) = c(io) + t(i) !update with the next link cost with the node
490:                 cmin(e(i)) = c(io) + t(i)
491:               End If
492:             End Do
493:             ncj = minloc(cmin, 1) !identify the location of cmin in the array
494:             cmin(ncj) = 50000000.0 !ineffectiveness of extracted cmin
495:  !**** calculation end judgment (end when all cmin are disabled) ***********
496:             If (minval(cmin,1)==50000000) Then
497:               Go To 100
498:             End If
499:  !******!move to the next node******************
500:             io = ncj
501:             If (fin_io(ncj)==1) Then !when calculation is completed, move to another node
502:               inot = minloc(fin_io, 1)
503:               io = inot
504:             End If
505:             fin_io(io) = 1
506:           End Do
507:  100      Continue
508:  !******record shortest path results***********************************
509:           p(j, :) = c(:)
510:         End Do
511:
512:       End Subroutine
513:  !******STOP*****************************************************************
514:  !    Calculate xij with DFS
515:       Recursive Subroutine calxij(dfs)
516:         Integer dfs, iii
517:
518:         If (j==des(ioi)) Then
```

```
519:          crou = 0.0
520:          Do i = 1, k
521:            crou = crou + tt(mark(i))
522:          End Do
523:          vn = (1.0D0+(kiu-1.0D0)*(-theta*crou))**(1.0D0/(kiu-1.0D0))
524:          dnm = dnm + vn
525:          Do i = 1, k
526:            num(mark(i)) = num(mark(i)) + vn
527:          End Do
528:        Else
529:          Do iii = headnode(j-1) + 1, headnode(j)
530:            If (omega(ioi,alink(iii))/=0) Then
531:              j = e(alink(iii))
532:              k = k + 1
533:              mark(k) = alink(iii)
534:              Call calxij(dfs+1)
535:              mark(k) = 0
536:              k = k - 1
537:            End If
538:          End Do
539:        End If
540:      End Subroutine
541:
542:  !c    Calculate xijrs and b1ijghrs with DFS
543:      Recursive Subroutine calxij2(dfs)
544:        Integer dfs, iii, mm
545:
546:        If (j==des(ioi)) Then
547:          crou = 0.0
548:          Do i = 1, k
549:            crou = crou + tt(mark(i))
550:          End Do
551:          vn = (1.0D0+(kiu-1.0D0)*(-theta*crou))**(1.0D0/(kiu-1.0D0))
552:          Do i = 1, k
553:            num(mark(i)) = num(mark(i)) + vn
554:            b2gh(mark(i)) = b2gh(mark(i)) + (1.0D0+(kiu-1.0D0)*(-theta*crou)) &
555:              **((2.0D0-kiu)/(kiu-1.0D0))
556:            Do ii = 1, k
557:              b1ijgh(mark(i), mark(ii)) = b1ijgh(mark(i), mark(ii)) + &
558:                (1.0D0+(kiu-1.0D0)*(-theta*crou))**((2.0D0-kiu)/(kiu-1.0D0))
559:            End Do
560:          End Do
561:
562:        Else
563:          Do iii = headnode(j-1) + 1, headnode(j)
564:            If (omega(ioi,alink(iii))/=0) Then
565:              j = e(alink(iii))
566:              k = k + 1
567:              mark(k) = alink(iii)
568:              Call calxij2(dfs+1)
569:              mm = j
570:              If (mm/=des(ioi) .And. num(mark(k))==0) Then
```

```
571:                omega(ioi, mark(k)) = 0
572:              End If
573:              mark(k) = 0
574:              k = k - 1
575:            End If
576:          End Do
577:        End If
578:      End Subroutine
579:  !   Calculate dijgh with DFS
580:      Recursive Subroutine caldijgh(dfs)
581:        Integer dfs, iii
582:
583:        If (j==des(ioi)) Then
584:          crou = 0.0
585:          Do i = 1, k
586:            crou = crou + tt(mark(i))
587:          End Do
588:          vn = (1.0D0+(kiu-1.0D0)*(-theta*crou))**(1.0D0/(kiu-1.0D0))
589:          dnm = dnm + vn
590:          Do i = 1, k - 1
591:            Do ii = i + 1, k
592:              dijgh(mark(i), mark(ii)) = dijgh(mark(i), mark(ii)) + vn
593:            End Do
594:          End Do
595:        Else
596:          Do iii = headnode(j-1) + 1, headnode(j)
597:            If (omega(ioi,alink(iii))/=0) Then
598:              j = e(alink(iii))
599:              k = k + 1
600:              mark(k) = alink(iii)
601:              Call caldijgh(dfs+1)
602:              mark(k) = 0
603:              k = k - 1
604:            End If
605:          End Do
606:        End If
607:      End Subroutine
608:  !******FINISH SUBROUTINE**********************************
609:     End Program
```