Graph Planarization Problem Optimization Based on Triple-Valued Gravitational Search Algorithm

IEEJ TRANSACTIONS ON ELECTRICAL AND ELECTRONIC ENGINEERING
IEEJ Trans 2014; **9**: 39–48
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI:10.1002/tee.21934

Paper

Graph Planarization Problem Optimization Based on Triple-Valued Gravitational Search Algorithm

Shangce Gao*, Member
Yuki Todo**[a], Non-member
Tao Gong*, Non-member
Gang Yang***, Non-member
Zheng Tang****, Non-member

This article presents a triple-valued gravitational search algorithm (TGSA) to tackle the graph planarization problem (GPP). GPP is one of the most important tasks in graph theory, and has proved to be an NP-hard problem. To solve it, TGSA uses a triple-valued encoding scheme and models the search space into a triangular hypercube quantitatively based on the well-known single-row routing representation method. The agents in TGSA, whose interactions are driven by the gravity law, move toward the global optimal position gradually. The position updating rule for each agent is based on two indices: one is a *velocity index* which is a function of the current velocity of the agent, and the other is a *population index* based on the cumulative information in the whole population. To verify the performance of the algorithm, 21 benchmark instances are tested. Experimental results indicate that TGSA can solve the GPP by finding its maximum planar subgraph and embedding the resulting edges into a plane simultaneously. Compared with traditional algorithms, a novelty of TGSA is that it can find multiple optimal solutions for the GPP. Comparative results also demonstrate that TGSA outperforms the traditional meta-heuristics in terms of the solution qualities within reasonable computational times. © 2013 Institute of Electrical Engineers of Japan. Published by John Wiley & Sons, Inc.

Keywords: gravitational search algorithm, triple-valued encoding, graph planarization

Received 12 July 2012; Revised 6 October 2012

1. Introduction

Graph planarization problem (GPP) is a widely researched optimization problem. It has not only theoretical importance related with topological problems but also practical applications in numerous areas, such as circuit board layout, facility layout, automatic graph drawing, and VLSI circuit routing [1]. A graph is said to be planar or embeddable in a plane if it can be drawn on the plane in such a way that no two of its edges intersect except at a common endpoint. Given an m-vertex n-edge graph $G = (V, E)$ with vertex set V and edge set E, the objective of finding the maximum planar subgraph (MPS) is to find a minimum cardinality subset of edges $F \subseteq E$ such that the graph $G' = (V, E \backslash F)$ resulting from the removal of the edges F from G is planar. As a graph's linear embedding problem, the GPP generally needs to fulfill two tasks: MPS acquisition and plane embedding. MPS acquisition is to verify if the input graph is planar, and further to extract the edges of its MPS if it is a nonplanar graph. Plane embedding is to embed the MPS of the input graph into a plane and perform the routing of the vertices and edges in the plane without any cross connections or cross edges. The GPP is very difficult to solve, and it belongs to NP-hard problems [2].

Several studies have been presented to propose algorithms for the GPP using different techniques. Initial works focused on developing sophisticated heuristic algorithms based on planarity testing, aiming to identify the edges of the MPS with low computational complexity [3–7]. Nevertheless, these algorithms were devoted only to the graph planarity test task (i.e. MPS acquisition). In other words, they failed to fulfill the plane embedding task. The reason was that there were no embedding mechanisms in these algorithms. To solve the GPP with the MPS acquiring and embedding tasks simultaneously, some meta-heuristics were proposed. In Ref. [8], the first example illustration was provided, and the proposed neural network was then further developed in Refs [9,10] with the purpose of improving its searching performance. In addition to neural networks, genetic algorithms [11,12] and artificial immune systems [13] also exhibited super applicability on the GPP. Although these meta-heuristics are capable of solving the GPP, their performance is limited with the size of the input graphs; thus seeking other meta-heuristics with better optimization abilities is still an open problem.

In this paper, a triple-valued gravitational search algorithm (TGSA) is proposed to solve the GPP. TGSA is an extended version of the gravitational search algorithm (GSA) [14,15]. The main characteristic of TGSA is its triple-valued encoding scheme, which makes the algorithm more suitable to manipulate the GPP. To realize this, the representation of GPP adopts the single-row routing method. The vertices of input graphs are first placed on a straight line. Then the edges are divided into three subsets: the upper edges over the line (U), the lower ones (L), and the directly eliminated ones (E), which naturally form a triangular

[a] Correspondence to: Dr. Yuki Todo. E-mail: yktodo@se.kanazawa-u.ac.jp

* College of Information Science and Technology, Donghua University, No. 2999 North Renmin Road, Shanghai 201620, China

** Brain-Like Information Processing Laboratory, Faculty of Electrical and Computer Engineering, Kanazawa University, Kakuma-machi, Kanazawa 920-1192, Japan

*** School of Information, Renmin University of China, No. 59 Zhongguancun Street, Beijing 100872, China

**** Graduate School of Innovation Life Science, University of Toyama, Gofuku 3190, Toyama-shi 930-8555, Japan

hypercube search space for the algorithm. Thereafter, the GPP is transferred into an optimization problem, where the objective of the implemented TGSA is to optimize the combinatorial problem of the edges' placement. It is important to highlight that TGSA is able to produce several optimal planar subgraphs, comparable to those obtained by other approaches. This is the crucial feature of population-based algorithms in general, and of the algorithm, TGSA, used in this research work in particular. Eventually, in order to show the accuracy of the proposed TGSA, a fair comparison is made with other six meta-heuristics. Experimental results verify the effectiveness of the proposed algorithm.

This paper is structured as follows. The following section makes a small literature review on the algorithms used when solving the GPP. The review also indicates that gravitational-search-based algorithms have never been used for such a study. Section 3 gives a general description of the GSA. Section 4 illustrates the problem representation, the designs, and the application of TGSA on the GPP. Simulation results are summarized and discussed in Section 5. Finally, some general remarks are given to conclude this paper.

2. Related Works

For finding the MPS of a given graph, numerous heuristics have been proposed. A widely used standard process is to start with a spanning tree of the input graph $G = (V, E)$, and to iteratively try to add the remaining edges one by one. In every step, a planarity testing algorithm is called for the obtained graph. If the addition of an edge would lead to a nonplanar graph, then the edge is disregarded; otherwise, the edge is added permanently to the planar graph obtained so far. Within this algorithmic framework, Jayakumar *et al*. [4] proposed a near-maximal planarity testing algorithm with computational complexity $O(|E|^2)$ based on the PQ-tree technique [3]. Kant [5] presented a corrected and more generalized version of Jayakumar's algorithm. Cai *et al*. [16] and Battista and Tamassia [17] proposed efficient algorithms with the same complexity bound of $O(|V|\log|E|)$, which were derived from the Hopcroft–Tarjan planarity testing algorithm [18] and the incremental planarity testing algorithm [17], respectively. Furthermore, linear-time $O(|V|+|E|)$ algorithms can be considered as in Refs [7],[19]. Obviously, the heuristics were developed for the purpose of reducing computational times. In addition, Poranen [20] proposed a hybrid simulated annealing method based on the planarity test [21] for MPS, claiming that the hybrid algorithm outperformed its earlier heuristics in terms of the solution quality and computational times. However, its deficiency was also distinct because it performed the planarity test numerous times during its execution. If an implementation for the planarity test was not available, the algorithm could not be used to solve the problem.

In general, the algorithms proposed for MPS cannot be directly adopted for the GPP since finding MPS is just one of its objectives, but their basic functions can motivate (or be incorporated into) the design of the GPP solving algorithms. In the literature, some meta-heuristics were proposed to deal with the MPS finding task and plane embedding task simultaneously. Among these algorithms, neural network learning methods, two-phase graph planarization algorithms (TGPAs), genetic algorithms, and artificial immune algorithms exhibited promising performance.

Using the neural network techniques, Takefuji and Lee presented a parallel planarization algorithm (TLNN) for generating a near-maximal planar subgraph within $O(1)$ time [8],[22], and claimed superior performance to previously published algorithms. An improved Hopfield network learning algorithm (WNN) with a gradient ascent technique for alleviating the inherent drawbacks (typically the local optima problem) of Takefuji and Lee's algorithm was proposed by Wang *et al*. [9]. Zhang and Qin [10] combined the Hopfield network with the simulated annealing strategy (ZNN), aiming to facilitate the flexibility of the algorithm and further to achieve a better performance.

In addition, Goldschmidt and Takvorian [23] presented a TGPA, based on which Resende and Ribeiro [24] proposed a greedy randomized adaptive search procedure (GRASP) by incorporating more randomness into it. GRASP had two phases: solution construction phase and solution improvement phase. GRASP often mainly focused on the randomized generation of high-quality starting solutions by very refined construction phases and sophisticated management of the solutions, while the subsequent solution improvement phase was usually performed by a rather simple local search. The first phase in GRASP was an enhancement version of the TGPA by randomly choosing one of the best candidates in the restricted candidate list. The second phase in GRASP aimed at minimizing the number of edges that needed to be removed to eliminate all edge crossings with respect to the first-phase sequence, which was realized by implementing a simple local search within a slightly restricted neighborhood. Simulation results indicated that GRASP could obtain better or competitive results than TGPA and branch-and-cut algorithms [25,26] for graphs having up to 300 vertices or 1500 edges.

Besides, the applicability of genetic algorithms to solving the GPP had been verified by Comellas in Ref. [11]. An effective genetic algorithm (EGA) performing crossover and mutation operators conditionally instead of probability could be also referred to as in Ref. [12]. Most recently, the authors [13] proposed a multilayered artificial immune system (MAIS) for the GPP, which took advantages of the technologies of search space minimization and feature extraction, exhibiting excellent performance on all tested benchmark problems. It is to be noted that this section does not aim to provide a comprehensive literature review, but intends to gives some insights into the ideas of the proposed algorithms, and further verify that gravitational search-based algorithms have never been studied for solving the GPP.

3. Gravitational Search Algorithms

Meta-heuristic algorithms mimic biological or physical processes. One of the newest meta-heuristic algorithms that has been inspired by the physical laws is the GSA. It was originally presented by Rashedi *et al*. [14] for optimizing continuous nonlinear functions, and then widely applied to many engineering problems [27–29]. The algorithm is based on the Newtonian gravity: every particle in the universe attracts every other particle with a force that is directly proportional to the product of their masses and inversely proportional to the square of the distance between them. Figure 1 depicts the conceptual graph of gravity law used in GSA.

In GSA, agents are considered as solutions and their performance is measured by their masses. Each object in GSA has two

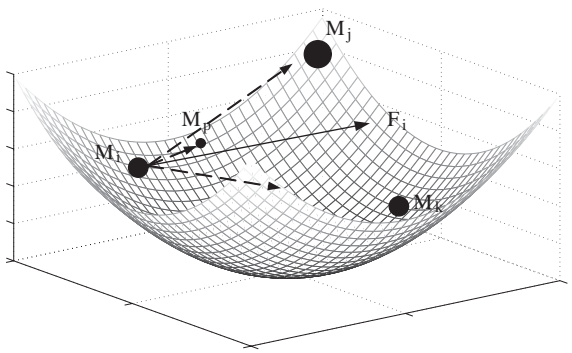


Fig. 1. Gravity law used in GSA: every agent accelerates toward the resulting force that acts on it from other agents

specifications: position and mass. The position of the agent corresponds to a solution of the problem on hand, and its mass is determined using a fitness function. To describe the continuous GSA [14] in formulations, consider a system with S agents in which the position of the ith agent is defined as follows:

$$X_i = (x_i^1, \ldots, x_i^d, \ldots, x_i^n), \quad i = 1, 2, \ldots, S$$

where x_i^d is the position of the ith agent in the dth dimension and n is the dimension of the search space.

The mass of each agent is calculated after computing the current population's fitness as follows:

$$q_i(t) = \frac{fit_i(t) - \text{worst}(t)}{\text{best}(t) - \text{worst}(t)} \tag{1}$$

$$M_i(t) = \frac{q_i(t)}{\sum_{j=1}^{s} q_j(t)} \tag{2}$$

where $M_i(t)$ and $fit_i(t)$ represent the mass and the fitness value of the agent i at the iteration t, respectively. Without loss of generality, best(t) and worst(t) are defined for a minimization problem as follows:

$$best(t) = \min_{j \in \{1,\ldots,S\}} fit_j(t) \tag{3}$$

$$worst(t) = \max_{j \in \{1,\ldots,S\}} fit_j(t) \tag{4}$$

To compute the acceleration of an agent, total forces from a set of heavier agents that apply on an agent should be considered based on the gravity law (5), which is followed by the calculation of agent acceleration using the law of motion (6). Afterward, the velocity and position of an agent are updated according to (7) and (8):

$$F_i^d(t) = \sum_{j \in K\ best, j \neq i} r_j G(t) \frac{M_j(t) M_i(t)}{R_{ij}(t) + \varepsilon} (x_j^d(t) - x_i^d(t)) \tag{5}$$

$$a_i^d(t) = \frac{F_i^d(t)}{M_i(t)} \tag{6}$$

$$v_i^d(t+1) = r_i \times v_i^d(t) + a_i^d(t) \tag{7}$$

$$x_i^d(t+1) = x_i^d(t) + v_i^d(t+1) \tag{8}$$

where r_i and r_j are two uniformly distributed random numbers in the interval $[0, 1]$, ε is a small value, $R_{ij}(t)$ is the Euclidean distance between two agents i and j, defined as $||X_i(t), X_j(t)||_2$, and *Kbest* is the set of first K agents with the best fitness value and biggest mass, which is a function of time as shown in (9), initialized to K_0 at the beginning and decreasing with time. The gravitational constant G will take an initial value G_0, and it will be reduced with time (10) [14]:

$$K = \lfloor (\beta + (1 - \frac{t}{T_{max}})(1 - \beta)) K_0 \rfloor \tag{9}$$

$$G(t) = G(G_0, t) = \alpha G(t-1) \tag{10}$$

The main components of binary GSA (BGSA) [15] are the same as those of continuous GSA (CGSA). Compared to CGSA, the main novelty of BGSA is its position updating rule, where only the switching between 0s and 1s takes place, and thus makes the search space form into a binary hypercube. Correspondingly, (7) and (8) are changed into the following equations, respectively:

$$\begin{aligned} v_i^d(t+1) = w(t) v_i^d(t) &+ 2r_1(x_i^p - x_i^d(t)) \\ &+ 2r_2(x^g - x_i^d(t)) \end{aligned} \tag{11}$$

$$x_i^d(t+1) = \begin{cases} 1 - x_i^d(t) & \text{if } r < \frac{1}{1+e^{-v_i^d(t+1)}} \\ x_i^d(t) & \text{otherwise} \end{cases} \tag{12}$$

where r_1, r_2, and r are three uniform random variables in the range $[0, 1]$, w is the inertia weight, and x_i^p and x^g represent the best previous position of the ith agent and the best previous position among all the agents in the population, respectively.

To sum up, GSA (either continuous or binary variant) is a memory-less algorithm and the agent direction is calculated based on the overall force obtained by other agents, which is different from other population-based algorithms, such as PSO, ACO[vnsn1], GA, etc. In GSA, the force is directly proportional to the fitness value while inversely proportional to the distance between solutions, so that heavy masses have large effective attraction radii and hence great intensities of attraction, revealing that the agents tend to move toward the best agent, which is the main characteristic of the algorithm. Previous works [14,15,29] on solving many optimization problems have demonstrated the superiority of the algorithm. However, these mentioned GSAs cannot be directly applied on the GPP, which motivates us to propose a novel encoding scheme into GSA, and further evaluate its performance on solving the GPP.

4. TGSA to Solve the GPP

4.1. Problem definition The way of representing the GPP is described. In this paper, we use the single-row routing representation that was originally proposed in Takefuji and Lee's algorithm [8] and widely adopted by many other studies [12],[13],[23],[24]. According to this representation method, an arbitrary sequence of all the vertices in the given graph is placed along a line, first. Then all the edges are determined to belong to any of the two sets of edges that may be represented without crossings above and below that line, respectively. Consider a simple undirected graph composed of four vertices and six edges as shown in Fig. 2(a). The graph is planar as long as two edges,

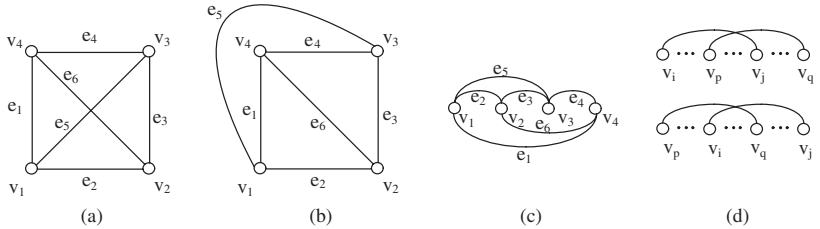


Fig. 2. Single-row routing representation used in GPP. (a) A graph with four vertices and six edges. (b) A planar graph. (c) A possible planar graph. (d) Violation conditions

$e_5 = (v_1, v_3)$ and $e_6 = (v_2, v_4)$, do not cross each other. Figure 2(b) shows a planar graph. Following the single-row routing representation method, the vertices in the graph are placed on a line and then the connection is established by either an upper edge or a lower edge. Figure 2(c) shows a possible planar graph based on the single-row routing representation. The two-edge-crossing violation condition can be easily determined from the single-row representation used. The existence of a crossing between two upper edges (v_i, v_j) and (v_p, v_q) (or two lower edges) is determined by the following conditions as shown in Fig. 2(d):

$$\text{if } v_i < v_p < v_j < v_q \text{ or } v_p < v_i < v_q < v_j \tag{13}$$

Consequently, under this representation, each edge has three possible states according to whether, in accordance with a determined vertex sequence, the edge is a lower edge, it is not considered, or it is an upper edge. The objectives of solving the GPP is actually to make the number of considered edges in the resultant planar subgraph as large as possible, and further embed these edges into the plane with respect to the established vertex sequence.

4.2. Description of TGSA In this paper, a novel approach based on the TGSA to solve the GPP is presented. Since there are three possibilities of placing the edges over the line, that is, to be the upper (U) edges, the lower (L) edges, and the eliminated (E) ones, the encoding scheme is naturally designed from the triple-valued set $\{U, L, E\}$. The agents in TGSA are defined as the edges sequences, whose values are taken from the triple-valued set. Formally, for a given m-vertex n-edge graph $G = (V, E)$, the position X_i of the agent i is expressed as:

$$X_i = (e_i^1, e_i^2, ..., e_i^n),\ \forall e_i^j \in \{U, L, E\}, j = 1, 2, \ldots, n$$

Here, it is worth emphasizing that the randomly generated position X_i might become an invalid solution for the GPP, because the resulting placement of edges is likely to satisfy the violation conditions in (13). Thus, a repair operator to ensure the position always corresponding to valid solutions is necessary. To realize this, a very simple repair operator is proposed in this paper as shown in Algorithm 1.

Algorithm 1 (Repair Operator used in TGSA):
Input: an arbitrary position X_i of the agent i, and an empty graph with an established vertex sequence.
01: **do:** randomly generated a sequence $(s_1, s_2, \ldots, s_n)$ from the set $\{1, 2, \ldots, n\}$
02: **for** $j = s_1$ to s_n
03: embed the edges one by one according to the sequence $(e_i^{s_1}, e_i^{s_2}, \ldots, e_n^{s_n})$ into the graph
04: if the current selected edge $e_i^{s_j}$ does not fulfill the violation conditions with respect to the graph, then embed it into the graph as a new edge ($e_i^{s_j} = U$ or L)
05: otherwise set $e_i^{s_j} = E$
06: **end-for**
07: **while:** the termination condition is satisfied
Output: the best repaired position X_i of the agent, and a resultant planar subgraph.

Obviously, all agents operated by the repair process will generate valid solutions for the GPP. In this paper, the termination condition in the repair operator is when the number of iteration reaches T_R. The computational complexity of the repair operator is $O(nT_R)$.

The fitness for position X_i of agent i is counted as the number of edges whose values equal U or L. The mass of each agent is calculated according to (1)–(4). To compute the force during agents using (5), the distance between agents should be defined in advance. Unlike binary or continuous variants of GSA, where Hamming or Euclidean distance metrics are typically used, the distance metric $|.|_T$ used in TGSA is defined according to the following equations:

$$\begin{aligned} |U - L| &= |U - E| = |L - E| = 1 \\ |U - U| &= |L - L| = |E - E| = 0 \end{aligned} \tag{14}$$

$$R_{ij}(t) = |X_i(t), X_j(t)|_T = \sum_{d=1}^{n} |x_i^d - x_j^d| \tag{15}$$

By doing so, the triangular hypercube forming the search space can be depicted quantitatively. The distance between each pair of basic elements in the triple-valued set $\{U, L, E\}$ has equal values. For instance, the solution shown in Fig. 2(c) corresponds to a position $X_i = (L, U, U, U, U, L)$, and its fitness is 6. The distance between X_i and $X_i = (L, U, U, U, E, L)$, $X_k = (L, U, U, U, E, U)$ is 1, 2, respectively.

The total forces from the set of first K agents with the heaviest mass that apply on an agent can be computed based on (5). The acceleration and velocity of an agent are calculated in the same way using (6) and (7), respectively.

As the positions of agents are restricted within the triangular hypercube discretely, (8) is no longer used because the resulting new position might be updated out of the hypercube. Instead, a proper probability function is defined to control the movement of agents in (16), where r_1 and r_2 are two uniform random variables distributed on the interval $[0, 1]$. $|x_i^d(t)|_U$ denotes the number of upper ones of the dth edge in the current population, while $|x_i^d(t)|_L$ and $|x_i^d(t)|_E$ denote the number of lower and eliminated ones, respectively.

$$x_i^d(t+1) = \begin{cases} U & \text{if } r_1 < tanh(v_i^d(t+1)) \text{ and} \\ & \quad r_2 < \dfrac{|x_i^d(t)|_U}{|x_i^d(t)|_U + |x_i^d(t)|_L + |x_i^d(t)|_E} \\ L & \text{if } r_1 < tanh(v_i^d(t+1)) \text{ and} \\ & \quad r_2 < \dfrac{|x_i^d(t)|_U + |x_i^d(t)|_L}{|x_i^d(t)|_U + |x_i^d(t)|_L + |x_i^d(t)|_E} \\ x_i^d & \text{otherwise} \end{cases} \tag{16}$$

In (16), the updating of the position for an agent i is related to two indices. The prime index is a *velocity index*, which is a function of the current velocity, $tanh(v_i^d(t+1)) = (1 + e^{-v_i^d(t+1)})^{-1}$, and determines whether the updating progress takes place or just remains the same as with the previous position. The second index is a *population index*, which is a cumulative index of being an upper edge or a lower edge for the current selected one. The population index is calculated based on the information collection of the whole population. For instance, if more dth edges of the agent to be upper ones exist in the population, larger is probability of the current dth edge to be updated into an upper one. In addition, the update into E never happens, since it will lead to a decrease of the fitness.

4.3. Properties of TGSA To conclude, the main contribution of the triple-valued encoding scheme together with the updating rule (16) is that it enables the TGSA to possess the following characteristics:

- The triple-valued encoding scheme of using the set $\{U, L, E\}$ naturally represents the choices of edges to be placed over the established vertices line. Compared to the continuous or binary GSAs, TGSA is novel and makes the algorithm more suitable to solve the GPP.

- A large value of velocity provides a high probability of changing the position of the agent with respect to its previous position (to be U, L, or remain the same), thus facilitating the agents that have large velocities (usually lighter masses) to move, rather than to stay at the original place. On the contrary, a small value of velocity provides a small probability of changing the position.
- Realizing the fact that the force acting on a heavier agent (implying to possess a good solution) causes a smaller acceleration of the agent according to (6), the velocity of a heavier agent usually is hard to be changed by the acceleration, indicating that a heavier agent is hard to move. Conversely, a lighter agent is likely to change its position, and move toward better positions at the end.
- Controlled by the population index in (16), which is implemented using a roulette wheel selection method, it is clear that if the current agent moves, it will move to the position where most other agents stand. Hence, all agents will move toward the global optimum position, thus finding the best solution eventually.

4.4. Algorithm To solve the GPP, an algorithm based on TGSA is shown in Algorithm 2. Here it should be noted that the vertex sequence is generated based on an improved Hamiltonian cycle generation (HCG) method. For more details of the original HCG method, refer to Ref. [23]. To make the paper self-explanatory, the improve HCG method is described as follows: The first vertex in the sequence is $\pi(1) = v_\beta$, where v_β is a randomly selected vertex in the given graph G. Different vertex sequence can be generated by selecting different β. After the first k vertices of the sequence have been determined, say $\pi(1)$, $\pi(2)$, ..., $\pi(k)$, the next vertex $\pi(k+1)$ is randomly selected from the vertices adjacent to $\pi(k)$ in G having the L least adjacencies in the subgraph G_k of G induced by $V \setminus \{\pi(1), \pi(2), ..., \pi(k)\}$. If $\pi(k)$ has no neighbors in G_k, we select $\pi(k+1)$ as a vertex of the L minimum degree in G_k with homogeneous probability. Then, the vertices of G are placed on a line in accordance with the determined vertex sequence π.

With respect to the generated vertex sequences, the TGSA is used to find the optimal placements of edges. Thus, the largest MPS can be acquired and embedded into a plane based on the position of the heaviest agent and its corresponding vertex sequence.

Algorithm 2 (Solve the GPP based on TGSA):
Input: a given nonplanar graph $G = (V, E)$ with $|V| = m$, $|E| = n$
01: initialize user-specified parameters, and randomly generate S positions for the agents
02: **do:**
03: **for** each agent $i = 1$ to S
04: generate a vertices π_i for agent i using the improved HCG method
05: operate the agent i using Algorithm 1
06: evaluate the fitness of agent i, and update $G(t)$, $best(t)$, $worst(t)$ and $M_i(t)$
07: **end-for**
08: compute the total force in different directions using (5)
09: compute the acceleration and velocity for each agent using (6) and (7)
10: update all agents' positions using Eq. (16)
11: **while:** the maximum iteration number achieves T_{max}
Output: an optimal agent with the heaviest mass, and the corresponding planar subgraph $G' = (V, E')$

5. Simulation Results and Discussion

To evaluate the performance of TGSA when applied it on the GPP, a set of 21 benchmark problems described in the literature [23] were used in the experiments. The problem instances are summarized in Table I, where the information data are the name of the input graph, the number of vertices ($|V|$), the number of edges ($|E|$), the Euler upper bound ($3|V| - 6$) on the number of edges in an MPS, and the number of edges in the best known solution for the GPP published earlier. The algorithm is coded using ANSI C language under the Visual Studio 2005 platform running on a personal PC (Intel(R) Core i3, 2.40 GHz with 2 GB RAM). In the experimental analysis, we tried to follow the suggestions given by Johnson [30].

5.1. Parameter sensitivity The user-specified parameters were first analyzed, and each combination was tested 30 times. Intuitively, the number of agents used in the population directly influences the trade-off between the search performance and the computational times. To make the subsequent comparison with other population-based algorithms [12],[13] fairly, the same size of the population was set, i.e. $S = 20$. Besides, the initial attraction scope of the gravity force was set to be the whole population, i.e. $K_0 = S$, and the associated shrink parameter β was set to be 2% as in the original work.

The maximum iteration number T_{max} determines the termination condition of the algorithm. The larger the value of T_{max}, the more the computational time the algorithm needs. In preliminary experiments, we found that TGSA converged to global or local optimal solutions for all instances within 1000 iterations. Table II records the smallest, largest, and average iteration number when TGSA found the global optimal solution or had been trapped into a local optimal solution (In this paper, the solution that cannot be improved over 100 iterations is regarded as a local optimum). From this table, we can find that, for the instances with small graph size, TGSA converges fast. Especially, for the instance G_1, TGSA can always find the global optimum using only one iteration. Thus, $T_{max} = 1000$ was sufficient to evaluate the performance of TGSA in all experiments.

Different from the declaration in Ref. [14] that the gravitational constant parameter G strongly effects the performance of the algorithm, in this study we found that TGSA was not so sensitive to this parameter. After setting the initial constant $G_0 = m^2 n^2$, a set of values as $\{0.8, 0.9, 0.95, 0.99, 0.995\}$ for the declining factor α was tested on all instances in the experiments. Similar quality results were obtained for different settings, revealing that TGSA was less sensitive to G. The reason for this seems to be the influence of population index in (16), which partly decreases the effects of the gravitational forces.

As to the maximum replication number T_R used in the repair operator, we set it as $\{1, 5, 10, 20, 50, 100\}$. Table III summarizes the best and average solution qualities and the computational times on the instances G_2 and G_{11}. From this table, we can find that $T_R = 10$ makes significant improvements of the solutions' quality than $T_R = 1$ and $T_R = 5$, while possessing competitive solutions when compared with $T_R = 20, 50, 100$. However, with the increment of the value of T_R, the computational times steadily increase. Thus, it can be concluded that the setting of $T_R = 10$ can well balance the trade-off between the solution quality and computational times.

Compared to the vertex sequence construction method (HCG) used in Ref. [23], the improved HCG method used in this paper is a generalized version of HCG. By incorporating the parameter L into HCG, more diversity of the vertex sequence can be obtained, thus enabling TGSA to search more different areas in the search space. In the experiments, we set L to be $\{1, 2, 3, 5, 10, m\}$,

Table I. Problem instances used in the experiments

Graph name	No. vertices	No. edges	Upper bound	Best known pre. (References)
G1	10	22	20[a]	20 ([8,9,23,24,12,13])
G2	45	85	82[a]	82 ([23,24,13])
G3	10	24	24[a]	24 ([23,24,13])
G4	10	25	24[a]	24 ([23,24,13])
G5	10	26	24[a]	24 ([23,24,13])
G6	10	27	24[a]	24 ([23,24,13])
G7	10	34	24[a]	24 ([23,24,13])
G8	25	69	69[a]	69 ([24,13])
G9	25	70	69[a]	69 ([24,13])
G10	25	71	69[a]	69 ([24,13])
G11	25	72	69[a]	69 ([24])
G12	25	90	69[a]	67 ([23,24])
G13	50	367	144	135 ([24])
G14	50	491	144	143 ([24,13])
G15	50	582	144	144 ([24])
G16	100	451	294	196 ([24])
G17	100	742	294	236 ([24])
G18	100	922	294	246 ([23,24])
G19	150	1064	444	311 ([24,13])
G20	300	4136	894	627 ([23])
G21	1000	9991	2994	1496 ([23])

[a]Actual known optimal size of the planar subgraph.

Table II. Number of iterations that TGSA needs when it converges to optimal (or suboptimal) solutions

	G1	G2	G5	G8	G13	G16	G19	G20	G21
Smallest	1	32	18	54	112	151	278	325	581
Largest	1	125	32	89	256	287	523	612	735
Average	1	54	26.3	77.7	181.3	235	424	448.3	642.7

Table III. The influence of the parameter T_R used in the experiments

		1	5	10	20	50	100
G_2	Best	81	82	**82**	82	82	82
	Average	79.5	81.7	**81.8**	81.8	81.8	81.8
	Time	9.8	10	**10.4**	10.9	12.1	13.9
G_{11}	Best	68	68	**69**	69	69	69
	Average	66.7	67.5	**68.2**	68.2	68.3	68.3
	Time	7.9	8.1	**8.4**	8.9	10.7	12

respectively, where m is the number of vertices of the input graph. Note that the original HCG method was a specific case of the improved one when $L = 1$. The case of $L = m$ indicated that the constraints of edges' adjacencies were not considered any longer. The experimental results are shown in Table IV, where the data recorded are the best and average qualities of final solutions, the computational times, and the average number of different vertex sequences per iteration obtained by the algorithm. From this table, we can find that, first, with the increment of the value of L, the computational times made no significant changes (less than 0.1 s) for all cases; second, the best solution qualities were acquired when $L = 3$; third, the search performance suddenly declines when $L = m$, which shows that the use of vertex's degree does facilitate HCG to generate more promising vertices sequences; and last but not least, the average number of different vertex sequences gradually becomes larger with the increment of the value of L. The larger the *Num*. in Table IV, the more the different the areas visited by the algorithm in the search space. Obviously, $L = 3$ was the best choice for the algorithm.

Table IV. The influence of the parameter L used in the experiments

		1	2	3	5	10	m
G_2	Best	82	82	**82**	82	82	79
	Ave.	80.3	81.8	**81.8**	81.8	81.6	76.5
	Time	10.4	10.4	**10.4**	10.4	10.4	10.4
	Num.	17.8	18.5	19.2	19.9	19.9	20.0
G_{11}	Best	68	69	**69**	69	69	66
	Ave.	66.8	67.9	**68.2**	68.1	68.1	65.5
	Time	8.4	8.4	**8.4**	8.4	8.4	8.4
	Num.	17.5	18.3	18.8	19.8	19.9	20.0

Table V. User-specified parameters employed in the experiments

Number of agents (i.e. population size)	$S = 20$
Initial gravitational constant	$G_0 = m^2 n^2$
Gravitation declining coefficient	$\alpha = 0.95$
Initial attraction scope	$K_0 = S$
Attraction narrowing coefficient	$\beta = 2\%$
Vertices sequence generation parameter	$L = 3$
Maximum replication number in Algorithm 1	$T_R = 10$
Maximum iteration number in Algorithm 2	$T_{\max} = 1000$

Based the above analysis, we can say that the performance of TGSA is competitive when using the user-specified parameters (for any given input graph $G = (V, E)$ with $|V| = m$ and E=n) shown in Table V.

5.2. Characteristics analysis Compared with CGSA and BGSA, TGSA has the main novelties of the encoding scheme which is based on the triple-valued set $\{U, L, E\}$, and the position updating rule shown in (16). The triple-valued encoding scheme naturally depicted the three choices (upper, lower, or eliminated) of placing edges upon the vertices line, and then quantitatively forming a triangular hypercube of search space for the algorithm. Afterward, the position updating rule matured the solutions within the search space.

Table VI. Comparative results with the variants of TGSA

	$TGSA_1$		$TGSA_2$		TGSA		
Graph	Best	Ave.	Best	Ave.	Best	Ave.	No.
G1	20	20	20	19.7	20	20	48
G2	82	80.5	80	78.2	82	81.8	17
G3	24	24	22	20.8	24	24	42
G4	24	24	21	20.7	24	24	41
G5	24	24	21	19.5	24	24	38
G6	24	23.7	20	18.4	24	24	35
G7	24	23.5	19	17.5	24	24	36
G8	69	68.8	56	54.2	69	69	45
G9	69	68.7	56	53.9	69	69	32
G10	68	66.5	56	53.5	69	69	29
G11	68	66.4	54	52.7	69	68.2	10
G12	68	67.3	51	48.3	69	68.4	12
G13	136	124.4	78	72.3	137	131.3	6
G14	143	133.2	82	72.6	143	140.6	5
G15	143	135	98	88.5	144	142.9	7
G16	205	199.3	103	97.1	205	201.7	5
G17	235	228.5	98	95.2	241	238.7	2
G18	249	239.8	121	118.2	257	248.5	2
G19	321	302.7	128	122.5	328	315.5	3
G20	754	728.2	452	449.5	762	734.2	1
G21	1878	1856.3	1378	1365.1	1903	1885.2	1

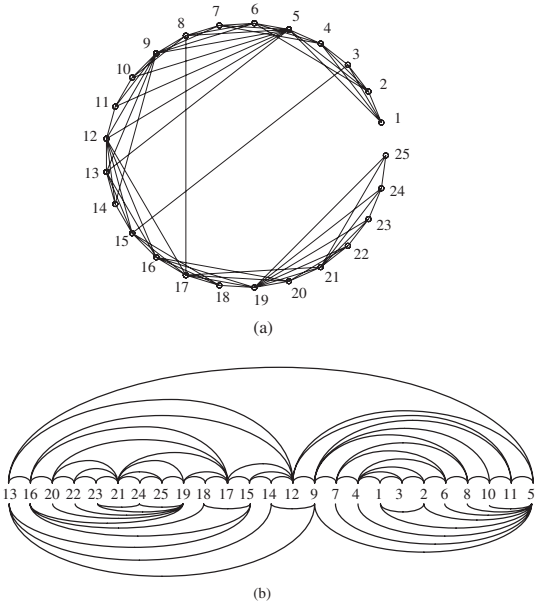


Fig. 3. (a) The original input graph, (b) a typical optimal solution obtained by TGSA for the instance G_{11}

In particular, two indices were related with the updating rule. In order to verify the effects of each factor index, two variants of TGSA were constructed (named $TGSA_1$ and $TGSA_2$). $TGSA_1$ utilizes the updating rule that only considered the velocity index (17), while $TGSA_2$ only took the population index into consideration (18).

$$x_i^d(t+1) = \begin{cases} U & \text{if } r_1 < tanh(v_i^d(t+1)) \\ L & \text{if } r_1 < tanh(v_i^d(t+1)) \\ x_i^d & \text{otherwise} \end{cases} \quad (17)$$

$$x_i^d(t+1) = \begin{cases} U & \text{if } r_2 < \frac{|x_i^d(t)|_U}{\sum} \\ L & \text{if } r_2 < \frac{|x_i^d(t)|_U + |x_i^d(t)|_L}{\sum} \\ x_i^d & \text{otherwise} \end{cases} \quad (18)$$

The comparative simulation results during TGSA, $TGSA_1$, and $TGSA_2$ are given in Table VI. For all tested instances, $TGSA_1$ performed slightly worse than TGSA in terms of the best and average qualities of solutions, indicating that the velocity index in (16) did enable the algorithm to move toward promising areas in the search space and finally to find good solutions for the GPP. However, the missing of population index in $TGSA_1$ also indicated that the population index had the effects of promoting the search performance for the algorithm. On the other hand, $TGSA_2$ performed significantly worse than TGSA, showing that only population index would result in a very fast local optimum trapping. From the above analysis regarding (16), we can conclude that both the velocity index and the population index play positive roles in determining the search directions for the algorithm.

For the convenience of describing the progress of solving the GPP using TGSA, we used G_{11} as an illustration. Figure 3(a) shows the input nonplanar graph G_{11} with 25 vertices and 72 edges. After generating a vertices sequence of $\{13, 16, 20, ..., 11, 5\}$ by the improved HCG method, TGSA optimized the values of the combination of all edges. Finally, 47 edges were set to be the values of U, 22 edges to be L, and 3 to be E. Figure 3(b) depicts its corresponding planar subgraph based on the single-row routing representation method. Clearly, TGSA can solve the GPP with two objectives simultaneously: not only the acquisition of an MPS, but also the embeddedness of these edges into a planar subgraph.

In addition, due to the stochastic nature of the algorithm and of the populations evolved during the convergence process, TGSA can produce more than one planar subgraph for the GPP. To show this capacity of TGSA, we give an example for the instance $G3$, as shown in Fig. 4. Figure 4(a) depicts the input graph $G3$ with 10 vertices and 24 edges, while Fig. 4(b)–(h) illustrate seven different global optimal solutions of G_3 found by TGSA. For all tested instances, the number of different global optimal solutions in the final iteration found by the algorithm was recorded. The index "No." shown in Table VI counted the total number of such solutions over 30 runs. A major benefit of this feature is that it will help the decision maker to assess and select an engineering-relevant planar subgraph for industrial applications.

5.3. Comparison with other algorithms Finally, Table VII summarizes the comparative results of TGSA with other six meta-heuristics. It should be noted that all the compared algorithms were able to solve the GPP by fulfilling the two objectives simultaneously. Those algorithms that were only capable of finding the MPS for the GPP were not considered in this study, even though they might find better MPSs. In Table VII, the recorded results of the earlier six algorithms were taken from the original literature, while the latter results were taken on the basis of 30 replication runs of TGSA. The symbol '–' means there is no record for the corresponding instance. From this table, it can be easily found that, for all tested instances, the sizes of the acquired MPSs by TGSA were larger than those by the other meta-heuristics. Besides, TGSA can always find the global optima for a part of the input graphs (9 out of 21 instances), because the average quality of solutions are equal to the best one, thus revealing the strong robustness of TGSA.

Regarding the computational times, Fig. 5 illustrates the computational times in seconds versus the number of edges of the input instances. For all the comparative algorithms, the times increased along with the size of the input graphs grown, nearly at polynomial speed. Although TGSA cannot dominate the other algorithms in terms of the computational times, it can be concluded that TGSA is able to find better solutions for the GPP within reasonable times.

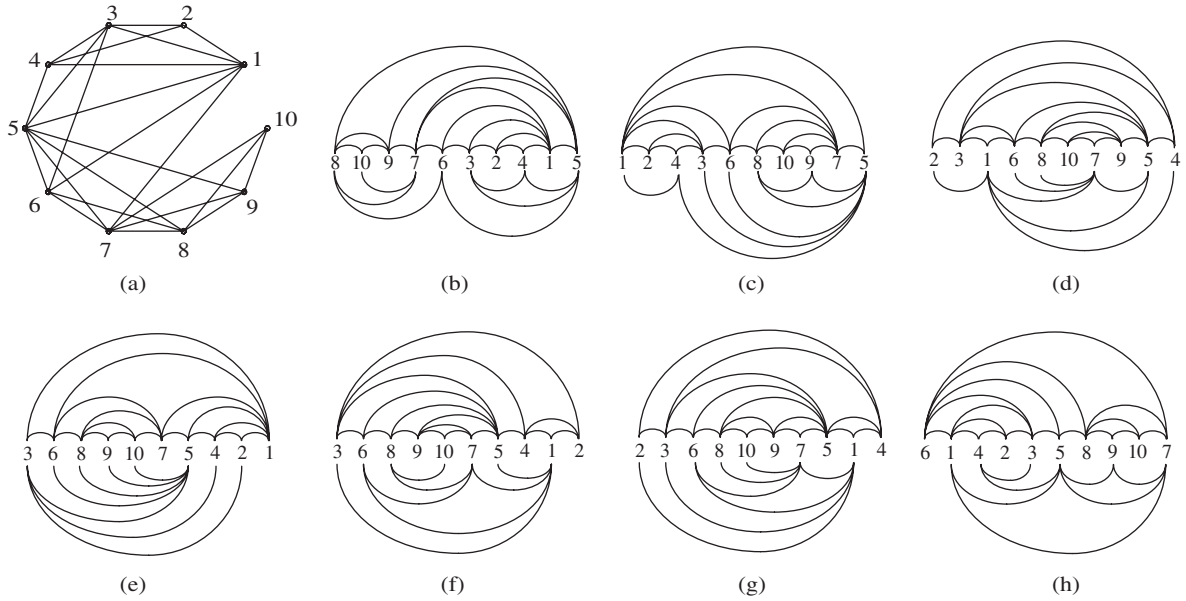


Fig. 4. Global optimum solutions for the instance G_3 based on TGSA

Table VII. Simulation results using seven algorithms

Graph	Ref. [8]	Ref. [9]	Ref. [23]	Ref. [24]	Ref. [12]	Ref. [13]	This paper	
							Best	Ave.
G1	20	20	20	20	20	20	20	20
G2	80	80	82	82	80	82	82	81.8
G3	21	22	24	24	22	24	24	24
G4	22	22	24	24	22	24	24	24
G5	22	22	24	24	22	24	24	24
G6	22	22	24	24	22	24	24	24
G7	23	23	24	24	23	24	24	24
G8	58	61	68	69	61	69	69	69
G9	59	61	69	69	61	69	69	69
G10	58	61	68	69	61	69	69	69
G11	60	61	68	69	61	68	69	68.2
G12	61	63	67	67	63	65	69	68.4
G13	70	82	129	135	84	131	137	131.3
G14	100	109	138	143	114	143	143	140.6
G15	101	115	142	144	119	142	144	142.9
G16	92	100	183	196	101	192	205	201.7
G17	116	126	215	236	127	225	241	238.7
G18	115	135	234	246	138	237	257	248.5
G19	127	138	291	311	145	311	328	315.5
G20	250	–	627	–	–	–	762	734.2
G21	322	–	1496	–	–	–	1903	1885.2

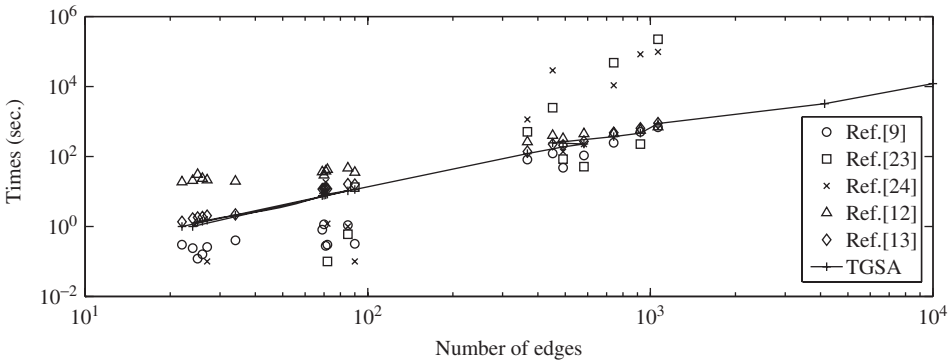


Fig. 5. Computational times of all tested algorithms

6. Conclusions

The TGSA has been successfully used to solve the GPP by finding its MPS and embedding the edges into a plane. Based on the single-row routing representation method, once the vertex sequence was determined by the improved HCG method, TGSA was utilized to optimize the placement of edges, wherein all agents interacted based on the gravity law, and the triple-valued encoding strategy of their positions naturally suited the three choices of edges' placements.

To improve the search performance of TGSA, the position updating rule was manipulated by two important indices. The *velocity index*, which is a probability function of velocities, is the prime factor controlling the search direction of an agent by indirectly using the weight of its mass and the forces from other agents. The other is the *population index*, which is based on the cumulative information collected from the whole population, directly influencing the movement direction by calculating the proportion of the selected edges in the population. By doing so, the local exploitation and global exploration of the search can be well balanced, thus enhancing the search ability of TGSA.

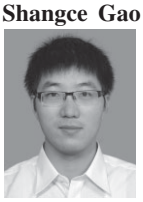
Experimental results based on 21 benchmark instances verified the effectiveness of TGSA. Compared to other traditional meta-heuristics, TGSA could always find better solutions for the GPP within reasonable computational times. In addition, due to its stochastic property and the populations evolved during the convergence process, TGSA can find multiple optimal solutions, which enables the decision maker to assess and select an engineering-relevant planar subgraph.

Acknowledgments

This work was partially supported by the National Natural Science Foundation of China under Grants 61203325 and 61003205, Genguang Project of Shanghai Educational Development Foundation (No. 12CG35), and the Programs for Young Excellent Talents in Tongji University (No. 2011KJ054) and Donghua University (No.104-07-0053012), respectively.

References

(1) Nishizeki T, Chiba N. *Planar Graphs: Theory and Algorithms.* Elsevier Science Publishers B.V.: North Holland; 1988.
(2) Liu P, Geldmacher R. On the deletion of nonplanar edges of a graph. Proceedings of the 10th Southeastern Conference on Combinatorics, Graph Theory and Computing, Boca Raton, FL, 1977; 727–738.
(3) Booth K, Lueker G. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithm. *Journal of Computer and System Sciences* 1976; **13**:335–379.
(4) Jayakumar R, Thulasiraman K, Swamy MNS. O (n2) algorithms for graph planarization. *IEEE Transactions on Computer-Aided Design* 1989; **8**(**3**):257–267.
(5) Kant G. An O (n2) maximal planarization algorithm based on pq-tree, Tech. Rep. RUU-CS-92-03, Department of Compter Science, Utrecht University, Utrecht, The Netherlands, 1992.
(6) Junger M, Leipert S, Mutzel P. A note on computing a maximal planar subgraph using PQ-trees,. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 1998; **17**(**7**):609–612.
(7) Djidjev H. A linear algorithm for the maximal planar subgraph problem. *Lecture Notes in Computer Science* 1995; **955**:369–380.
(8) Takefuji Y, Lee KC. A near-optimum parallel planarization algorithm. *Science* 1989; **245**(**4923**):1221–1223.
(9) Wang RL, Tang Z, Cao QP. An efficient parallel algorithm for planarization problem. *IEEE Transactions on Circuits and Systems I Fundamental Theory and Applications* 2002; **49**(**3**):397–401.
(10) Zhang J, Qin Q. A new neural network algorithm for planarization problems. *Science in China Series F: Information Sciences* 2008; **51**(**12**):1947–1957.
(11) Comellas F. Using genetic algorithms for planarization problems. In *Computational and Applied Mathematics*. Brezinski IC, Kulish U (eds). Elsevier Science Publishers B.V.: North Holland; 1992; 93–100.
(12) Wang RL, Okazaki K. Solving the graph planarization problem using an improved genetic algorithm. *IEICE Transactions on Fundamentals* 2006; **E89:-A**(**5**):1507–1512.
(13) Gao S, Wang RL, Tamura H, Tang Z. A multi-layered immune system for graph planarization problem. *IEICE Transactions on Information and Systems* 2009; **E92:-D**(**12**):2498–2507.
(14) Rashedi E, Nezamabadi-pour H, Saryazdi S. Gsa: a gravitational search algorithm. *Information Sciences* 2009; **179**(**13**):2232–2248.
(15) Rashedi E, Nezamabadi-pour H, Saryazdi S. Bgsa: binary gravitational search algorithm. *Natural Computing* 2010; **9**(**3**):727–745.
(16) Cai J, Han X, Tarjan RE. An O(m log n)-time algorithm for maximal planar subgraph problem. *SIAM Journal on Computing* 1993; **22**:1142–1162.
(17) Battista GD, Tamassia R. Incremental planarity testing. Proceedings of IEEE Symposium on Foundation of Computer Science, 1989; 436–441.
(18) Hopcroft J, Tarjan RE. Efficient planarity testing. *Journal of the ACM* 1974; **21**:549–568.
(19) Hsu WL. A linear time algorithm for finding maximal planar subgraphs. *Lecture Notes in Computer Science* 1995; **1004**: 352–362.
(20) Poranen T. A simulated annealing algorithm for the maximum planar subgraph problem. *International Journal of Computer Mathematics* 2001; **81**(**5**):555–568.
(21) Cimikowski R. On heuristics for determining the thickness of a graph. *Information Sciences* 1995; **85**(**1–3**):87–98.
(22) Takefuji Y, Lee KC, Cho YB. Comments on 'O (n2) algorithm for graph planarization'. *IEEE Transactions on Computer-Aided Design* 1991; **10**(**12**):1582–1583.
(23) Goldschmidt O, Takvorian A. An efficient graph planarization two-phase heuristics. *Networks* 1994; **24**:69–73.
(24) Resende MGC, Ribeiro CC. A GRASP for graph planarization. *Networks* 1997; **29**:173–189.
(25) Mutzel P. The maximum planar subgraph problem, Ph.D. thesis, Universitat zu Koln, 1994.
(26) Junger M, Mutzel P. Maximum planar subgraphs and nice embeddings: practical layout tools. *Algorithmica* 1996; **16**:33–59.
(27) Bahrololoum A, Nezamabadi-pour H, Bahrololoum H, Saeed M. A prototype classifier based on gravitational search algorithm. *Applied Soft Computing* 2012; **12**:819–825.
(28) Yin M, Hu Y, Yang F, Li X, Gu W. A novel hybrid k-harmonic means and gravitational search algorithm approach for clustering,. *Expert Systems with Applications* 2011; **38**:9319–9324.
(29) Li C. Ts fuzzy model identification with gravitational search based hyper-plane clustering algorithm. *IEEE Transactions on Fuzzy Systems* 2011; **99**:1–12.
(30) Johnson DS. A theoretician's guide to the experimental analysis of algorithms. Proceedings of the 5th and 6th DIMACS Implementation Challenges, American Mathematical Society, 2002; 215–250.



Shangce Gao received the B.S. degree from Southeast University, Nanjing, China, in 2005, and the M.S. and Ph.D. degrees in intellectual information systems and innovative life sciences from the University of Toyama, Toyama, Japan, in 2008 and 2011, respectively. From 2011 to 2012, he was an Associate Research Fellow with the Key Laboratory of Embedded System and Service Computing, Ministry of Education, Tongji University, Shanghai, China. He is currently an Associate Professor with the College of Information Sciences and Technology, Donghua University, Shanghai. His research interests include intelligent computing and information processing. Dr Gao has been a recipient of the Outstanding Academic Performance Award of IEICE Hokuriku Branch in 2008, and the Outstanding Academic Achievement Award of IPSJ Hokuriku Branch in 2011.

Yuki Todo received the B.S. degree from Zhejiang University, Zhejiang, China, the M.S. degree from Beijing University of Posts and Telecommunications, Beijing, China, and the D.E. degree from Kanazawa University, Kanazawa, Japan, in 1983, 1986, and 2005, respectively. From 1987 to 1989, she was an Assistant Professor with the Institute of Microelectronics, Shanghai Jiaotong University, Shanghai, China. From 1989 to 1990, she was a research student at Nagoya University, Nagoya, Japan. From 1990 to 2000, she was a Senior Engineer with Sanwa Newtech Inc., Miyazaki, Japan. From 2000 to 2011, she worked with Tateyama Systems Institute, Toyama, Japan. In 2012, she joined Kanazawa University, where she is now an Associate Professor with the School of Electrical and Computer Engineering. Her current research interests include multiple-valued logic, neural networks, and optimization.

Tao Gong received the M.S. and Ph.D. degrees degree from the Central South University, China, in 2003 and 2007, respectively. He is currently an Associate Professor in artificial intelligence and security with Donghua University, China, and a Visiting Scholar with Purdue University, USA, Department of Computer Sciences, and CERIAS. He is a Life Member of Sigma Xi and The Scientific Research Society. He is the General Editor-in-Chief of *Immune Computation*, the first leading journal in its field, and is on the editorial boards of many international journals. His research interests include applications of immune computation in the area of network security.

Gang Yang received the B.S. and M.S. degrees from Shandong University, China, in 2002 and 2005, respectively, and the Ph.D. degree from the University of Toyama, Japan, in 2010. In 2010, he joined Renmin University of China as a Lecturer in the School of Information. His main research interests include neural network, pattern recognition, and optimization.

Zheng Tang received the B.S. degree from Zhejiang University, Zhejiang, China, in 1982 and the M.S. and D.E. degrees from Tshinghua University, Beijing, China, in 1984 and 1988, respectively. From 1988 to 1989, he was an Instructor with the Institute of Microelectronics, Tshinhua University. From 1990 to 1999, he was an Associate Professor with the Department of Electrical and Electronic Engineering, Miyazaki University, Japan. In 2000, he joined the University of Toyama, Japan, where he is currently a Professor with the Department of Intellectual Information Systems. His current research interests include intellectual information technology, neural networks, and optimization.