Reporting all segment intersections using an arbitrary sized work space

PAPER *Special Section on Discrete Mathematics and Its Applications*

Reporting All Segment Intersections Using an Arbitrary Sized Work Space

Matsuo KONAGAYA[†a)], ***Student Member and*** **Tetsuo ASANO**[†b)], ***Member***

SUMMARY This paper presents an efficient algorithm for reporting all intersections among n given segments in the plane using work space of arbitrarily given size. More exactly, given a parameter s which is between $O(1)$ and $O(n)$ specifying the size of work space, the algorithm reports all the segment intersections in roughly $O(n^2/\sqrt{s} + K)$ time using $O(s)$ words of $O(\log n)$ bits, where K is the total number of intersecting pairs. The time complexity can be improved to $O((n^2/s)\log s + K)$ when input segments have only some number of different slopes.
key words: *computational geometry, adjustable work space algorithm, segment intersection detection and reporting, isothetic segment, read-only input model*

1. Introduction

There are increasing demands for highly functional consumer electronics such as printers, scanners, and digital cameras. To achieve this functionality they need sophisticated embedded software. One fundamental difference from software used in conventional computers is that there is little allowance of working space which can be used by the software. Programs have been developed under the assumption that sufficient memory space is available. The situation above, however, asks us to design algorithms which work in small work space. One extreme is to use only constant number of words. Algorithms using only constant number of words of $O(\log n)$ bits have been called *log-space algorithms* where n is input size. We assume that input data is stored in a read-only array. If a variable of $O(\log n)$ bits is available we can read any input data. Algorithms under this constraint have been extensively studied in complexity theory under the name of log-space algorithms. However, the constraint to the constant work space seems too severe for practical applications. It is quite reasonable to use $O(\sqrt{n})$ work space for an image of size $O(\sqrt{n}) \times O(\sqrt{n})$. We call such algorithms using $o(n)$ work space for an input set of size n as **small work space algorithms**.

More interesting is to design an algorithm which runs fast using work space which is available when it is to be executed. We call such an algorithm as an **adjustable work space algorithm** since the size of work space is adjustable at any time preserving basic property that the more work space is available the faster the algorithm is. The size of work space is measured in this paper any the number of words of $O(\log n)$ bits used to in an algorithm. Tradeoffs between time and space are also important for this class of algorithms.

One such good example is the sorting algorithm [6] by Chan and Chen. Given n input data on a read-only array, their algorithm outputs those input data in the increasing order in $O((n/s)(n + s\log s))$ time with $O(s)$ work space.

In this paper we propose several adjustable work space algorithms designed for a problem of reporting all intersections among given segments in the plane, which is one of the most fundamental problems in computational geometry.

The problem has been well studied. Given n segments in the plane, we can report all K intersections in $O(n\log n + K)$ time if we can use $O(n)$ work space. Since $\Omega(n\log n + K)$ time is required in the worst case, the algorithm given by Balaban [2] achieving $O(n\log n + K)$ time and $O(n)$ space is optimal.

This paper is organized as follows. In Section 2 we begin with a simple adjustable work space algorithm for mutually intersecting detection pairs of segments, which runs in $O((n^2/s)\log s)$ time using work space of $O(s)$ for a set of n segments stored in a read-only array. Section 3 extends the result to the problem of reporting all K intersections among n given segments using $O(s)$ work space. We present three different adjustable work space algorithms all of which run in $O((n^2/s)\log s + K)$ time for a set of n isothetic segments (e.g. each of given segments is either horizontal or vertical segment). We need some special treatment if input segments may overlap each other, that is, if their intersection (in the mathematical sense) is a line segment, not a line. We show this problem can be resolved using techniques called filtering search. We also present an adjustable work space algorithm for a general set of segments with arbitrary slopes. The algorithm runs in roughly $O(n^2/\sqrt{s} + K)$ time. Section 4 gives conclusions and some future works.

2. Segment Intersection Detection

Segment intersection detection is a problem of determining whether there is any pair of mutually intersecting segments in an input set of segments in the plane. A simple and efficient algorithm [15] is known for the problem. The algorithm sweeps the plane while visiting each endpoint of input segments in the sorted order and detects an intersection if any. It runs in $\Theta(n\log n)$ time for any set of n segments in the plane.

We design an efficient adjustable work space algorithm using $O(s)$ space for segment intersection detection a given set of segments stored in a read-only array. A variable s

†Graduate School of Information Sciences, JAIST
a) E-mail: matsu.cona@jaist.ac.jp
b) E-mail: t-asano@jaist.ac.jp
DOI: 10.1587/transfun.E0.A.1

is between $O(1)$ and $O(n)$. This algorithm becomes basis of our algorithm for reporting segment intersections. See sections 3.1 and 3.2.

The algorithm first partitions an input set S into $m = n/s$ disjoint subsets $S_1, S_2, \ldots, S_m$. Whenever s does not divide n we add extra dummy segments. So we assume that each subset has exactly s segments. We do nothing in practice except computing the size m of the partition. Since the input segments are stored in a read-only array, this partition is done in the index order, that is, S_1 contains the first s segments in the array, S_2 consists of the next s segments, and so on. Then, for each pair (S_i, S_j) with $i < j$ we perform a plane sweep to detect any intersection among given segments in the set $S_i \cup S_j$. It is done in $O(s \log s)$ time using $O(s)$ work space by a standard plane sweep algorithm. Since we have $O((n/s)^2)$ different pairs, the algorithm runs in $O((n^2/s) \log s)$ time. The algorithm is referred to as Algorithm 1, where a function BentleyOttmanPlaneSweep() is a function which implement a standard plane sweep algorithm by Bentley and Ottman [3].

Algorithm 1: Segment intersection detection.

```
Partition the set S into m = n/s disjoint subsets
S_1, S_2, ..., S_m using Index Partition.
for each pair of subsets (S_i, S_j), i, j = 1 ... m do
    apply BentleyOttmanPlaneSweep(S_i ∪ S_j).
    if any intersection is found then
        stop after reporting the intersection.
    end
end
stop after reporting "No intersection."
```

Theorem 1: Given n segments in the plane in a read-only array and a parameter value s between $O(1)$ and $O(n)$, Algorithm 1 correctly determines whether there is any intersection among input segments in $O((n^2/s) \log s)$ time using $O(s)$ work space.

More generally, we propose two different ways of partitioning a given set S of n segments. Since these methods are simple, it may apply the other problems using limited work space.

Index Partition: A given set S of n elements is partitioned into $m = n/s$ disjoint subsets $S_1, \ldots, S_m$ by the indices, that is, S_1 consists of the first s elements in the array storing S, S_2 of the next s elements, and so on. If s does not divide n, then we add extra dummy segments.

Property Partition: We are given a set S of n elements and c properties for the elements. Then, S is partitioned into $m \le n/s + c$ disjoint subsets $S_{1,1}, S_{1,2}, \ldots, S_{c,r}$ with $cr \le m$ where $S_{p,q}$ denotes the q-th subset of at most s elements of the p-th property.

The index partition is simple since only index calculation is needed. To have a property partition we scan the input array while checking properties of the elements. Thus, it takes $O(cn)$ time to enumerate all the subsets. In this paper we take slopes of segments as properties.

3. Segment Intersection Reporting

In this section we consider the segment intersection reporting problem. It is to report, given a set of segments in the plane, all intersecting pairs among them. This reporting problem is more difficult than the segment intersection detection problem because we want to design an algorithm for reporting all intersecting pairs in an output sensitive manner while the algorithm for the detection problem can stop as soon as it finds any intersecting pair. It is not so easy to design an algorithm so that it runs in an output sensitive manner. More exactly, if we denote by K the total number of intersecting pairs, the computation time should be $T(n, s) + O(K)$, where $T(n, s)$ only depends on the number n of segments and the size s of work space. The number of segment intersections could be $O(n^2)$ in the worst case. If we have $\Omega(n^2)$ intersections, then a brute-force algorithm of examining all pairs of segments suffices since it reports all the intersections in $O(n^2)$ time and it needs $\Omega(n^2)$ time to report all of them. Of course, the brute-force algorithm is not optimal unless $K = \Omega(n^2)$.

The segment intersection reporting problem has been well studied. The first algorithm by Bentley and Ottman [3] based on plane sweep runs in $O((n + K) \log n)$ time using $O(n + K)$ work space. It is not so hard to reduce the space to $O(n)$ while keeping the time complexity. The first output-sensitive algorithm for the problem was given by Mairson and Stolfi [12] under the name of red-blue intersection. In the problem we are given two sets of segments colored red or blue. Assuming there is no intersection among segments of the same color, they gave an optimal algorithm which reports all the intersections in $O(n \log n + K)$ time using $O(n)$ space. The result was strengthened by Chazelle and Edelsbrunner [8] and further by Balaban [2]. Although the algorithm by Chazelle and Edelsbrunner needs $O(n + K)$ space, Balaban's algorithm uses only $O(n)$ work space and thus it is theoretically optimal.

Very little has been studied so far when work space is limited to $o(n)$. A space-efficient algorithm is presented by Chen and Chan [10] , which runs in $O((n + K) \log^2 n)$ time using only $O(\log^2 n)$ extra work space assuming that input segments are stored in a regular read/write array and thus the array can be used as a work space. Especially, a technique referred to as an implicit data structure proposed by Munro [14] can be used. In our paper, however, such a technique cannot be used since input data is stored in a read-only array.

Algorithm 2: Segment Intersection Reporting for a set of isothetic segments.

```
/* Preprocessing stage:                              */
Partition the set S into m = n/s disjoint subsets
S_1, S_2, ..., S_m using Index Partition.
/* 1st stage: Reporting all
   intersections within each subset                  */
for each subset S_i, i = 1 ... m do
    report all segments intersecting by
    BentleyOttmanPlaneSweep(S_i).
end
/* 2nd stage: Reporting all
   intersections between two subsets                 */
for each pair (S_i, S_j) of subsets , i, j = 1 ... m do
    U ← all horizontal segments in S_i and all
    vertical segments in S_j.
    report all segments intersecting by
    BentleyOttmanPlaneSweep(U).
    U′ ← all vertical segments in S_i and all
    horizontal segments in S_j.
    report all segments intersecting by
    BentleyOttmanPlaneSweep(U′).
end
```

3.1 Isothetic Segments

We begin with a simple situation where input segments are either horizontal or vertical. For the time being we assume that no two of them overlap. More exactly, we assume that for any two segments ℓ_i and ℓ_j of the same direction (horizontal or vertical) no endpoint of ℓ_i lies on ℓ_j. Under this assumption it is rather easy to design an algorithm for reporting all segment intersections using only $O(s)$ space in addition to a read-only array storing n input segments.

We can design an efficient algorithm by slightly modifying Algorithm 1 for segment intersection reporting.

Theorem 2: Given n horizontal or vertical segments without any overlap in the plane stored in a read-only array and a parameter value s between $O(1)$ and $O(n)$, Algorithm 2 correctly reports all K intersections between input segments in $O((n^2/s)\log s + K)$ time using $O(s)$ work space.

Proof. Due to the assumption that no two segments overlap each other, no two horizontal (resp., vertical) segments intersect. Thus, after reporting all intersections within each subset, all the remaining intersections are made by two isothetic segments from different subsets. Thus, all the intersections are correctly reported. Since every intersection is reported exactly once, the algorithm runs in $O((n^2/s)\log s + K)$ time. □

3.2 Algorithm Using Property Partition

Here is an algorithm based on a different idea. In the algorithms above we have partitioned a given set of n isothetic segments into n/s subsets so that each subset has exactly s segments. Then, each subset was further decomposed into a set of horizontal segments and one of vertical segments. In the second stage we take a pair (S_i, S_j) and perform a standard plane sweep for the two sets $U_1 = H(S_i) \cup V(S_j)$ and $U_2 = V(S_i) \cup H(S_j)$. Here, $H(S_i)$ (resp. $V(S_i)$) denotes a set of all horizontal (resp. vertical) segments in S_i. Since the partition into subsets is done only by indices, it may happens that $U_1 = \emptyset$ and $U_2 = S_i \cup S_j$ or $U_1 = S_i \cup S_j$ and $U_2 = \emptyset$. It means that we may have a set of segment of so different sizes for plane sweep in the second stage.

Here is a simple way of keeping the set size. When an input set S of n isothetic segments is given, we use the property partition described before. The property we use is the slope of segment, horizontal or vertical. Using the property we partition a given set S into m_h subsets $H_1, H_2, \ldots, H_{m_h}$ and m_v subsets $V_1, V_2, \ldots, V_{m_v}$. Each H_i contains only horizontal segments and each V_j contains only vertical segments. Every $H_i, 1 \le i \le m_h$ consists of exactly s horizontal segments in the index order. Just the same for $V_1, \ldots, V_v$. Due to the definition $m_h + m_v = n/s$.

At the second stage we take two subsets H_i and V_j. In the previous algorithms the subsets were obtained just by computing indices. In this case, however, we maintain two pointers (indices), one for H_i and the other for V_j. The pointer for H_i keeps the last horizontal segment of H_i. If the last segment for H_{i-1} is ℓ_p, then the pointer starts from $p+1$ and then we examine segments $\ell_{p+1}, \ell_{p+2}, \ldots$ by incrementing the pointer until we get s horizontal segments. The function ChooseNextHorizontalSegment() scans the segment array from the current position until the next horizontal segment is found. ChooseNextVerticalSegment() is similar. all intersections is described as follows.

Theorem 3: Given n isothetic segments without any overlap in the plane stored in a read-only array and a parameter value s between $O(1)$ and $o(n)$, Algorithm 3 correctly reports all K intersections between input segments in $O((n^2/s)\log s + K)$ time using $O(s)$ work space.

Proof. We partition a set S of n segments into m_h subsets $H_1, \ldots, H_{m_h}$ and m_v subsets of $V_1, \ldots, V_{m_v}$ as above. Now it is obvious that the algorithm reports all K intersections in $O(m_h m_v s \log s + K)$ time. Since $m_h m_v \le (n/(2s))^2$, its worst running time is still $O((n^2/s)\log s + K)$. □

Algorithm 3: Segment Intersection Reporting for a set of isothetic segments using mono-color partition.

```
/* Let l_1, l_2, ..., l_n be given line
   segments.                              */
repeat
    H = ∅
    repeat
        l_i = ChooseNextHorizontalSegment()
        Insert l_i into H
    until |H| = s or all horizontal segments are
    exhausted
    V = ∅.
    repeat
        l_j = ChooseNextVerticalSegment()
        Insert l_j into V
        /* now we have two subsets H and
           V                              */
        Report all segment intersections by
        BentleyOttmanPlaneSweep(H ∪ V)
    until |V| = s or all vertical segments are
    exhausted
until H = ∅ or V = ∅
```

3.3 Algorithm Using Filtering Search

There is yet another way of achieving the running time $O((n^2/s)\log s + K)$. We use the partition by index $(S_1, S_2, \ldots, S_m)$, $m = n/s$. For each subset S_i, we take all horizontal segments in S_i and put them into a data structure so that for any query vertical segment ℓ_q all k intersections of ℓ_q with those horizontal segments in S_i can be reported in $O(k + \log s)$ time. The data structure we use is a **trapezoidal decomposition** and **filtering search** [7].

Given a set $H(S_i)$ of at most s horizontal segments, we first compute a sufficiently large rectangle enclosing all the given segments and then extend rays from each endpoint of those segments until they hit an input segment or the boundary (see Figure 1). The resulting planar subdivision into rectangles is called the trapezoidal decomposition. If we incorporate a data structure for point location and a graph representing vertical adjacency of those rectangles, then we can report intersections on an arbitrarily given query vertical segment ℓ_q by first locating one of its endpoint and then follow the adjacency graph. Unfortunately, this algorithm is not good enough to achieve our target running time. Suppose we have located the lower endpoint of ℓ_q. Then, we want to find a rectangle just above the current rectangle. If a rectangle is vertically adjacent to many rectangles then it takes time to find the rectangle intersecting the query segment. In the data structure defined by Chazelle [7] we add chords (vertical sides) so that none of top and bottom sides of rectangles is incident to more than two vertical sides. The trapezoidal decomposition shown in Figure 1 is modified using arrowed chords in Figure 2.

This problem has been extensively studied. From a theoretical point of view, Chazelle [8] presented an algorithm with $O(s \log s)$ preprocessing time, $O(s)$ space, and $O(k + \log s)$ search time. It is theoretically optimal with respect to the worst case.

In this paper we use the Chazelle's data structure outlined above. In addition we use his filtering search technique to achieve the target search time with the linear size of the data structure. We have also the other data structure by Edahiro el al. [11] in mind for practical applications.

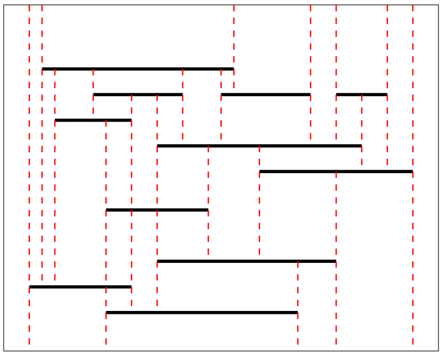

Fig. 1 Trapezoidal decomposition associated with a set of horizontal segments.

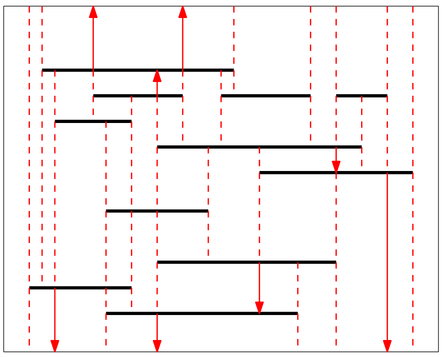

Fig. 2 Trapezoidal decomposition for filtering search in which none of top and bottom sides of rectangles is incident to more than two vertical edges.

Algorithm 4: Segment Intersection Reporting for a set of isothetic segments using trapezoidal decomposition with filtering search.

```
/* Preprocessing stage:                           */
Partition the set S into m = n/s disjoint subsets
S_1, S_2, ..., S_m using Index Partition.
for each subset S_i, i = 1...m do
    Let H(S_i) be a set of all horizontal segments in
    S_i.
    Build a data structure TD_i by trapezoidal
    decomposition and filtering search for H(S_i).
    for each segment ℓ_q in S do
        if ℓ_q is vertical then
            Report all intersections of ℓ_q with H(S_i)
            using the data structure TD_i.
        end
    end
end
```

Theorem 4: Given n segments in the plane stored in a read-only array and a parameter value s between $O(1)$ and $O(n)$, Algorithm 4 correctly reports all K intersections between input segments in $O((n^2/s)\log s + K)$ time using $O(s)$ work space.

Proof. Each subset S_i contains at most s horizontal segments. We can build the trapezoidal decomposition with filtering search in $O(s \log s)$ time using $O(s)$ work space. Then, for each vertical segment ℓ_q we can report all k intersections of ℓ_q with those in the data structure in $O(\log s)$ time, thus in total $O(K_i + n \log s)$ time, where K_i is the number of intersection reported for S_i. Hence, the total running time is given by

$$\sum_i O(K_i + n\log s) = O(K + (n^2/s)\log s).$$

□

3.4 Segments of at Most c Different Slopes without Overlap

The algorithm for horizontal and vertical segments can be extended to a more general case where given segments have at most c different slopes and we can assume that no two segments of the same slope overlap each other, where c is $o(\sqrt{n})$. In Algorithm 2 above, for each pair (S_i, S_j) of subsets we have considered two sets, one consisting of horizontal segments from S_i and vertical segments from S_j, and the other defined by replacing the roles of S_i and S_j. Then, we never see intersections between segments in one subset (those have been reported in the first stage). If there are at most c different slopes instead of just two, we can choose $O(c^2)$ combinations of distinct slopes. For each combination (α_p, α_q) of distinct slopes, we create a set of all segments of slope α_p from the subset S_i and a set of all those of slope α_q from the other subset S_j and then apply the plane sweep algorithm for the union of the two sets to report all intersections. This is also an example of the property partition of a given set. Since no two segments of the same slope intersect or overlap, we can apply the red-blue intersection reporting algorithm by Mairson and Stolfi [12] (or Balaban's optimal algorithm [2]).

Algorithm 5: Segment intersection reporting for a set of segments of at most c different slopes.

```
/* Preprocessing stage:                           */
Partition the set S into m = n/s disjoint subsets
S_1, S_2, ..., S_m using Index Partition.
/* 1st stage: Reporting all
   intersections within each subset               */
for each subset S_i, i = 1...m do
    Report all segment intersections by
    BentleyOttmanPlaneSweep(S_i)
end
/* 2nd stage: Reporting all
   intersections between two subsets              */
for each pair (S_i, S_j) of subsets i, j = 1...m do
    for each pair of slopes (α_p, α_q) do
        Let U be the set consisting of all segments
        of slope α_p in S_i and all those of slope α_q in
        S_j.
        Report all segment intersections by
        BentleyOttmanPlaneSweep(U)
    end
end
```

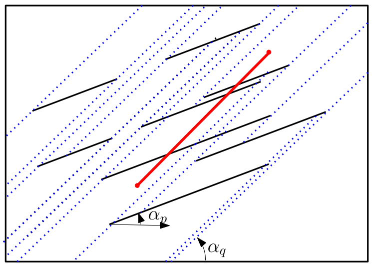

Fig. 3 Trapezoidal decomposition defined by two distinct slopes α_p and α_q, where all the segments have the slope α_p and a query segment has the slope α_q.

Theorem 5: Given n segments of at most c different slopes in the plane stored in a read-only array and a parameter value s between $O(1)$ and $o(n)$, Algorithm 6 correctly reports all K intersections between input segments in $O((c^2n^2/s)\log s + K)$ time using $O(s)$ work space.

It is not so hard to adapt Algorithm 6 so as to report segment overlaps as well. We can use the same mechanism as before. For each pair of slopes we define a similar trapezoidal decomposition as shown in Figure 3. If we rotate segments of those slopes so that they are isothetic each other then the same mechanism works.

3.5 General Case

We have efficient algorithms when given segments do not have many different slopes. Unfortunately, none of our algorithms works efficiently without the condition. So, we need a completely different idea for a general case.

A key data structure is one proposed by Agarwal and Sharir [1] based on a so-called CSW data structure [9] by Chazelle, Sharir, and Welzl. We use the following result by Agarwal and Sharir [1].

Theorem 6 (Agarwal and Sharir [1]): Given a collection S of n segments in the plane, a constant $\varepsilon > 0$, and a parameter s with $n^{1+\varepsilon} \leq s \leq n^2$, we can preprocess S into a data structure of size s, in time $O(s^{1+\varepsilon})$, so that, given any query segment ℓ_q, we can report all K segments of S intersecting ℓ_q in time $O(n^{1+\varepsilon}/\sqrt{s}+K)$, or can count the number of such segments in time $O(n^{1+\varepsilon}/\sqrt{s})$.

We assume that there is no overlap among given segments. A basic framework of our algorithm is just the same as before. After partitioning a given set S into at most $m = n/s$ disjoint subsets $S_1, \ldots, S_m$, in the first stage we report all intersections within each subset, and then in the second stage we report all intersections between segments from distinct subsets. For the second stage, we build a data structure $\mathcal{D}_i$ for each subset S_i given by Agarwal and Sharir mentioned above and then report intersections for each segment in the remaining subset.

Algorithm 6: Segment intersection reporting for a general set of segments.

```
/* Preprocessing stage:                                  */
t = s^{1/(1+ε)}.
Partition the set S into m = ⌈n/t⌉ disjoint subsets
S_1, S_2, ..., S_m using Index Partition.
/* 1st stage: Reporting intersections
   within each subset.                                   */
for each subset S_i, i = 1 ... m do
    Report all segment intersections by
    BentleyOttmanPlaneSweep(S_i)
end
/* 2nd stage: Reporting intersections
   between two subsets.                                  */
for each subset S_i, i = 1 ... m do
    Build a data structure D_i for S_i.
    for each segment ℓ_r ∈ S_{i+1} ∪ ··· ∪ S_m do
        Report all intersections of ℓ_r with those
        segments in S_i using the data structure D_i.
    end
end
```

Theorem 7: Given n segments without overlap in the plane stored in a read-only array and a parameter value s between $O(1)$ and $o(n)$, Algorithm 6 correctly reports all K intersections between input segments in $O(n^2 s^{-\frac{1-\varepsilon}{2(1+\varepsilon)}} + K)$ *time* using $O(s)$ work space for any small constant $\varepsilon > 0$.

Proof. Given n segments and a parameter value s, let t be $s^{1/(1+\varepsilon)}$ for a small constant $\varepsilon > 0$. Then, we partition the set S into $m = \lceil n/t \rceil$ disjoint subset $S_1, \ldots, S_m$, each of $t = O(s)$ segments. For each subset S_i we construct a data structure $\mathcal{D}_i$ of size $O(t^{1+\varepsilon}) = O(s)$ in $O(t^{(1+\varepsilon)^2})$ time Theorem 6 [1]. Using this data structure, we can report all intersections of a query segment ℓ_r with those segments in S_i in time $O(t^{1/2(1+\varepsilon)} + K(S_i, \ell_r))$, where $K(S_i, \ell_r)$ is the number of those intersections of ℓ_r with the segments in S_i. Then, the total running time $T(n)$ is given by

$$T(n) = \sum_{i=1}^{\lceil n/t \rceil} O(t \log t + K(S_i) + t^{(1+\varepsilon)^2} + nt^{\frac{1}{2}(1+\varepsilon)} + \sum_{\ell_r \in S_{i+1} \cup \cdots \cup S_m} K(S_i, \ell_r)),$$

where $K(S_i)$ is the number of intersections within the set S_i. Since we have

$$\sum_{i=1}^{\lceil n/t \rceil} K(S_i) + \sum_{\ell_r \in S_{i+1} \cup \cdots \cup S_m} K(S_i, \ell_r) = O(K),$$

we obtain

$$T(n) = O(\tfrac{n}{t} t \log t + \tfrac{n}{t} t^{(1+\varepsilon)^2} + \tfrac{n}{t} nt^{\frac{1}{2}(1+\varepsilon)} + K$$
$$= O(n \log t + nt^{\varepsilon^2 + 2\varepsilon} + \frac{n^2}{\sqrt{t^{(1-\varepsilon)}}} + K).$$

Replacing $t^{1+\varepsilon}$ with s, we obtain the theorem. □

4. Conclusions and Future Works

In this paper we have presented adjustable work space algorithms for detecting and reporting intersections among given segments. Those algorithms run in work space of any size between $O(1)$ and $o(n)$, assuming that n input segments are stored in a read-only array. In our conjecture, segments do not have many different slopes in reality. If the number of different slopes is bounded by $o(\sqrt{n})$ our algorithms run almost in an optimal way. However, if the assumption does not hold, our algorithm has to use a sophisticated data structure which is too impractical. So, one of the most important open problems is to devise a more practical algorithm for the general case.

Acknowledgment

This work was partially supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Scientific Research on Priority Areas and Scientific Research (B).

References

[1] P. K. Agarwal, and M. Sharir, “Applications of a New Space-Partitioning Technique,” Discrete Comput. Geom., vol.9, pp.11-38, 1993.

[2] I. J. Balaban, “An optimal algorithm for finding segments intersections,” Proc. 11th ACM Sympos. on Comput. Geom., pp.211-219,

1995.

[3] J. L. Bentley, and T.A. Ottmann, “Algorithms for Reporting and Counting Geometric Intersections, ” IEEE Trans. on Computers, C-28, 9, pp.643 - 647, 1979.

[4] M. de Berg, M. van Kreveld, R. van Oostrum, and M. Overmars, “Simple traversal of a subdivision without extra storage, ” Int. J. of Geographical Information Science, 11, 4, pp.359-373, 1997.

[5] M. Blum, R. W. Floyd, V. Pratt, R. Rivest, and R. Tarjan, “Time bounds for selection,” J. Comput. System Sci., 7, pp.448-461, 1973.

[6] T. M. Chan, and E. Y. Chen, “Multi-Pass Geometric Algorithms,” Discrete Comput. Geom., 37, 1, pp.79-102, 2007.

[7] B. M. Chazelle, “Filetering Search: a new approach to query-answering,” SIAM J. Comput., vol.15, pp.703-724, 1986.

[8] B. Chazelle, and H. Edelsbrunner, “An optimal algorithm for intersecting line segments in the plane,” J. of the ACM, Volume 39, 1. pp.1-54, 1992.

[9] B. Chazelle, M. Sharir, and E. Welzl, “Quasi-optimal upper bounds for simplex range searching and new zone theorems,” Proc. 6th ACM Sympos. on Comput. Geom., pp.23-33, 1990.

[10] E. Y. Chen, and T. M. Chan, “A space-efficient algorithm for segment intersection,” Proc. 15th Canad. Conf. Comput. Geom., pp.68-71, 2003.

[11] M. Edahira, K. Tanaka, T. Hishino, and T. Asano, “A Bucketing Algorithm for the Orthogonal Segment Intersection Search Problem and Its Practical Efficiency,” Algorithmica, 4, pp.61-76, 1989.

[12] I-I. G. Mairson, and J. Stolfi, “Reporting line segment intersections,” In R. Earnshaw, editor, Theoretical Foundations of Computer Graphics and CAD, number F40 in NATO ASI Series, pp.307-326. Springer-Verlag, 1988.

[13] David M. Mount, “Geometric Intersection,” Handbook of Discrete and Computational Geometry, CRC Press, pp.857-876,2004.

[14] J. I. Munro, “An implicit data structure supporting insertion, deletion, and search in $O(\log^2 n)$ time,” J. Comput. Sys. Sci., 33, pp.66-74, 1986.

[15] M. I. Shamos, and D. J. Hoey, “Geometric intersection problems, ” Proc. 17th IEEE Sympos. on Found. of Comp. Science, pp.208-215, 1976.

Matsuo Konagaya received B.S. degree from Shizuoka Institute of Science and Technology(SIST), Shizuoka, Japan, in 2010 and M.S. degree from Japan Advanced Institute of Science and Technology(JAIST), Ishikawa, Japan in 2012. He is currently pursuing a Ph.D. degree at JAIST. His research interests include algorithms and data structures, especially in computational geometry.

Tetsuo Asano received B.E., M.E., and Ph.D degrees from Osaka University, Japan, in 1972, 1974, and 1977, respectively. He is now a professor in School of Information Science at JAIST. His research interest includes algorithms and data structures, especially in computational geometry, combinatorial optimization, computer graphics, computer vision using geometric information.