Japan J. Indust. Appl. Math., 23 (2006), 127–138 Area ⟨3⟩

A Sweep-Line Algorithm for the Inclusion Hierarchy among Circles

Deok-Soo KIM[†1,*], Byunghoon LEE[†2] and Kokichi SUGIHARA[†3]

[†1] *Department of Industrial Engineering, Hanyang University, 17 Haengdang-dong, Seongdong-gu, Seoul 133-791, Korea*
E-mail: dskim@hanyang.ac.kr
[†2] *Voronoi Diagram Research Center, Department of Industrial Engineering, Hanyang University, 17 Haengdang-dong, Seongdong-gu, Seoul 133-791, Korea*
E-mail: bhlee@voronoi.hanyang.ac.kr
[†3] *Department of Mathematical Informatics, Graduate School of Information Science and Technology, University of Tokyo, 7-3-1, Hongo, Bunkyo-ku, Tokyo 113-8656, Japan*
E-mail: sugihara@mist.i.u-tokyo.ac.jp

Received January 24, 2005
Revised October 31, 2005

Suppose that there are a number of circles in a plane and some of them may contain several smaller circles. In this case, it is necessary to find the inclusion hierarchy among circles for the various applications such as the simulation of emulsion and diameter estimation for wire bundles. In this paper, we present a plane-sweep algorithm that can identify the inclusion hierarchy among the circles in $O(n \log n)$ time in the worst-case. Also, the proposed algorithm uses the sweep-line method and a red-black tree for the efficient computation.

Key words: nesting of circles, plane sweep, red-black, interval search, computational geometry

1. Introduction

Suppose that a set of circles $C = \{c_1, c_2, \ldots, c_n\}$ are given, where c_i is the circle centered at (x_i, y_i) with radius r_i. We assume that a circle may contain other circles while the intersections between circles on their circumferences are not allowed. Given such a set C, we want to construct the hierarchy of inclusion relationship among the circles. We propose a worst case $O(n \log n)$ time algorithm based on a sweep-line approach and a red-black tree as a primal data structure.

Such a problem occurs in many applications. An *emulsion*, for example, usually consists of two different kinds of liquids where one with a less total volume is distributed in the other in the form of small particles [12] [14]. In the emulsion, particles attract each other via a governing force which is usually represented by a function of two factors: the radii of both particles and the distance between them. Note that the force of attraction is inversely proportional to the distance.

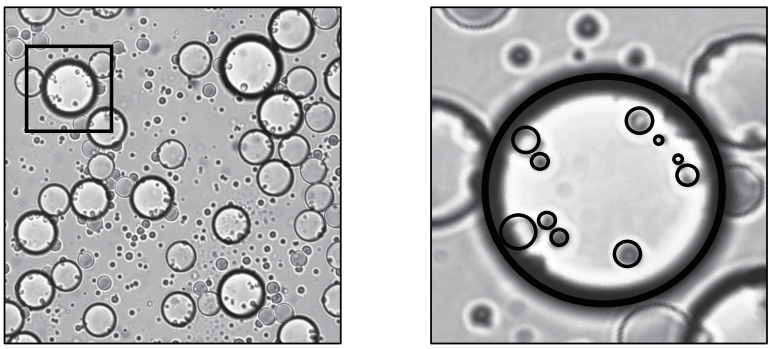
Due to the attraction, small particles in the earlier state get closer to each other to collide and eventually agglomerate to form a larger particle. Hence, a product

*Corresponding author.

in an emulsion state may change its properties and qualities over time. Therefore, the particle dynamics in an emulsion are closely related to the proximity among particles.

Fig. 1 (a) illustrates a photograph of an emulsion, and Fig. 1 (b) shows an enlargement of the particle in the square in Fig. 1 (a). As illustrated in the Fig. 1 (b), the particle contains several smaller particles inside which are also in the state of emulsion. It turns out that the agglomeration behavior of the subsystem for the smaller particles in a larger particle is identical to the higher level system among larger particles. However, both systems work independently. Note that the emulsion nested as described in the above is called a *multiple emulsion*.

In the analysis of such a multiple emulsion, the initial state of the system is usually identified by a photograph taken by a microscope and therefore the obtained data is 2D rather than 3D. Hence, in order to perform the analysis, it is necessary to extract the hierarchy of inclusion relationship among the circles and calculate inter- and intra-attractive forces among the circular particles. Fig. 1 (b) shows particles as circles in a plane and the centers and radii of these circles can be obtained by an image processing technique such as [9] [10].



(a) (b)

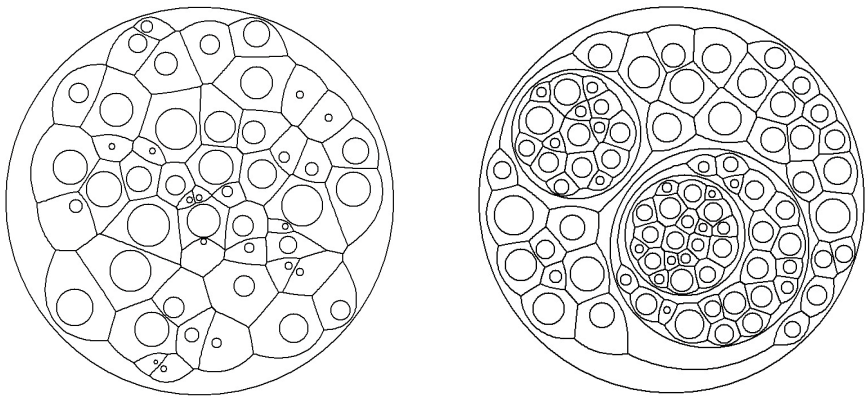
Fig. 1. An example of a multiple emulsion, (a) the top-most level, (b) the details inside a particle.

A similar phenomenon exists in other engineering disciplines such as colloids and ceramics, and their time behavior can be analyzed in a similar manner [5]. Even though such products are very common in an everyday life, there has not been a convenient computational tool to analyze the time behavior effectively and efficiently. Since physical experiments are usually expensive and time consuming, a computational simulation is getting rather important alternative for the competitiveness in the product development.

To devise an efficient simulation tool for such cases, algorithms and data structures for computing and storing the proximity among particles in an appropriate form are inevitable. One of the best alternative for such a proximity representation

is Voronoi diagram of particles. In particular, the Euclidean Voronoi diagrams of circles, often called an additively weighted Voronoi diagram, can be most appropriate for such a problem. Once a Voronoi diagram is constructed, various problems such as finding the shortest path and avoiding circular obstacles can be efficiently solved [13] [15, p. 214].

Algorithms for constructing a Euclidean Voronoi diagram of circles and a Euclidean Voronoi diagram of circles contained in a larger circle have been developed very recently [6] [7] [8]. Fig. 2 (a) shows an example of the Euclidean Voronoi diagram of circles contained in a larger circle. To construct the diagram efficiently for a multiple emulsion, it is necessary to identify the hierarchy of inclusion relationship among particles in the photograph. Then, the Euclidean Voronoi diagrams for circular particles can be constructed for each level in the hierarchy independently. Fig. 2 (b) shows a Euclidean Voronoi diagram of circles in four different levels in a hierarchy.



(a) (b)

Fig. 2. Voronoi diagrams for circles in a large circle. (a) a single depth case, (b) a multiple depth case.

2. Inclusion Hierarchy and the Sweep-Line Approach

Suppose that we are given with nine circles as shown in Fig. 3 (a), and the inclusion hierarchy among the circles is represented as a tree shown in Fig. 3 (b). Note that the circumferences of the circles do not intersect each other, but some circles are completely contained in some others. In the tree in Fig. 3 (b), c_1, c_4, and c_9 are the top-most circles in the hierarchy and placed in the node of the highest level, except the root. In this example, c_9 does not contain any other circle while c_1 contains two circles which do not contain any other circle. In the case of c_4, it contains two circles where one of them again contains two other circles.

To construct the inclusion hierarchy among circles, we propose an algorithm with $O(n \log n)$ time in the worst case based on a plane-sweep technique which is

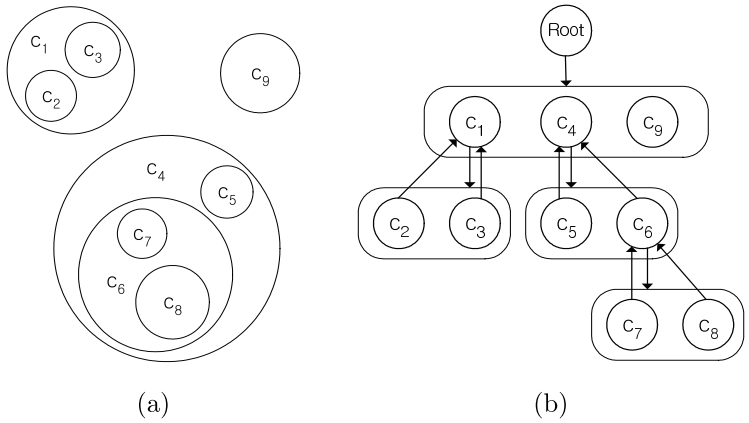


Fig. 3. Nine circles and their inclusion hierarchy. (a) nine circles, (b) the hierarchy among nine circles.

efficient by taking advantage of some orders among the related geometric entities [1, p. 22] [2] [11] [15, p. 10]. To apply the plane-sweep method, we consider a moving vertical line, called *a sweep-line*, that sweeps the plane from left to right. The proposed algorithm also employs a red-black tree as a primal data structure to facilitate efficient searches, insertions, and deletions of appropriate data items in $O(\log n)$ time.

2.1. Events in the plane

Suppose that two extreme points are assigned to a circle c_i: the leftmost and the rightmost extreme points of the circle denoted by L_i and R_i, respectively. Hence, there are $2n$ extreme points for n circles. These extreme points are sorted

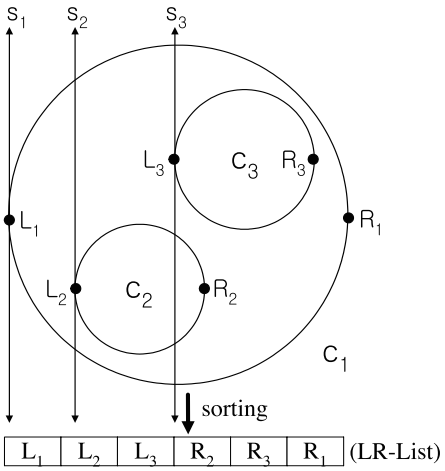


Fig. 4. LR-list and the events at left extreme points.

according to their X-coordinates. Note that their Y-coordinates are identical to those of the centers of the corresponding circle. Let *LR-list* be the sorted array of points. The sweep-line, moving from left to right, sometimes hits the extreme points, and those occasions are called *events*.

Those events corresponding to the leftmost extreme points of the circles are called *opening events* since they correspond to events opening new intervals. On the other hand, the events corresponding to the rightmost extreme points are called *closing events* since some intervals are closed. A circle is named *a generating circle* for an event if the event is defined from this circle. For example, the circle c_1 is the generating circle of the events for L_1 and R_1. Fig. 4 illustrates an example of three circles, three instances of the sweep-line with the corresponding opening events, and the related LR-list.

2.2. Interval generation

The sweep-line S at each location is divided into *intervals* depending on how the sweep-line intersects circles. Then, the intervals are maintained in a red-black tree to facilitate efficient searches, insertions, and deletions in $O(\log n)$ time in the worst case.

An interval i consists of two Y-coordinates, the lower and the upper values of the interval. We call these values *interval values*. In Fig. 5, the lower and the upper values of i_1 on the sweep-line at S_1 are the Y-coordinate value of L_1 and ∞, respectively. Similarly, the lower and the upper values of i_3 are $-\infty$ and the Y-coordinate value of L_1, respectively. In the case of i_2, both values are identical to the Y-coordinate value of L_1, and therefore i_2 degenerates to a point. We call such an interval *a zero-interval*.

Initially, we assume that the sweep-line S_0 consists of only one interval as $[i_1]$

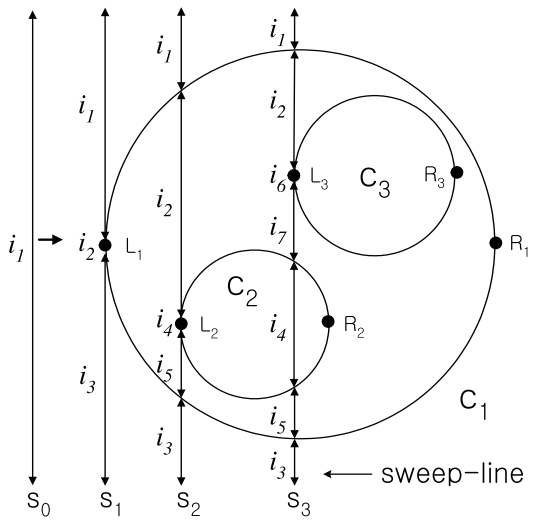


Fig. 5. The creation of intervals at opening events.

where the interval is defined by $-\infty$ and ∞. As the sweep-line proceeds to L_1, as shown by S_1 in Fig. 5, interval i_1 is divided into three intervals $[i_1, i_2, i_3]$. Hence, the sweep-line now splits into three intervals $[i_1, i_2, i_3]$. Both the lower value of i_1 and the upper value of i_3 are the same as Y-coordinate value of L_1, and i_2 is a zero-interval. Then, the sweep-line proceeds to the next event at L_2, and the one of the intervals is further divided into more subintervals.

In Fig. 6 (b), S_2 is tangent to the circle c_2 at L_2 and intersects c_1 at exactly two points since L_2 is between L_1 and R_1. Then, the interval values of i_1, i_2 and i_3 should be updated to reflect the changes due to the intersection between S_2 and c_1. Note that the interval i_2, which was a zero-interval at S_1 in Fig. 6 (a), has the lower and the upper values defined by the intersections between c_1 and S_2, respectively. Therefore, we divide the interval i_2 into 3 subintervals, i_2, i_4, and i_5, since the interval i_2 now contains L_2 in the sweep-line at S_2. In addition, the interval values of i_2 are updated and two new intervals, i_4 and i_5, are created with appropriate interval values.

In Fig. 6 (b), for example, L_2 is located in i_2 and, therefore, the lower value of i_2 has to be updated once more with the Y-coordinate value of L_2. Then, new intervals i_4 and i_5 are inserted below i_2 so that the sweep-line now consists of five intervals $[i_1, i_2, i_4, i_5, i_3]$ at S_2. Therefore, when a sweep-line meets L_i, we create two more intervals where one of them is always a zero-interval. What happens to the sweep-line at other opening event is similar to what happened at S_2.

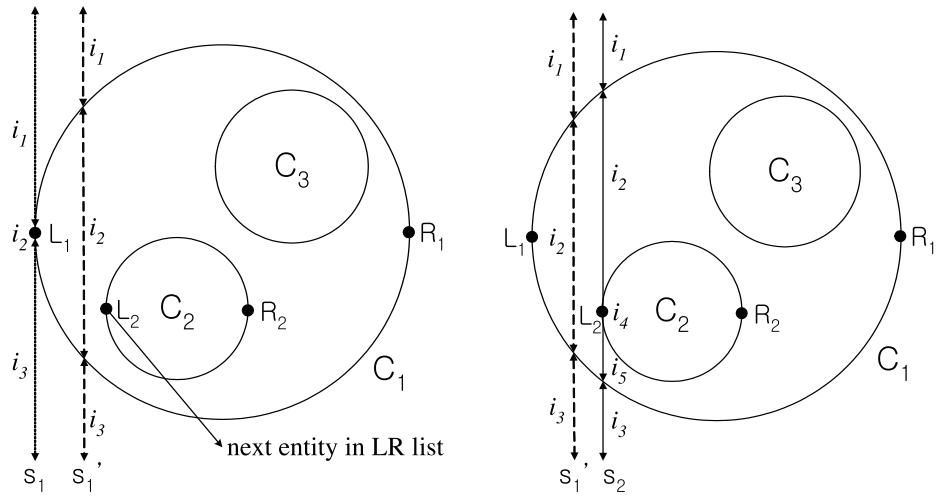


Fig. 6. A transition between two consecutive opening events. (a) immediately before a new opening event, (b) immediately after hitting the opening event.

2.3. Detection of inclusion relationship

When a sweep-line hits a circle at its leftmost extreme point, the decision can be made for the inclusion relationship between the circle and the others met

before. In Fig. 6, for example, when the sweep-line S_2 hits the circle c_2 at L_2, we can immediately know that L_2 is in the interval i_2 which corresponds to the circle c_1 at S'_1. Hence, the circle c_1 should include the circle c_2. In other words, if an interval corresponding to a circle c_i includes the Y-coordinate value of the leftmost extreme point L_j of another circle c_j, then c_i includes c_j.

2.4. Interval deletion

When a sweep-line hits the rightmost extreme point of a circle, e.g. S_4 in Fig. 7 (a), a closing event has occurred. In this case, two intervals, created at the opening event of the corresponding circle, should be deleted. For example, the intervals i_4 and i_5 for the sweep-line at S_3 are deleted when the sweep-line is at S_4 as shown in Fig. 7 (a). In addition, the lower value of interval i_7 and the upper value of interval i_3 are updated, respectively. Fig. 7 (b) shows the modified intervals of the sweep-line at S_4 after the intervals i_4 and i_5 are deleted.

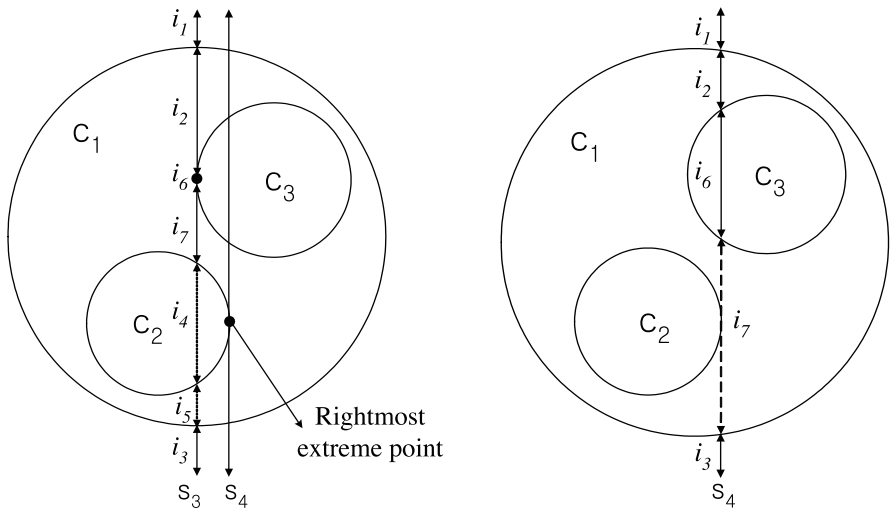


Fig. 7. Deletion of two intervals at a closing event. (a) two intervals, i_4 and i_5, are to be deleted at a right extreme point, (b) the modified intervals of the sweep-line at S_4 after deletion of two intervals.

2.5. Algorithm

The preceding explanation for making a hierarchy of inclusion relationship among circles can be summarized as follows. Let $c(p)$ be the circle associated with the event p, and $y(p)$ be the Y-coordinate of the event point p. Recall that the set of intervals treated here is just the division of the sweep-line into non-overlapping intervals.

Algorithm (Construction of inclusion hierarchy)

Input : A set of circles $C = \{c_1, c_2, \ldots, c_n\}$ in the plane.
Output : An inclusion hierarchy among the circles.

Procedure:
1. Initialize the interval list I as $I = (i_0)$. I is maintained in a red-black tree.
2. Calculate the extreme points of the circles in C, and store them in an array E.
3. Sort the extreme points in E in the nondecreasing order of their x coordinates, and let $P = (p_1, p_2, \ldots, p_{2n})$ be the resulting event list.
4. **For** $j \leftarrow 1$ **until** $2n$ **do**
   if p_j is an opening event, **then do**
      begin
      find the interval $i = (y, y')$ including p_j from the red-black tree I;
      if there is a circle c_k that includes i,
         report "the circle c_k includes the circle $c(p_j)$";
      else
         report "no circle includes the circle $c(p_j)$";
      generate two new intervals $i' = (y, y(p_i))$, $i'' = (y(p_i), y')$, and
         insert them in the list I;
      end
   else
      delete from I the two intervals associated with the circle $c(p_j)$;

3. Data Structure

Since the design of appropriate data structure is essential for the design of the algorithm, we explain the data structures for representing the inclusion hierarchy. For the efficiency of our algorithm, we have used a red-black tree as the primary data structure for the intervals and some secondary data structures for circles, extreme points and intervals.

A circle is associated with its center, the radius, the left and right extreme points. In addition, an extreme point, either L_i or R_i, is represented by its X-coordinate value, a pointer to its generating circle, and a pointer to an interval created at the extreme point as a zero-interval. Note that an extreme point is also pointed from the corresponding generating circle. However, the Y-coordinate value of an extreme point needs not be explicitly stored since it is identical to the Y-coordinate of the center of the generating circle.

An interval is represented by a lower value, an upper value, and three pointers to related circles. There is a pointer, denoted as `CC_PTR`, to a containing circle from an interval in which the interval is contained and the value of this pointer is determined when the interval is created. When the interval is split by a new leftmost extreme point, this pointer is duplicated to the non-zero-interval. Two

more pointers are `UC_PTR` and `BC_PTR` pointing to circles which yield the upper and the lower values of the interval, respectively.

In Fig. 6 (b), for example, both i_2 and i_5 point to c_1, while i_4 points to c_2 as their containing circles, in their `CC_PTR` pointers. The `CC_PTR` pointer of an interval to its containing circle is used to determine the hierarchical relationship between two circles in $O(1)$ time. For example, since L_2 lies in the interval i_2 which points c_1 as its generating circle, the decision that c_1 contains c_2 can be made immediately at L_2 by simply checking which circle i_2 points as a containing circle. Due to two more pointers `UC_PTR` and `BC_PTR`, new values of the upper and the lower of an interval can be also computed in $O(1)$ time for the sweep-line at each new event.

On the other hand, the intervals are maintained in two different data structures simultaneously: a doubly-linked list and a red-black tree. Intervals are stored in a doubly-linked list in the ascending or descending order of the interval values so that the immediate lower and upper interval values can be determined in $O(1)$ time. This is necessary for the constant time update of new interval values. In addition, an extreme point and the generating circle for the extreme point are pointed from each other. At the closing event, therefore, we can also find the interval to be deleted in $O(1)$ time.

At the same time, the intervals are also maintained in a red-black tree to facilitate faster operations on the intervals. In the red-black tree, operations such as insertion, search and deletion have $O(\log n)$ time complexity in the worst-case.

4. Red-Black Tree

The intervals created by the sweep-line play the major role in the construction of inclusion hierarchy among circles. Since the efficient representation intervals is of importance and the intervals are already ordered according to their Y-coordinate values, the binary search tree is naturally considered as a primary data structure. Among various binary search trees including AVL and (2,4) trees, we employed a red-black tree to store the interval list [3, p. 253] [4, p. 379]. Even though AVL and (2,4) trees also provide search, insert, and delete operations having $O(\log n)$ worst-case time complexity like a red-black tree, they have some drawbacks.

In the case of the AVL tree, when an insertion occurs, one tri-node restructuring operation, called a rotation, is sufficient to restore the height-balance property globally. After deletion of an element deletion, however, it is necessary for an AVL-tree to do $O(\log n)$ restructuring to restore the height-balance property globally. Similarly, the (2,4) tree may require several fusing and split operations to be conducted after an insertion or deletion. On the other hand, a red-black tree, requires only $O(1)$ operation after insertion or deletion to make the tree balanced [4, p. 416].

4.1. Interval search

Each time the sweep-line hits a new circle, a new leftmost extreme point is produced and consequently we need to locate in which interval the extreme point lies. Since this search decides the hierarchical relationship between two circles, it

should be done efficiently. Since the intervals are stored in a red-black tree using the interval values as keys, the appropriate interval can be found in $O(\log n)$ time.

Note, however, that the upper and the lower values of an interval, in particular the interval that the current extreme point lies in, are computed from the old sweep-line at the previous event. Hence, it is necessary to update the interval values before the search for an interval containing a current extreme point.

4.2. Interval insertion

When the sweep-line hits a leftmost extreme point L_m of a circle at S_m, two new intervals are created. Let i_{k+1} and i_{k+2} be those new intervals and suppose that i_k contains L_m.

Then, i_{k+1} and i_{k+2} should be inserted below the interval i_k containing the corresponding leftmost extreme point L_m. Using the standard insertion operation for a red-black tree, these new intervals can be inserted into the tree in $O(\log n)$ time. After the interval i_k is found in $O(\log n)$ time in the tree, new intervals are also maintained in the linked list in $O(1)$ time.

4.3. Interval deletion

When a closing event occurs at the rightmost extreme point R_m, appropriate intervals corresponding to L_m have to be deleted from the data structure. Since the closing event at R_m means that the corresponding circle c_m is now being closed, two intervals related to the circle should be removed from the interval set. Note that R_m has a pointer to c_m, and c_m has also a pointer to those intervals. Hence, those intervals can be deleted from the interval list in $O(1)$ time and can be also deleted from the interval tree using the standard delete operation for a red-black tree in $O(\log n)$ time in the worst-case. However, it should be cautioned that the interval values of the intervals immediately before and after the deleted intervals should be updated by merging two intervals into one. Note that it can be done in $O(1)$ time.

5. Time Complexity

Given n circles in the plane, the time complexity of the proposed algorithm is $O(n \log n)$ time in the worst-case. In the first step, a sorted extreme point list can be constructed in $O(n \log n)$ time in the worst-case. In addition, the proposed algorithm traverses each point in the sorted extreme point list only once.

At each leftmost extreme point in the extreme point list, two intervals are created and inserted to the interval tree, stored in a red-black tree, as explained in the previous section. Then, an interval search operation should be performed so that two new intervals can be inserted underneath the interval found in the tree. On the other hand, at each rightmost extreme point, two intervals are deleted from the interval tree. Both of insertion and deletion are done in $O(\log n)$ time in the worst-case. The tree restructuring operation which has to be done after the insertion or deletion takes only $O(1)$ time in the worst-case for a red-black tree. Hence, all three operations, the interval insertion, search, and deletion, can be done

in $O(\log n)$ time in the worst-case. In addition, a circle containing another can be determined in $O(1)$ time since each interval has a pointer to its generating circle.

Therefore, the proposed algorithm guarantees $O(n \log n)$ time complexity in the worst-case to construct the hierarchy of inclusion among n circles in the plane.

6. Conclusions

In this paper, we have presented a plane-sweep algorithm to construct the inclusion hierarchy among circles in the plane, where circles have different radii and do not intersect at their circumferences. Since there are various applications for this problem such as the simulation of physical and chemical processes, the development of an efficient algorithm for this problem is importance.

The presented algorithm, based on the sweep-line concept in the plane, creates a sorted list of extreme points for the circles and then takes care of events at those extreme points by creating appropriate data structures. The algorithm uses a red-black tree to maintain intervals created at each event so that efficient search, insertion, and deletion can be done. The presented algorithm constructs the hierarchy among n circles in the plane in $O(n \log n)$ time in the worst-case.

Acknowledgements. The first two authors were supported by the Creative Research Initiatives from Ministry of Science and Technology in Korea, and the third author was supported by Grant-in-Aid for Scientific Research of the Japan Society for the Promotion of Science.

References

[1] M. de Berg, M. van Kreveld, M. Overmars and O. Schwarzkopf, Computational Geometry (2nd Ed.). Springer, 2000.

[2] S. Christensen, L.M. Kristensen and T. Mailund, A sweep-line method for state space exploration. Lecture Notes in Computer Science **2031**, 2001, 450–464.

[3] T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein, Introduction to Algorithms (2nd Ed.). Vol. 2, The MIT Press, 2001.

[4] M.T. Goodrich and R. Tamassia, Data Structures and Algorithms in Java (2nd Ed.). Wiley, 2001.

[5] C.-W. Hong, From long-range interaction to solid-body contact between colloidal surfaces during forming. Journal of the European Ceramic Society, **18** (1998), 2159–2167.

[6] D.-S. Kim, D. Kim and K. Sugihara, Voronoi diagram of a circle set from Voronoi diagram of a point set: I. Topology. Computer Aided Geometric Design, **18** (2001), 541–562.

[7] D.-S. Kim, D. Kim and K. Sugihara, Voronoi diagram of a circle set from Voronoi diagram of a point set: II. Geometry. Computer Aided Geometric Design, **18** (2001), 563–585.

[8] D.-S. Kim, D. Kim and K. Sugihara, Voronoi diagram of circles in a large circle. Lecture Notes in Computer Science **2669**, 2003, 847–855.

[9] C. Kimme, D. Ballard and J. Sklansky, Finding circles by an array of accumulators. Communication of ACM, **18**, No.2 (1975), 120–122.

[10] M.D. McIlroy, Best approximate circles on integer grids. ACM Transactions on Graphics, **2**, No.4 (1983), 237–263.

[11] J. Nievergelt and F.P. Preparata, Plane-sweep algorithms for intersecting geometric figures. Communication of ACM, **25**, Issue 10 (1982), 739–747.

[12] C. Oh, J.-H. Park, S.-I. Shin and S.-G. Oh, O/W/O multiple emulsions via one-step emulsification process. Journal of Dispersion Science and Technology, **25**, No.1 (2004), 53–62.

[13] A. Okabe, B. Boots, K. Sugihara and S.N. Chiu, Spatial Tessellations: Concepts and Applications of Voronoi Diagrams (2nd Ed.). John Wiley and Sons, 2000.
[14] J.-H. Park, C. Oh, S.-I. Shin, S.-K. Moon and S.-G. Oh, Preparation of hollow silica microspheres in W/O emulsions with polymers. Journal of Colloid and Interface Science, **266** (2003), 107–114.
[15] F.P. Preparata and M.I. Shamos, Computational Geometry: An Introduction. Springer-Verlag, 1985.