JOURNAL OF ALGORITHMS 4, 310–323 (1983)

Finding the Connected Components and a Maximum Clique of an Intersection Graph of Rectangles in the Plane

HIROSHI IMAI AND TAKAO ASANO

Department of Mathematical Engineering and Instrumentation Physics, Faculty of Engineering, University of Tokyo, Tokyo, Japan

Received August 15, 1981

An intersection graph of rectangles in the (x, y)-plane with sides parallel to the axes is obtained by representing each rectangle by a vertex and connecting two vertices by an edge if and only if the corresponding rectangles intersect. This paper describes algorithms for two problems on intersection graphs of rectangles in the plane. One is an $O(n \log n)$ algorithm for finding the connected components of an intersection graph of n rectangles. This algorithm is optimal to within a constant factor. The other is an $O(n \log n)$ algorithm for finding a maximum clique of such a graph. It seems interesting that the maximum clique problem is polynomially solvable, because other related problems, such as the maximum stable set problem and the minimum clique cover problem, are known to be *NP*-complete for intersection graphs of rectangles. Furthermore, we briefly show that the k-colorability problem on intersection graphs of rectangles is *NP*-complete.

INTRODUCTION

An intersection graph of given n objects in R^k is obtained by representing each object by a vertex and connecting two vertices by an edge if and only if the corresponding objects intersect. It is known that any graph is representable as an intersection graph of convex objects in three or more dimensions, and also that there exist graphs which cannot be represented as intersection graphs of convex objects in two dimensions [12].

In [8], Roberts considered intersection graphs of rectangles (or boxes) with sides parallel to the coordinate axes in the Euclidean space, and showed that the smallest k, such that all graphs with n vertices can be represented as intersection graphs in k dimensions, is $\lceil n/2 \rceil$. If objects are intervals on a one-dimensional line, their intersection graphs are interval

0196-6774/83 $3.00

graphs, which have various good properties from the algorithmic as well as the practical viewpoint [6].

Here we consider two problems on intersection graphs of rectangles with sides parallel to the axes in two dimensions (see Fig. 1). We first consider the problem of finding the connected components of these intersection graphs, and by employing the techniques used in [7] we present an $O(n \log n)$ algorithm, where n is the number of rectangles. By using this algorithm iteratively, we can solve the problem for regular $2m$-gons and some other polygons in $O(n \log n)$ time. Since we can easily show that $\Omega(n \log n)$ is a lower bound on the time required to solve these problems as in the case of the closest-point problem [10], these algorithms are optimal to within a constant factor.

Then, we consider the problem of finding a maximum clique and a more general problem of finding a maximum-weighted clique of such an intersection graph. We show that these problems can also be solved in $O(n \log n)$ time, which seems interesting in view of the fact that the problems of finding a minimum clique cover and of a maximum stable set are *NP*-complete even for these restricted intersection graphs in two dimensions [2]. In the last section, we briefly show the *NP*-completeness of the k-colorability problem on intersection graphs of rectangles.

These problems are of interest in integrated circuit layout techniques, image processing [9], and numerical analysis [7].

1. Finding the Connected Components

1.1. *Rectangles*

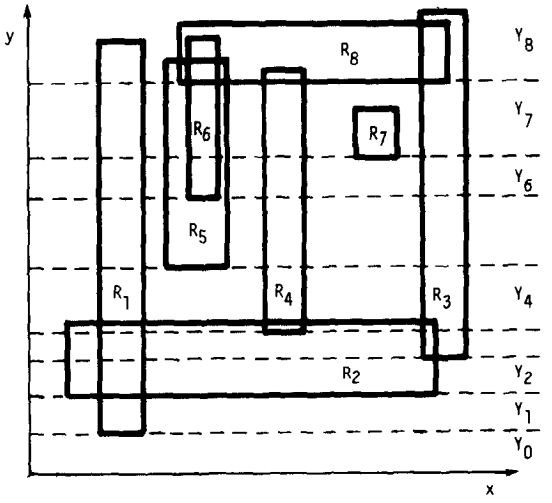
We consider n rectangles $R_i = \{(x, y) | x_i^- \leqslant x \leqslant x_i^+, y_i^- \leqslant y \leqslant y_i^+\}$ $(i = 1, \ldots, n)$. For convenience, we assume that x_i^-, x_i^+ $(i = 1, \ldots, n)$ and y_i^-, y_i^+ $(i = 1, \ldots, n)$ are different from one another, respectively (even if some of them are equal, the argument below is valid with a slight modification [7]).

Of course, there is a naive algorithm for the problem of finding the connected components to check every pair of rectangles for their intersections, but it takes $O(n^2)$ time.

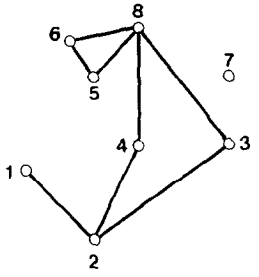
To find the connected components in $O(n \log n)$ time, we partition the plane into $n + 1$ regions, $Y_0, Y_1, \ldots, Y_n$ by n horizontal lines, $y = y_i^-$ $(i = 1, \ldots, n)$ (see Fig. 1), and we iteratively identify the intersections of rectangles in each region Y_k. For this purpose, we denote by V_k the set of indices of the rectangles intersecting region Y_k, and sort $2|V_k|$ numbers x_i^-, x_i^+ $(i \in V_k)$ in increasing order and define $J_k[i, \pm]$, $h_k[j]$, and $g_k[j]$ as follows:

$$J_k[i, -] = j,\ h_k[j] = i, \qquad \text{if } x_i^- \text{ is the } j\text{th number},$$

$$J_k[i, +] = j,\ h_k[j] = i, \qquad \text{if } x_i^+ \text{ is the } j\text{th number},$$



(a)



(b)

$V_8 = \{1,3,4,5,6,8\}$

(c)

j	1	2	3	4	5	6	7	8	9	10	11	12
$x_i^{\pm}$	x_1^-	x_1^+	x_5^-	x_8^-	x_6^-	x_6^+	x_5^+	x_4^-	x_4^+	x_3^-	x_8^+	x_3^+
$g_8[j]$	1	1	1	2	3	3	2	2	2	2	2	1

(d)

FIG. 1. (a) Rectangles with sides parallel to the axes and the partition of the plane. (b) The intersection graph. (c) V_8. (d) $g_8[j]$.

and

$$g_k[j] = (\text{the number of rectangles } R_{i'} \text{ such that } J_k[i', -] \leqslant j \leqslant J_k[i', +]).$$

Note that

$$g_k[j] = (\text{the number of } i' \in V_k \text{ such that } J_k[i', -] \leqslant j) - (\text{the number of } i' \in V_k \text{ such that } J_k[i', +] < j).$$

With these definitions, the following lemma holds, which plays the central role in the algorithm.

LEMMA 1. *Let* $j^- = J_k[i, -]$ *and* $j^+ = J_k[i, +]$ *for* $i \in V_k$.

(i) *Rectangle* R_i *intersects all rectangles* $R_{i'}$ *such that* $j^- < J_k[i', \pm] < j^+$ (*in this paper*, "$\pm$" *always means* "$+$" *or* "$-$").

(ii) *If* $g_k[j^-] \neq 1$, *rectangle* R_i *intersects rectangle* $R_{i'}$ *or another rectangle which intersects rectangle* $R_{i'}$, *where* $i' = h_k[j^- - 1]$.

(iii) *If* $g_k[j^+] \neq 1$, *rectangle* R_i *intersects rectangle* $R_{i'}$ *or another rectangle which intersects rectangle* $R_{i'}$, *where* $i' = h_k[j^+ + 1]$.

Based on Lemma 1, we can find the connected component, to which rectangle R_i is to belong by intersections within the region Y_k. Thus we obtain the following algorithm. We denote by $M[i]$ the set of indices of the rectangles which have been found to be in the same connected component containing rectangle R_i.

procedure CONNECTED__COMPONENTS;

```
0.  begin
1.    M[i]:= {i}, for i = 1,..., n;
2.    for k:= 1 to n do
3.      begin
4.        let i ∈ V_k − V_{k−1}; (such i is uniquely determined.)
5.        j⁻:= J_k[i, −]; j⁺:= J_k[i, +];
6.        for all i' ∈ V_k such that j⁻ < J_k[i', ±] < j⁺, union M[i]
          and M[i'];
7.        if g_k[j⁻] ≠ 1, union M[i] and M[h_k[j⁻ − 1]];
8.        if g_k[j⁺] ≠ 1, union M[i] and M[h_k[j⁺ + 1]]
9.      end
10. end.
```

The validity of this algorithm can easily be shown by the induction on k by applying Lemma 1 at each step, so that the detailed proof is omitted.

Let us consider how to implement this algorithm in $O(n \log n)$ time. We represent each V_k by a 2-3 tree [1]. In the tree for V_k (hereafter, we denote

this tree by $T(V_k)$), the leaves correspond to x_i^-, x_i^+ ($i \in V_k$), which are arranged in increasing order (see Fig. 2). In the algorithm, the tree $T(V_k)$ is constructed from the tree $T(V_{k-1})$ by deleting the leaves corresponding to $x_{i'}^-$, $x_{i'}^+$ such that $i' \in V_{k-1} - V_k$, and inserting new two leaves corresponding to x_i^-, x_i^+ such that $i \in V_k - V_{k-1}$ into the tree. A 2-3 tree allows us to execute insertion or deletion of a leaf in $O(\log n)$ time.

Moreover, a 2-3 tree makes it possible to update $g_k[j]$ by executing insertion and deletion in $O(\log n)$ time in the following way. We implicitly represent $g_k[j]$ by attaching a number to each node of the tree. That is, with each node n_p we associate $f[n_p]$ such that

$$g_k[J_k[i, \pm]] = \sum_{n_p} f[n_p],$$

where the summation is taken over nodes n_p which lie on the path from the leaf corresponding to $x_i^{\pm}$ to the root of the tree (see Fig. 2). In the algorithm we must compute $g_k[j^-]$ and $g_k[j^+]$, and update $f[n_p]$ and $g_k[j]$ by executing insertion and deletion of a leaf. Each one of these manipulations can be done in $O(\log n)$ time. The data structures described here are similar to those devised by Galil and Naamad [3] for the maximum network-flow algorithm. For completeness, we describe how to manipulate these data structures in Appendix 1. Thus the execution of the algorithm except for step 6 takes $O(n \log n)$ time in total if we use the data structures described above and the Union-Find algorithm [11].

The remaining problem is how to implement step 6 of the algorithm. A naive implementation, to scan every leaf of $T(V_k)$ between the leaves corresponding to x_i^-, x_i^+, apparently takes $O(n^2)$ time in total.

At the beginning of the kth iteration of steps 2–9 of the algorithm, we can assume that the connected components of an intersection graph in region $Y_0 \cup Y_1 \cup \cdots \cup Y_{k-1}$ have been found. Suppose that i is found at step 4. At that time, $M[i] = \{i\}$, and the set of $V_k - \{i\}$ is partitioned into several connected components as assumed. Consider a graph L_k, which is defined by representing each leaf of the tree $T(V_k)$ by a vertex and

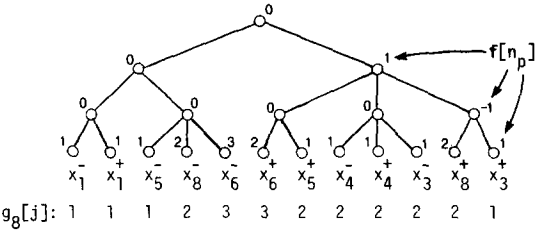


FIG. 2. The 2-3 tree $T(V_8)$.

connecting two vertices corresponding to the adjacent leaves by an edge (so that L_k consists of one simple path). Then we see that the task at step 6 is to find the set of the connected components which contain the vertex on the path between the two vertices corresponding to x_i^-, x_i^+ on the graph L_k. Furthermore, we define a graph L_k^* by contracting every edge of L_k whose end vertices correspond to the same connected component at that time. Let the set of vertices of L_k^* be $\{u_1, \ldots, u_m\}$, the set of edges be $\{(u_p, u_{p+1}) | 1 \leqslant p < m\}$ and $S[u_p]$ be the connected component to which u_p corresponds at that time. (Note that L_k^* is again one simple path.) Then we have only to find the vertices on the path of L_k^* between the two vertices corresponding to x_i^-, x_i^+. The properties of the graph L_k^* can be summarized as follows:

LEMMA 2.

(i) $S[u_p] \neq S[u_{p+1}]$ $(1 \leqslant p < m)$.

(ii) *If* $S[u_p] = S[u_q] \neq S[u_r]$ *for* $p < r < q$, *where none of* u_p, u_q *and* u_r *correspond to* $M[i] = \{i\}$, *then* $S[u_w] \neq S[u_r]$ *for any* w *such that either* $w < p$ *or* $w > q$ (*see Fig.* 3).

Proof. (i) By definition.

(ii) Since $S[u_p] = S[u_q]$, the region $Y_0 \cup Y_1 \cup \cdots \cup Y_{k-1}$ is divided, by the chain of intersecting rectangles whose ends are R_p and R_q, into the interior region containing R_r and the exterior region containing R_w. Consequently,

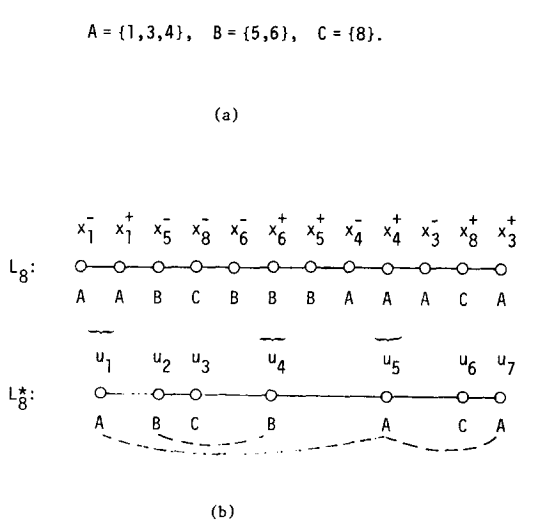


FIG. 3. (a) Connected components (just before the execution of step 6 at the 8th iteration). (b) Graphs L_8 and L_8^*.

if $S[u_r] = S[u_w]$, there must exist a chain of intersecting rectangles whose ends are R_r and R_w, and this chain must cross the chain whose ends are R_p and R_q. This would imply that $S[u_p] = S[u_q] = S[u_r] = S[u_w]$, which contradicts the hypothesis that $S[u_p] = S[u_q] \neq S[u_r]$. □

We also assume that there exists a rectangle $R_{i'}$ such that $j^- < J_k[i', \pm] < j^+$, where $j^- = J_k[i, -]$ and $j^+ = J_k[i, +]$. (Otherwise, step 6 of the algorithm can be dispensed with at the kth iteration.) Let $i_p = h_k[j^- + 1]$ and $i_q = h_k[j^+ - 1]$, and u_p and u_q be the vertices of L_k^* to which i_p and i_q correspond, respectively. Then $a_k = q - p + 1$ equals the number of vertices of L_k^* on the path from u_p to u_q. If we denote by b_k the number of connected components which are to be joined with $M[i]$ at step 6 of the kth iteration, the following lemma holds.

LEMMA 3. $a_k \leqslant 2b_k - 1$.

Proof. Induction on a_k. In the case of $a_k = 1$, obvious. Assume that the lemma holds for all $a_k < K (K \geqq 2)$. Let $a_k = K$. If $b_k \geqq a_k - 1$, then by Lemma 2 (i) the lemma holds. Thus we can assume that $b_k \leqq a_k - 2$ and that $S[u_{p'}] = S[u_{q'}]$ for some vertices $u_{p'}$ and $u_{q'}$ such that $p \leqslant p' < q' < q$. Now consider two subpaths Q_1 and Q_2:

$$Q_1 = u_p, u_{p+1}, \ldots, u_{p'}, u_{q'+1}, \ldots, u_q;$$
$$Q_2 = u_{p'}, u_{p'+1}, \ldots, u_{q'}.$$

Let a_k^m $(m = 1, 2)$ be the number of vertices on Q_m, and b_k^m $(m = 1, 2)$ be the number of connected components in Q_m. Since both Q_1 and Q_2 have $K - 1$ or less vertices, we have by the induction hypothesis,

$$a_k + 1 = a_k^1 + a_k^2 \leqslant (2b_k^1 - 1) + (2b_k^2 - 1).$$

By Lemma 2 and $S[u_{p'}] = S[u_{q'}]$, $b_k = b_k^1 + b_k^2 - 1$, thus we have

$$a_k \leqslant 2b_k - 1. \quad \square$$

Lemma 3 assures that the time to scan the vertices on that path of L_k^* is $O(b_k)$. (Note that $\Sigma_k b_k \leqslant n - 1$.)

By representing L_k^* by another 2-3 tree, whose ordered leaves correspond to the ordered vertices $(u_1, \ldots, u_m)$ of L_k^*, and providing the additional lists of those leaves, we can execute step 6 of the algorithm in $O(b_k \log n + \log n)$ time. The detailed manipulations of this 2-3 tree are described in Appendix 2.

From the above considerations, the execution of the kth iteration takes at most $C(b_k \log n + \log n)$ time, where C is a constant independent of k. Thus this algorithm takes at most $C(\Sigma_k (b_k \log n + \log n))$ time in total. Since $\Sigma_k b_k \leqslant n - 1$, this bound is $O(n \log n)$.

1.2. *Regular 2m-gons and Others*

Here we consider the intersection graphs of regular $2m$-gons P_i ($i = 1,\ldots, n$), whose sides are parallel with each other. Then each regular $2m$-gon P_i can be represented as the union of m rectangles $R_i^1,\ldots, R_i^m$ which are composed of m opposite sides of the polygon, where all sides of rectangles $R_1^q, R_2^q,\ldots, R_n^q$ ($1 \leqslant q \leqslant m$) are parallel to some coordinate axes (see Fig. 4). By using these rectangles, we can identify the intersection of these two polygons as in the following lemma.

LEMMA 4. *Two regular $2m$-gons P_i and $P_{i'}$, whose sides are parallel with each other, intersect if and only if at least one of the corresponding pair of rectangles R_i^q and $R_{i'}^q$ ($1 \leqslant q \leqslant m$) intersect.*

Lemma 4 can be proved by plane geometry, and thus we obtain the following algorithm.

procedure CONNECTED_COMPONENTS; (for $2m$-gons)

1. *begin*
2. find partition t_q of $\{1,\ldots, n\}$ for $q = 1,\ldots, m$ by the connected components of the intersection graph of R_i^q ($i = 1,\ldots, n$);
3. find the finest partition t^* such that every t_q is a refinement of t^*;
4. *end*.

The validity of this algorithm is evident from Lemma 4, that is, we can easily show that P_i and $P_{i'}$ intersect if and only if i and i' are contained in the same component of t^*. It is easy to implement this algorithm with the time bound of $O(mn \log n)$, because the problem of finding the finest partition such that given two partitions are its refinements can be solved in almost linear time by using the Union-Find algorithm [11]. This algorithm is $O(n \log n)$ if m is thought to be a constant. The algorithm works well with the same time bound for other types of polygons so long as they satisfy the property of Lemma 4.

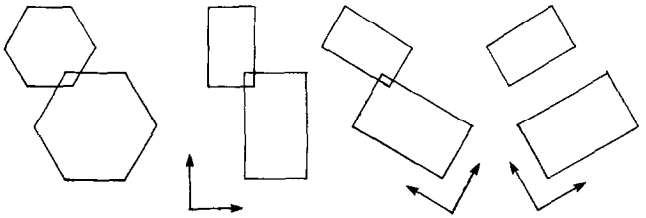


FIG. 4. Representation of regular hexagon as the union of three rectangles.

2. Finding a Maximum Clique

Though the problem of finding a maximum clique in a general graph is *NP*-complete, it can be shown that the one for intersection graphs of rectangles can be solved in polynomial time. We present an $O(n \log n)$ algorithm for this problem. The following lemma shows one of the good properties of the intersection graphs of rectangles which simplify the problem.

Lemma 5. *A family of subsets of the set of rectangles R_i $(i = 1, \ldots, n)$ with sides parallel to the axes satisfies the Helly property. That is, for any subset of $\{R_1, \ldots, R_n\}$ of rectangles of which every pair intersects, all the rectangles contained in the subset have nonempty intersection.*

Consequently, for any clique of an intersection graph of rectangles, there exists a rectangular region which is included in the rectangle corresponding to every vertex contained in the clique and vice versa. Thus, the problem of finding a maximum clique of an intersection graph of rectangles is equivalent to the problem of finding a rectangular region which is included in the maximum number of rectangles.

First, we concentrate our attention on the problem of finding the maximum number d^* of rectangles which intersect one another. Let

$$d_k = \max\left\{|I| \,\middle|\, \bigcap_{i \in I} R_i \neq \phi, I \subset V_k\right\},$$

$$\text{then } d^* = \max_k \{d_k\}.$$

Moreover the following lemma holds.

Lemma 6. $d_k = \max\{g_k[j] \mid j = J_k[i, \pm], i \in V_k\}$.

Thus the following algorithm is given.

procedure MAXIMUM_CLIQUE;

```
1.  begin
2.      d*:= 0;
3.      for k:= 1 to n do
4.          begin
5.              d_k:= max{g_k[j] | j = J_k[i, ±], i ∈ V_k};
6.              d*:= max{d*, d_k};
7.          end;
8.  end.
```

As in the case of computing and updating $g_k[j]$ in finding the connected components, a simple implementation is possible to scan every $g_k[j]$ to find

d_k in $O(n^2)$ time. To compute d_k quickly, we attach a number $c[n_p]$ besides $f[n_p]$ to each node n_p in the tree, where $c[n_p]$ is defined as follows:

$$c[n_p] = f[n_p] \qquad \text{if } n_p \text{ is a leaf,}$$

$$c[n_p] = f[n_p] + \max\{c[n_q]\} \qquad \text{if } n_p \text{ is not a leaf,}$$

where the max is taken over the nodes n_q which are sons of n_p. Then, apparently we have

$$d_k = c[n_{\text{root}}].$$

In so doing, we can execute insertion or deletion together with updating $f[n_p]$ and $c[n_p]$ in $O(\log n)$ time, if we resort to the data structures of Galil and Naamad [3] (see Appendix 1). Thus we can find d_k in $O(\log n)$ time, and d^* in $O(n \log n)$ time in total.

Once the maximum number d^* of rectangles which intersect one another is known, it is easy to find a maximum clique itself in $O(n \log n)$ time, because we have only to reconstruct the tree such that $c[n_{\text{root}}] = d^*$ and identify the vertices of the maximum clique by scanning this tree.

This algorithm can easily be generalized for the weighted problem. To each vertex v of an intersection graph of rectangles, we associate a non-negative number $w(v)$ (for instance, an area of rectangle corresponding to vertex v), and define the weight of a clique S by

$$w(S) = \sum_{v \in S} w(v).$$

The problem is to find a maximum-weighted clique of an intersection graph of rectangles.

This problem can be solved in $O(n \log n)$ time by modifying $g_k[j]$ (consequently, $f[n_p]$ and $c[n_p]$) in the algorithm for finding a maximum clique as follows:

$$g_k[j] = \sum w(i') \qquad (j = J_k[i, \pm]),$$

where the summation is taken over all i' such that $J_k[i', -] \leqslant j \leqslant J_k[i', +]$.

A simple extension of the above algorithm to higher dimensions yields an $O(n^{k-1} \log n)$ algorithm for the problem of finding a maximum clique of an intersection graph of given n rectangles in k dimensions, which is polynomially bounded if k is thought to be a constant. Note, however, that the problem of discerning whether a given graph is the intersection graph of rectangles with sides parallel to the axes in two dimensions is known to be *NP*-complete [5]. Consequently, we can apply the above algorithm only to a graph which is represented as an intersection graph of rectangles.

3. Remark on the Complexity of Coloring Intersection Graphs of Rectangles.

We have shown that the problem of finding a maximum clique of an intersection graph of rectangles can be solved in polynomial (specifically, $O(n \log n)$) time, whereas it is known that the problems of finding a minimum clique cover and a maximum stable set are *NP*-complete [2].

In connection with the maximum clique problem, the k-colorability problem of intersection graphs of rectangles will be worth noting. First, we claim that the 3-colorability problem of such graphs is *NP*-complete. The proof is straightforward from the result of [4] such that the 3-satisfiability problem is polynomially transformable to the 3-colorability problem. In [4], a graph G is constructed so that G is 3-colorable if and only if the given instance of the 3-satisfiability problem is satisfiable. We can easily see that such a graph G is always representable as an intersection graph of rectangles in the plane. Thus the 3-colorability problem of these intersection graphs is *NP*-complete.

This implies that the k-colorability problem of an intersection graph of rectangles is *NP*-complete if k is considered as part of "instance." Even if $k \geqslant 3$ is fixed, it can easily be shown that the k-colorability problem is also *NP*-complete by induction on k.

4. Concluding Remarks

We have presented $O(n \log n)$ algorithms for the problem of finding the connected components as well as a maximum clique of an intersection graph of rectangles in the plane, where the time bound of $O(n \log n)$ is achieved by using sophisticated data structures.

It is interesting that a maximum clique of an intersection graph can be found in polynomial time, where the Helly property plays an important role. Further characterization of an intersection graph of rectangles such as that in [8] will enable us to generalize the results described here.

Appendix 1

Here we consider how to update both $f[n_p]$ and $c[n_p]$ with executing insertion and deletion. (Basic operations of insertion, deletion and searching where the leaf should be inserted are described in [1].) We only consider the manipulation about the leaf corresponding to x_i^- (in the case of x_i^+, the argument below can be applied with a slight modification). For a leaf n_p, we define $U[n_p]$ as the set of nodes which are sons of the nodes on the path

from n_p to the root and lie on the righthand side of that path (see Fig. 5). (Note that the cardinality of $U[n_p]$ is $O(\log n)$.)

We sometimes transfer $f[n_p]$ from the node n_p to its sons n_q. This is achieved by

$$(f[n_q], c[n_q]) := (f[n_q] + f[n_p], c[n_q] + f[n_p]),$$
$$(f[n_p], c[n_p]) := (0, c[n_p]).$$

When we let the node n_r son of node n_t (for instance, inserting a new leaf), we must transfer $f[n_p]$ from the node n_p to the sons of n_p, where n_p is on the path from n_t to the root, because without transferring $f[n_p]$ the total sum of $f[n_p]$ on the path from n_t to the root may not be zero and the value of $g_k[j]$ may become wrong.

After transforming the tree by inserting a new leaf n_s (before transforming the tree by deleting the leaf n_s) corresponding to x_i^-, in order to update $f[n_p]$ and $c[n_p]$ correctly, we have only to add one to (subtract one from) $f[n_p]$ such that $n_p \in U[n_s]$, which has effect to add one to (subtract one from) $g_k[j]$ such that $J_k[i, -] < j$, and compute $c[n_p]$ of node n_p on the path from n_s to the root, where $c[n_s] := f[n_s]$.

By manipulating the 2-3 tree in the above way, we can execute insertion and deletion in $O(\log n)$ time.

APPENDIX 2

Here we consider how to manipulate the 2-3 tree for L_k^* in order to execute step 6 at the kth iteration in $O(b_k \log n + \log n)$ time.

Since the execution of step 6 modifies several components, it is convenient for us to define another graph LW_k by contracting every edge of L_k^*

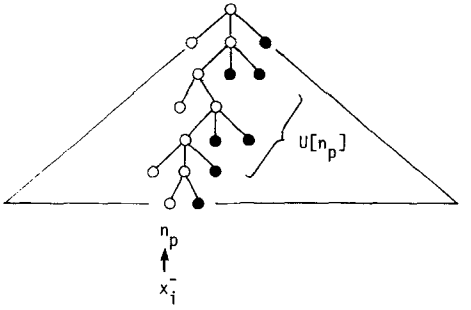


FIG. 5. $U[n_p]$.

whose end vertices correspond to the same connected component just after the execution of step 6 of the kth iteration. We denote the 2-3 tree for $L_k^*(LW_k)$ by $T(L_k^*)(T(LW_k))$. The problem can be considered to consist of two stages.

(i) Constructing $T(L_k^*)$ from $T(LW_{k-1})$. At this stage, the leaves corresponding to i such that $i \in V_{k-1} - V_k$ are deleted to get $T(V_k)$ out of $T(V_{k-1})$. Then, the vertices of LW_{k-1} which consist of only those i's disappear in the L_k^*. Therefore we must delete those leaves of $T(LW_{k-1})$ corresponding to those vertex of LW_{k-1}. During these deletions, the two leaves corresponding to the same component may become adjacent by deleting the only leaf between them that corresponds to another component. (Consider the case of deleting the leaves corresponding to $M[7] = \langle 7 \rangle$ at the seventh iteration in Fig. 1.) But it occurs at most once for each deletion of a leaf.

After completing up these deletions, we must insert the two leaves corresponding to $M[i] = \langle i \rangle$ at step 4 of the kth iteration, where $i \in V_k - V_{k-1}$. In inserting the leaf corresponding to $x_i^{\pm}$, if in L_k both vertices, adjacent to the vertex which corresponds to $x_i^{\pm}$, correspond to the same component at that time, then this insertion may cause splitting the leaf corresponding to those two vertices of L_k. (Consider the case of inserting the leaf corresponding to x_8^+ at the eighth iteration in Fig. 1.) However, it occurs at most once for each insertion.

By these manipulations, $T(L_k^*)$ is obtained from $T(LW_{k-1})$.

(ii) The execution of step 6 of the kth iteration (constructing $T(LW_k)$ from $T(L_k^*)$). Here we must contract every edge of L_k^* whose end vertices are found to correspond to the connected component joined with $M[i]$. It is easy to contract the edges on the path between the two vertices of L_k^* corresponding to i in $O(b_k \log |L_k^*|)$ time, because we have only to scan this path with deleting every leaf between two leaves corresponding to i on the tree $T(L_k^*)$.

Before then, however, there arises the problem to contract the edges which are not on that path of L_k^*. Consider the case to execute step 6 at the eighth iteration in Fig. 1, where two vertices of L_k^*, which correspond to the connected component containing R_1 and R_5, respectively, before executing step 6, come to correspond to the same component at step 6. However, the edge of L_k^*, whose end vertices are those two vertices, is not on the path of L_k^* between two vertices corresponding to x_i^-, x_i^+.

We can show that the number of those edges is at most $(a_k + 1)$ by using Lemma 2. To identify those edges quickly, we keep a doubly linked list for each component, which links the vertices of L_k^* corresponding to the same component according to the order that the vertices of L_k^* are arranged. By providing these additional lists, we can contract those edges in $O(b_k \log |L_k^*|)$ time, and in total, execute step 6 in $O(b_k \log |L_k^*|)$ time.

ACKNOWLEDGMENTS

The authors are deeply indebted to Professor Masao Iri of the University of Tokyo for bringing these problems to their attention, and to Dr. Masataka Nakamura and Dr. Kazuo Murota for their valuable advice. This research was supported by the Grant-in-Aid for Scientific Research of the Ministry of Education, Science and Culture of Japan under Grant: YSE(A) 56750231 (1981).

REFERENCES

1. A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, "The Design and Analysis of Computer Algorithms," Addison-Wesley, Reading, Mass., 1974.
2. R. J. FOWLER, M. S. PATERSON, AND S. L. TANIMOTO, Optimal packing and covering in the plane are *NP*-complete, *Inform. Process. Lett.* **12** (1981), 133–137.
3. Z. GALIL AND A. NAAMAD, Network flow and generalized path compression, *in* "Proceedings of the 11th Annual ACM Symposium on Theory of Computing," Atlanta, pp. 13–26, 1979.
4. M. R. GAREY, D. S. JOHNSON, AND L. STOCKMEYER, Some simplified *NP*-complete graph problems, *Theoret. Comput. Sci.* **1** (1976), 237–267.
5. F. GAVRIL, Some *NP*-complete problems on graphs, *in* "Proceedings of the 11th Conference on Information Sciences and Systems," Johns Hopkins University, Baltimore, pp. 91–95, 1977.
6. M. C. GOLUMBIC, "Algorithmic Graph Theory and Perfect Graphs," Academic Press, New York, 1980.
7. H. IMAI, Finding connected components of an intersection graph of squares in the Euclidean plane, *Inform. Process. Lett.* **15** (1982), 125–128.
8. F. S. ROBERTS, On the boxicity and cubicity of a graph, *in* "Recent Progress in Combinatorics" (W. T. Tutte, Ed.), pp. 301–310, Academic Press, London/New York, 1969.
9. H. SAMET, Connected component labeling using quadtrees, *J. Assoc. Comput. Mach.* **28** (1981), 487–501.
10. M. I. SHAMOS AND D. HOEY, Closest-point problems, *in* "Proceedings of the 16th Annual IEEE Symposium on Foundations of Computer Science," New York, pp. 151–162, 1975.
11. R. E. TARJAN, Efficiency of a good but not linear set union algorithm, *J. Assoc. Comput. Mach.* **22** (1975), 215–225.
12. G. WEGNER, Eigenschaften der nerven homologisch-einfacher Familien in R^n, Doctorial dissertation, Göttingen, 1967.