Research Contribution

A Dynamic Algorithm for Placing Rectangles without Overlapping

TAKESHI TOKUYAMA*, TAKAO ASANO** and SHUJI TSUKIYAMA***

We consider the dynamic allocation problem of rectangles such that a mixed sequence of insertions and deletions of rectangles in a rectangular board is executed without any pair of rectangles overlapping each other. We present an $O(n \log \log n)$ time algorithm for insertion of a rectangle if n rectangles have been already placed in the board. We solve the problem by reducing it to the contour construction problem on a special class of arrangements of rectangles.

1. Introduction

The problem of allocating a set of iso-oriented rectangles without overlapping has widespread applications. Assume a rectangular board W and a set S of rectangles are given.

Problem 1. *Find an allocation of the rectangles of S on W such that none overlaps another.*

Problem 1 is a kind of two-dimensional bin packing problem that is NP-hard [1]. Therefore, we would like to consider the following problem of dynamic placement, where we process a series of insertions and deletions, updating suitably preprocessed data.

Problem 2. *Suppose n rectangles $R_1, R_2, \ldots, R_n$ have already been placed on W without overlapping. Then decide whether a rectangle $Rect(\alpha, \beta)$ of width α and height β can be inserted in the orthogonal region $W-R_1 \cup R_2 \cup \cdots \cup R_n$. If so, construct the region consisting of all possible placements of (the right-upper corner of) the inserted rectangle.*

The region consisting of all possible placements defined above is called the *admissible region*. Figure 1 illustrates the problem. The admissible region is important in applications of the dynamic placement problem to chip layout, window systems, scheduling, and map layout. We design an algorithm for constructing an admissible region in $O(n \log \log n)$ time and $O(n)$ space, using $O(n)$ space dynamic data. We reduce the problem to the *contour construction problem* and the *rectangulation problem* on a special class of arrangements of rectangles called *FIFO arrangement*. Our algorithm, which is simple as well as efficient, is essentially a plane sweep algorithm using an $O(\log \log n)$ time priority queue, a

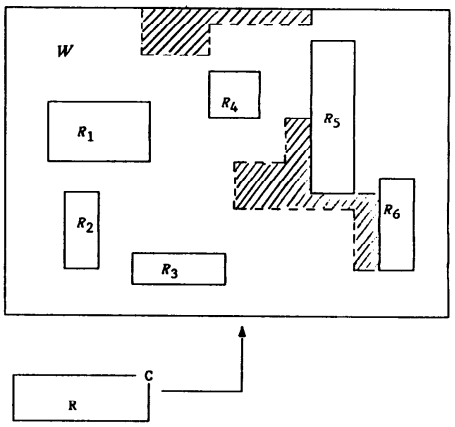

Fig. 1 Admissible region (The union of the shaded regions). The rectangle R can be inserted without overlapping rectangles R_i ($i=1, 2, \ldots, 6$) if the right-upper corner C is located in the admissible region.

visible skeleton, and *saving boxes*, which are defined later.

2. Admissible Region and Extended Rectangles

Let us characterize the admissible region mentioned above. Suppose that $R=[l, r] \times [b, t]$ is a rectangle placed on the board with the vertex set $\{(l, b), (l, t), (r, b), (r, t)\}$ such that $l<r$ and $b<t$. If we deal with the insertion problem of a rectangle with width α and height β, we consider the extended rectangle $R[\alpha, \beta]=[l, r+\alpha] \times [b, t+\beta]$ of R. Further, we contract the board $W=[l_w, r_w] \times [b_w, t_w]$ to obtain $W_{\alpha,\beta}=[l_w+\alpha, r_w] \times [b_w+\beta, t_w]$.

Observation 2.1 (Fig. 2)

A rectangle $Rect(\alpha, \beta)$ of width α and height β is placed in $P_0=W-\bigcup_{i=1} R_i$ if and only if its right-upper corner

*IBM Research, Tokyo Research Laboratory, 5-19, Sanbancho, Chiyoda-ku, Tokyo, Japan.
**Faculty of Science and Technology, Sophia University, Chiyoda-ku, Tokyo, Japan.
***Faculty of Science and Engineering, Chuo University, Bunkyo-ku, Tokyo, Japan.

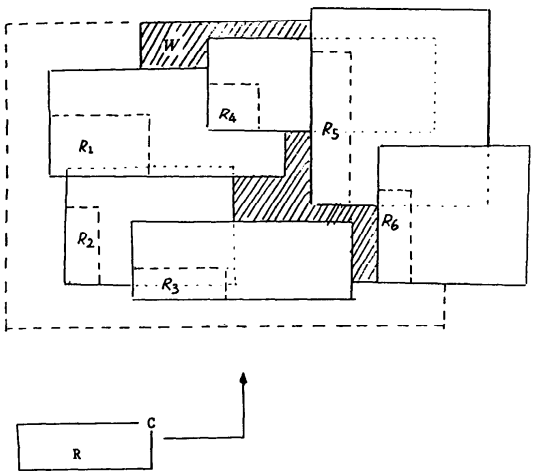

Fig. 2 Arrangement of extended rectangles. Each rectangle of R_i ($i=1, 2, \ldots, 6$) is extended to the right by the width the rectangle R and upwards by its height. The board W is shrunk to the right and upwards by the width and height of R, respectively.

is located in the region $P_{\alpha,\beta}=W_{\alpha,\beta}-\bigcup_{i=1}^{n} R_i[\alpha, \beta]$. Therefore, the *admissible region* coincides with $P_{\alpha,\beta}$.

The problem is now reduced to the contour construction problem, since the problem of constructing the admissible region is equivalent to that of constructing the contour of the union of extended rectangles $\bigcup_{i=1}^{n} R_i[\alpha, \beta]$. $O(n \log n+k)$ time algorithms are known to construct the contour of a union of n rectangles, where k is the complexity of the contour [2, 5]. Wood [5] presented a research problem of constructing the contour of a set of rectangles in $O(n \log \log n+k)$ time provided that the both vertical and horizontal sorted orders of the vertices are known. That problem is still open in general; however, it is true for the set of extended rectangles $R_i[\alpha, \beta]$ ($i=1, 2, \ldots, n$):

Theorem 2.2 (main theorem)

1. The admissible region $P_{\alpha,\beta}$ is constructed in $O(n \log \log n)$ time and $O(n)$ space, updating $O(n)$ space dynamic data. The Admissible region is obtained as a disjoint union of connected orthogonal regions, which are represented by the edge lists of their boundary orthogonal polygons.

2. $O(\log n)$ time is needed to update the data when we remove a rectangle from the board.

We maintain the following three lists as dynamic data:

(1) List of all horizontal edges of rectangles R_i ($i=1, 2, \ldots, n$) sorted with respect to y-values,

(2) List of all left vertical edges of rectangles sorted with respect to x-values, and

(3) List of all right vertical edges of rectangles sorted with respect to x-values.

Without loss of generality, we assume that each pair of the x-values (resp. y-values) has two different values,

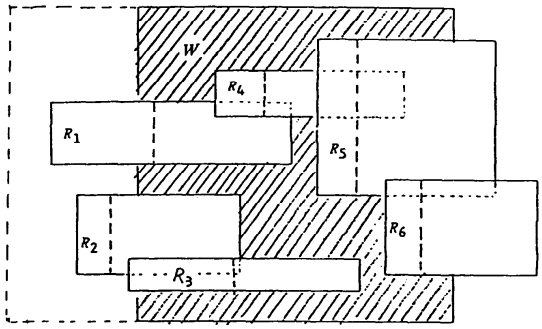

Fig. 3 Arrangement of horizontally extended rectangles. Each rectangle R_i ($i=1, 2, \ldots, 6$) is extended to the right by the width of R. The board W is shrunk to the right by the same amount.

since we can apply a perturbation method to a singular case.

If we consider a special case where W is the entire plane $\mathbf{R}^2$, we obtain the following corollary from the theorem:

Corollary 2.3 *The contour of union of the extended rectangles is constructed in $O(n \log \log n)$ time from the three lists above.*

Before dealing with the admissible region $P_{\alpha,\beta}$, we consider the arrangement consisting of horizontally extended rectangles $R_i[\alpha]=R_i[\alpha, 0]$ ($i=1, 2, \ldots, n$) and a horizontally contracted board $W_\alpha=[l_w+\alpha, r_w] \times [b_w, t_w]$ (see Fig. 3).

The following observation suggests the importance of studying the arrangement constructed from horizontally extended rectangles in order to solve our problem.

Observation 2.4 *Rect(α, β) can be placed in $P_0=W-\bigcup_{i=1}^{n} R_i$ if and only if the vertical segment of length β can be placed in $P_\alpha=W_\alpha-\bigcup_{i=1}^{n} R_i[\alpha]$.*

By adding vertical cuts to an orthogonal region, we can decompose the region into a set of rectangular vertical strips. This decomposition is called a (vertical) rectangulation if we decompose the region into a minimum number of strips (Fig. 4). The rectangulation is uniquely determined.

Suppose that we have obtained the rectangulation of the orthogonal region P into vertical strips $V_1, V_2, \ldots, V_k$. Without loss of generality, we can assume that each pair of vertical (resp. horizontal) edges of P has two different x-values (resp. y-values). Then each side of a vertical strip is bordered by at most an edge of P. We consider a directed graph $G(P)$ called the rectangulation graph associated with the rectangulation (Fig. 5). The node set of $G(P)$ is $\{a_1, a_2, \ldots, a_k\}$, where a_i corresponds to the vertical strip V_i. Two nodes, a_i and a_j, are connected by a directed arc $\langle a_i, a_j \rangle$ if and only if V_i and V_j are adjacent in P sharing a vertical cut such that V_i is located to the left of V_j. Each node has four edge fields in which we store the names of edges of P bordering the corresponding four sides of the strip. In Fig. 5, an edge of P_α is denoted by using the name of a rec-

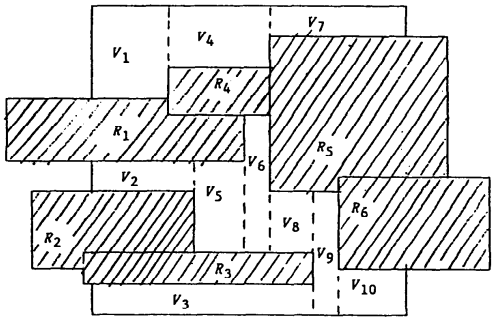

Fig. 4 Vertical rectangulation. The intersection of the horizontally shrunken board W with the outer region of the union of the horizontally extended rectangles is decomposed into vertical strips.

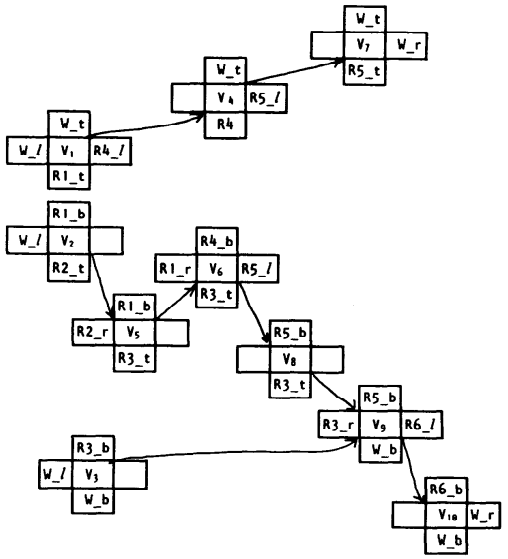

Fig. 5 Rectangulation graph corresponding to Fig. 4. A node contains the name of a strip and the edge of the (horizontally extended) rectangle (or the frame of the board W) bounding each side of it. A pair of nodes corresponding to adjacent strips are connected by an arc.

tangle or the board W. The node a_6 corresponding to V_6 is bordered by *R1_r* (right edge of R_1), *R4_b* (bottom edge of R_4), *R5_l* (left edge of R_5), and *R3_t* (top edge of R_3).

In the following two sections, we design an $O(n \log \log n)$ algorithm to construct the rectangulation graph $G(P_\alpha)$ (mainly, we deal with the case in which $W=\mathbf{R}^2$.) In the final section, we demonstrate the construction of the admissible region $P_{\alpha,\beta}$ from $G(P_\alpha)$ in $O(n)$ time.

3. FIFO Arrangement of Rectangles

Let $\Lambda=(R_1, R_2, \ldots, R_n)$ be an arrangement of rectangles, where $R_i=[l_i, r_i]\times[\beta_i, t_i]$. Λ is called a *FIFO arrangement* if it satisfies the following condition:

Condition 3.1 (FIFO property) *If R_i and R_j overlap and $l_i \le l_j$, then $r_i \le r_j$.*

If we place the rectangles of Λ on a plane in order of the x-values of the left edges of rectangles, eliminating hidden parts of previous placed rectangles, we get an orthogonal subdivision $S(\Lambda)$ of $Q(\Lambda):=\bigcup_{i=1}^{n} R_i$. $S(\Lambda)$ is called the *visible skeleton* of Λ (Fig. 6). We consider $S(\Lambda)$ to be a planar graph whose vertex is an intersection point (of edges) or a corner of a rectangle. If Λ is a FIFO arrangement, each horizontal edge of a rectangle contains at most three vertices of $S(\Lambda)$. Hence, the total number of vertices in $S(\Lambda)$ is at most $6n$. An upper horizontal edge of a rectangle bounds at most two regions (open connected components) of $\mathbf{R}^2-S(\Lambda)$, and each region is bounded by at least an upper horizontal edge of a rectangle because of the FIFO property. Hence, there are at most $2n$ regions in $\mathbf{R}^2-S(\Lambda)$, and it follows from Euler's relation for planar graphs that $S(\Lambda)$ contains at most $8n-1$ edge segments. We have obtained the following lemma:

Lemma 3.2 *For a FIFO arrangement Λ, the complexity of the visible skeleton $S(\Lambda)$ is $O(n)$.*

We explain the relation of a FIFO arrangement to our placement problem. Let Γ be the set of horizontally extended rectangles $R_i(\alpha)=[l_i, r_i+\alpha]\times[b_i, t_i]$ $(i=1, 2, \ldots, n)$. Suppose an extended rectangle $R_i(\alpha)$ overlaps with $R_j(\alpha)$ so that $l_i \le l_j$ and $r_i+\alpha \ge r_j-\alpha$. Then, immediately, R_i overlaps with R_j, which is a contradiction. We then have the following proposition:

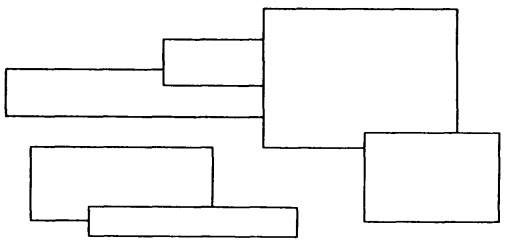

Fig. 6 Visible skeleton associated with a FIFO arrangement.

Proposition 3.3 *Γ is a FIFO arrangement.*

From now on, we deal with a FIFO arrangement Λ. Moreover, we assume that both the sorted order of the set of y-values of horizontal edges and that of the set of x-values of vertical edges are known. For our dynamic placement problem, we can construct these sorted lists for the arrangement of horizontally extended rectangles in $O(n)$ time from our dynamic data. Let $G(P)$ be the rectangulation graph of the complement space $P=\mathbf{R}^2-Q(\Lambda)$ of $Q(\Lambda)$. Then,

Theorem 3.4 *$G(P)$ is constructed in $O(n \log \log n)$ time and $O(n)$ space.*

We give a proof of this therem in the following section.

4. Plane Sweep and $O(\log \log n)$ Priority Queue

Let us recall the $O(\log \log n)$ priority queue of Emde Boas [3, 6, 7], which is an efficient data structure for maintaining ordered lists dynamically. Let U be an ordered set of size n. An element of U is either active or inactive. The set M consisting of all active elements is updated dynamically. We define the operations FIND, SPLIT, and UNION:

- ***FIND(u)***: Return the smallest active element larger than the element u of U.
- ***SPLIT(u)***: Activate the element u of U.
- ***UNION(m)***: If m is an active element, inactivate m (else nop).

Theorem 4.1 [6, 7] *An $O(n)$ mixed sequence of FIND, SPLIT, UNION is executed in $O(n \log \log n)$ time, using a data structure of $O(n)$ space and $O(n \log \log n)$ preprocessing.*

Further, we use the following two operations:

- ***PREDECESSOR(u)***: Return the largest active element smaller than u.
- ***MEMBER(u)***: Return 1 if u is an active element, else return 0.

The operations PREDECESSOR and MEMBER need $O(\log \log n)$ time and $O(1)$ time, respectively. The details of the data structure are explained by Mehlhorn [3].

We design a plane sweep algorithm to construct the rectangulation graph $G(P)$. Let $X(vert)$ be the sorted set of the x-values of the vertical edges of rectangles. We sweep the plane with vertical lines through the members of $X(vert)$ from left to right, maintaining the list of horizontal edges of the visible skeleton $S(\Lambda)$ intersecting with the current weep line.

Definition 4.2 $\mathscr{H}(\theta)$ denotes the set of all horizontal edges of horizontally extended rectangles whose part segments appearing in the visible skeleton $S(\Lambda)$ intersect the sweep line θ at their interior points or left endpoints.

$\mathscr{H}(\theta)$ is regarded as the set of active elements at the sweep line θ in the set $\mathscr{H}$ consisting of all horizontal edges of the horizontally extended rectangles sorted with respect to their y-values.

We adopt the $O(\log \log n)$ priority queue in order to maintain $\mathscr{H}(\theta)$. We also consider a label $L(h)$ for each element h of $\mathscr{H}(\theta)$. Intuitively, we assign to $L(h)$ the name of the rectangle that covers the region infinitesimally higher than h at the current sweep line θ. $L(h)=0$ means that the region belongs to the complement region $\mathbf{R}^2-Q(\Lambda)$. We reset $L(h)$ to -1 whenever UNION(h) is done. Further, we equip a queue B_i called the *saving box* associated with each rectangle R_i ($i=1, 2, \ldots, n$). As the algorithm sweeps to the left endpoint of an upper horizontal edge h, h is inserted into B_i if the region infinitesimally higher than the left endpoint of h belongs to R_{ui} in $S(\Lambda)$. B_0 is associated with the complementary region.

At each sweep line, one of the operations INSERT and DELETE below is done:

1. *If* the sweep line arrives at the *left* edge of a rectangle R_k, *then* *INSERT*(R_k).
2. *If* the sweep line arrives at the *right* edge of a rectangle R_k, *then* *DELETE*(R_k).

Procedure ***INSERT***(R_k);
begin
1: u:=the lower edge of R_k;
2: v:=the upper edge of R_k;
3: p:=L(PREDECESSOR(v)); {The region above v is covered by R_p}
4: insert v in B_p; {v is stored in the saving box associated with R_p}
5: COMPLETE_STRIP(PREDECESSOR(u)); {call a subroutine}
6: w:=FIND(u);
7: ***while*** $w<v$ ***do*** {climb up the current list}
begin
8: COMPLETE_STRIP(w);
9: UNION(w);
10: w:=FIND(w);
11: ***end;***
12: SPLIT(u);
13: SPLIT(v);
14: L(u):=k;
15: L(v):=p;
16: MAKE_STRIP(PREDECESSOR(u)); {call a subroutine}
17: MAKE_STRIP(v);
end;

Procedure ***DELETE***(R_k);
begin
1: u:=the lower edge of R_k;
2: v:=the upper edge of R_k;
3: COMPLETE_STRIP(PREDECESSOR(u));
4: COMPLETE_STRIP(v);
5: UNION(u);
6: UNION(v);
7: MAKE_STRIP(PREDECESSOR(u));
repeat {draw from the saving box}
8: delete w from B_k;
9: ***if*** MEMBER(w)=1 ***then***
begin
10: L(w):=0;
11: MAKE_STRIP(w);
end;
12: ***until*** B_k is vacant;
end;

Procedure ***MAKE_STRIP***(z);
begin
1: ***if*** $L(z)=0$ ***then***
begin
2: Create a strip between the horizontal edges z and FIND(z) such that its left side is bounded by the current sweep line;
3: Create a node of the rectangulation graph corresponding to the strip;
4: ***if*** the strip is adjacent to a strip at the current

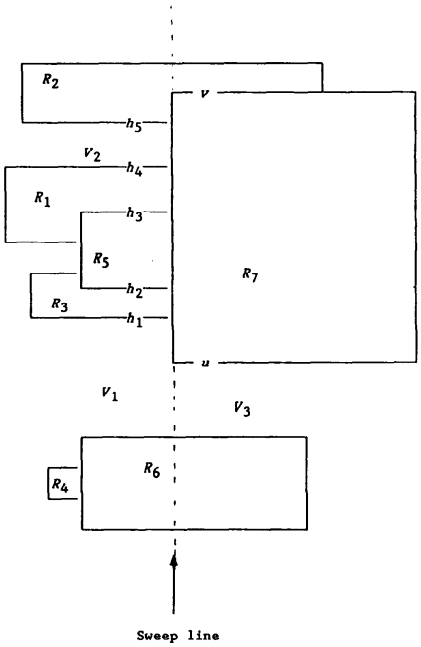

Fig. 7 Sweep algorithm. INSERT R_7.

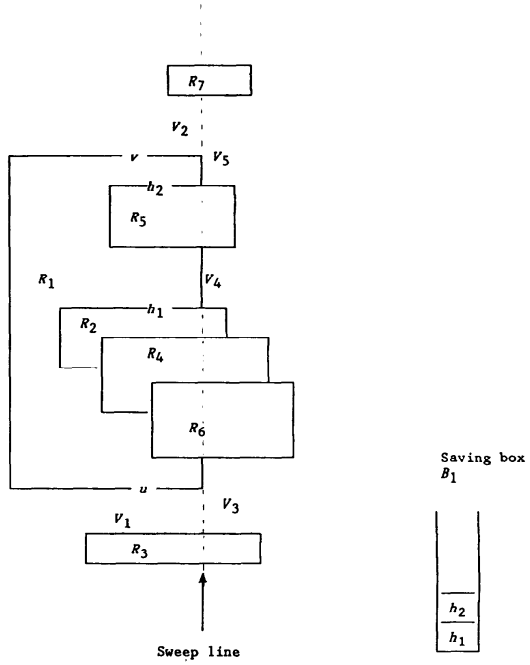

Fig. 8 Sweep algorithm. DELETE R_1.

sweep line ***then***
begin
5. Create the corresponding arc in the rectangulation graph;
end;
end;
end;

Procedure ***COMPLETE_STRIP***(z);
begin
1: ***if*** $L(z)=0$ ***then***
begin
2: Close the right side of the strip between z and *FIND*(z) with the current sweep line;
end;
end;

We illustrate the procedure INSERT in Fig. 7, where the sweep line comes to the left edge of R_7. First, we store the upper edge v of R_7 in the saving box B_2 corresponding to R_2. next, we FIND the lower edge u of the rectangle R_7 and complete the strip V_1. Then, we climb up the current list to the upper edge v, removing the horizontal edges $h_1, h_2, \ldots, h_5$ from the list, and completing V_2. Lastly, we insert u and v into the current list, and create a vertical strip V_3 adjacent to V_1. We illustrate the procedure DELETE in Fig. 8, where the sweep line comes to the right edge of R_1. The saving box B_1 contains h_1 and h_2. Thanks to the FIFO property, only those regions whose lower edges are contained in the saving box are created (except for the region contiguous to the bottom-right corner of R_1) when we delete R_1. First, we FIND the lower edge u and upper edge v of R_1 to complete the strips V_1 and V_2. Next, we update the current list, removing u and v from it. Then, we create V_3 adjacent to V_1. Finally, referring to the saving box B_1, we create strips V_4 and V_5 whose lower edges are h_1 and h_2 respectively. V_5 is adjacent to V_2.

We remark that if we do not use the saving box, we can climb up the list naively in four steps from u to h_1 in this example; however, such a strategy increases the complexity to $O(n^2)$ in the worst case.

We now analyze the complexity of the algorithm.

Proposition 4.3 *During the sweep algorithm, the number of iterations is $O(n)$ for each of the operations FIND, UNION, SPLIT, PREDECESSOR, MEMBER, MAKE_STRIP, and COMPLETE_STRIP.*

Proof When we execute INSERT(R_k), the number of iterations in the while-loop is equal to the number of horizontal edges touching the left horizontal edge of R_k in the visible skeleton $S(\Lambda)$. Therefore, the total number of iterations in all the INSERT procedures is not more than the number of vertices of $S(\Lambda)$, which is known to be $O(n)$ from Lemma 3.2. When we execute DELETE(R_k), the number of iterations in the repeat-loop is equal to the number of elements stored in the saving box B_k. Since an (upper) horizontal edge of a rectangle is contained in at most one saving box, the total size of the saving boxes is less than n. Hence, the total number of iterations in all DELETE procedures is $O(n)$. This proves the proposition.

COMPLETE_STRIP is done in $O(1)$ time.

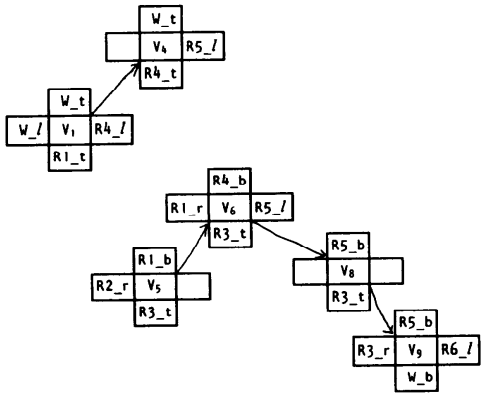

Fig. 9 Graphical representation of the admissible region.

MAKE_STRIP needs $O(\log \log n)$ time, excluding the time spent searching for adjacent strips in step 4. We can design the procedures INSERT and DELETE so that they pass at most two candidates of adjacent strips to the subroutine MAKE_STRIP, since two strips are adjacent at a sweep line only if the boundaries of both strips share a vertex of the inserted (or deleted) rectangle. Thus, a MAKE_STRIP can be executed in $O(\log \log n)$ time in our algorithm. We have shown that each single operation in the procedures INSERT and DELETE needs at most $O(\log \log n)$ time. Hence, it follows from Proposition 4.3 that the algorithm constructs the graph $G(P)$ as a planar graph in $O(n \log \log n)$ time, using $O(n)$ space. The proof of Theorem 3.4 is complete.

Note: If we use a dynamic balanced tree [3] instead of an $O(\log \log n)$ priority queue to maintain $\mathscr{H}(\theta)$, we obtain a simple $O(n \log n)$ algorithm.

5. Construction of Admissible Region

Proposition 3.3 and theorem 3.4 ensure that we can obtain the graph $G(P)$ for $P=\mathbf{R}^2-\bigcup_{i=1}^{n} R_i(\alpha)$ in $O(n \log \log n)$ time. Our target is the admissible region $P_{\alpha,\beta}$. We obtain $G(P_{\alpha,\beta})$ (Fig. 9) from $G(P)$ in $O(n)$ time as follows:

1. For each node a of $G(P)$, compute the intersection of the strip $V(a)$ stored in a with the horizontally contracted board W_α.
2. $V(a):=V(a)\cap W_\alpha$ (change the information of the strip stored in a).
3. Erase each node a for which the height of $V(a)$ is less than β.

Finally, we show the construction of the amissible region from the rectangulation graph. $G(P_{\alpha,\beta})$ is decomposed into its connected components in linear time by using depth-first search, and each connected component corresponds to a connected component of the admissible region. Since $G(P_{\alpha,\beta})$ is a planar graph, it subdivides the plane into cells (i.e. connected regions). Inside each cell, we walk along each connected component of the boundary of the cell, forgetting the direction of arcs. At each visited node, we walk around it inside the cell from the current adjacent side (associated with the current arc) to the next adjacent side, reporting the contents of edge fields. Thus, we obtain a list of edges of rectangles. Abridging it by removing immediate repetitions, we obtain the edge list is a connected component of the boundary of the corresponding connected component of the admissible region $P_{\alpha,\beta}$. In Fig. 9, there is only one cell (external cell). If we walk around the connected component of the graph corresponding to the union of strips V_5, V_6, V_8, and V_9, we get the edge list (*R2_r, R1_b, R1_r, R4_b, R5_l, R5_b, R6_l, W_b, R3_r, R3_t*) of the corresponding connected component of the admissible region (see Fig. 2). This process takes $O(n)$ time. Hence, we have obtained Theorem 2.2.

Acknowledgement

The authors express their gratitude to Dr. N. Kubo, Sharp Corp., for suggesting the problem considered here in two-dimensional memory allocation for window systems, from which this research originated.

References

1. BAKER, B. S., COFFMAN, E. G. and RIVEST, R. L. Orthogonal packing in two dimensions, *SIAM J. Comput.*, **9**-4 (1980), 846-855.
2. GUTING, R. H. An Optimal Algorithm for Iso-Oriented Rectangles, *J. Algorithms* 5 (1984), 303-326.
3. MEHLHORN, K. Data Structures and Algorithms, **1**, *Springer Verlag*, 1984.
4. PREPARATA, F. P. and SHAMOS, M. I. Computational Geometry—An Introduction, *Springer Verlag*, 1985, 1988 (second ed.).
5. WOOD, D. The Contour Problem for Rectilinear Polygons, *Information Processing Letters*, **19** (1984), 229-236.
6. VAN EMDE BOAS, P. Preserving Order in a Forest in Less than Logarithmic Time and Linear Space, *Information Processing Letters*, **6** (1977), 80-82.
7. VAN EMDE BOAS, P. and ZIJSTRA, E. Design and Implementation of an Efficient Priority Queue, *Math. Syst. Theory*, **10** (1977), 99-127.

(Received May 25, 1989; revised January 30, 1990)