アルゴリズム 5−19
（1989. 1. 26）

交差のない長方形配置の動的アルゴリズム

徳山豪[1]、 浅野孝夫[2]、 築山修治[3]

[1]日本アイ・ビー・エム東京基礎研究所、 [2]上智大学理工学部、 [3]中央大学理工学部

平面上に配置された長方形の集合の交差図形の研究は、計算幾何学の基本的なテーマである。 本研究では、ＦＩＦＯ交差と呼ばれる特殊な交差図形を定義し、ＦＩＦＯ交差の外形図や補領域の分割が、O(n log log n) の手間で得られることを示す。 この結果を用いて、効率的な「交差の無い長方形配置問題」の動的アルゴリズムが得られる。 即ち、n個の長方形の穴が開いている直交領域に対して、与えられた長方形がその中に配置できるかどうか判断して、可能な配置を全て求める問題である。新しく長方形が配置されると、それは直交領域の新しい穴としてみなされる。ここで与えるアルゴリズムは、O(n)の記憶領域を用いて、一回の長方形配置探索にO(n log log n)の計算時間を要する。データの更新は、直交領域の一個の穴の増加、減少に対して O(log n)で行える。

A Dynamic Algorithm for Placing Rectangles without Intersection

Takeshi Tokuyama[1], Takao Asano[2], and Shuji Tsukiyama[3]

[1] IBM Research, Tokyo Research Laboratory, 5-19, Sanbancho, Chiyoda-ku, Tokyo, Japan

[2] Faculty of Science and Technology, Sophia University, Chiyoda-ku, Tokyo, Japan

[3] Faculty of Science and Engineering, Chuo University, Bunkyo-ku, Tokyo, Japan

Construction of the contour of the union of rectangles is an important problem in computational geometry. In this paper, we define a special class of arrangements of rectangles, called *FIFO arrangements*, whose contour is constructed in $O(n \log\log n)$ time. Using the result, we solve the following dynamic allocation problem of rectangles efficiently: given a rectangle and an orthogonal region with n non-intersecting rectangular holes, find a possible placement of the rectangle in the region and create a new hole. We present an algorithm that takes $O(n \log\log n)$ time and $O(n)$ space; this algorithm finds not only one but all possible placements.

Introduction

In this paper, we deal with the problem of finding an allocation of a set of rectangles without overlapping (we consider only vertical-horizontal rectangles). This problem has applications to 2-dimensional memory allocation, VLSI design, scheduling, and graphics. Let us imagine that we place a set of rectangular chips on a board.

Problem 1 *Suppose a rectangular board W and a set of rectangles $\{Rect(a_1, b_1), Rect(a_2, b_2), \ldots, Rect(a_n, b_n)\}$ are given. Find an allocation of the rectangles on W without overlapping each other, where $Rect(a, b)$ is a rectangle of width a and height b*

Problem 1 is a kind of 2-dimensional bin packing problem that is hard to be solved [B-C-R]. We can find an approximate solution of this problem by placing all rectangles in a "reasonable" way and then remove overlapping rectangles. Bentley and Wood [B-W], and McCreight [M] designed efficient algorithms to list intersections of rectangles that have already been placed on a plane. However, to construct a complete solution, we must place the rectangles that have been removed back on the board without overlapping the rectangles already there. Therefore, we would like to consider the following problem of incrementing placement:

Problem 2 *Suppose n rectangles $R_1, R_2, \ldots, R_n$ have already been placed on W without overlapping. Then decide whether the next rectangle $Rect(\alpha, \beta)$ can be placed on the orthogonal region $W - R_1 \bigcup R_2 \bigcup \ldots \bigcup R_n$. If so, find all possible placements.*

Figure 1 and Figure 2 illustrate the problem. The set of all possible placements of the top-right corner of the inserted rectangle is called an *admissible region.* We are concerned with dynamic algorithms for problem 2, where we process a series of insertions and deletions updating suitably preprocessed data. An admissible region is constructed in $O(n \log n)$ time by using the algorithms of Gutting or Wood [G] [W]. This bound is optimal for the static problem. However, we can improve this upper bound to $O(n \log \log n)$ for the dynamic problem. Our result is as follows:

Theorem 1 *There exists a dynamic algorithm that constructs the admissible region for the insertion of a rectangle in $O(n \log \log n)$ time, where n is the number of rectangles already placed on the board. We maintain a data structure that needs $O(n)$ space and $O(\log n)$ updating time when we insert or delete a rectangle.*

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 $b < t$. We consider the extended rectangle $R[\alpha, \beta] = [l, r + \alpha] \times [b, t + \beta]$. Further, we shrink 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 (Figure 3.)

1. $Rect(\alpha, \beta)$ can be placed in $P_0 = W - \bigcup^{n} R_i$ if and only if $P_{\alpha,\beta} = W_{\alpha,\beta} - \bigcup^{n} R_i[\alpha, \beta]$ is non-empty.
2. We can place $Rect(\alpha, \beta)$ such that its top-right corner is located on any point of $P_{\alpha,\beta}$.

Therefore, it suffices to compute the *admissible region* $P_{\alpha,\beta}$ efficiently. An admissible region is not connected in general.

The arrangement constructed from $R_i[\alpha, \beta]$ $(i = 1,2,\ldots,n)$ is not easy enough to handle. Thus, we consider the easier arrangement consisting of horizontally extended rectangles $R_i[\alpha] = R[\alpha, 0]$ $(i = 1,2,\ldots,n)$ and a horizontally contracted board $W_\alpha = [l_w + \alpha, r_w] \times [b_w, t_w]$. (See Figure 4.)

The following lemma ensures that it suffices to study the arrangement constructed from $R_i[\alpha]$ $(i = 1,2,..,n)$ in order to solve our problem.

Lemma 2.2 *$Rect(\alpha, \beta)$ 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 some 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 (Figure 5.) The vertical rectangulation is uniquely determined. The set of all maximal vertical segments that can be placed in an orthogonal region is obtained from its rectangulation. Therefore, we shall deal with the rectangulation of P_α.

Suppose that we obtain the rectangulation of a orthogonal region P into vertical strips $V_1, V_2, \dots, V_k$. Without loss of generality, we can assume that each pair of vertical edges of P has two different x-values. We consider a directed graph $G(P)$ associated with the rectangulation (Figure 6.) The vertex set of $G(P)$ is $\{a_1, a_2, \dots, a_k\}$, where a_i corresponds to the vertical strip V_i. Each node keeps information about the four edges of the corresponding strip. Two nodes, a_i and a_j, are connected by a direct edge $[a_i, a_j]$ if and only if V_i and V_j are adjacent in P_α, sharing a vertical cut such that V_i is located to the left of V_j. $G(P)$ has the following properties:

Facts 2.3
1. $G(P)$ is a planar graph.
2. Each node has at most two outgoing edges and two incoming edges.

We have the following lemma:

Lemma 2.4 *If we can obtain the graph $G(P_\alpha)$ in $F(n)$ time, we can compute the admissible region in $F(n)+O(k)$ time, where k is the complexity of the region P_α.*

In fact, the complexity of P_α is $O(n)$; thus, it suffices to construct the graph $G(P_\alpha)$ efficiently. Since we are considering a dynamic placement problem, we can assume that we know the sorted lists of vertical edges (resp. horizontal edges) of previously placed rectangles with respect to x-values (resp. y-values.) As is well known, these lists are updated in $O(\log n)$ time. The arrangement constructed from $R_i[\alpha]$ (i = 1,2,...,n) has a nice property that we call the *FIFO property*. In the following section, we study arrangements of rectangles with the FIFO property.

FIFO arrangement of rectangles

Let $\Lambda = (R_1, R_2, \dots, R_n)$ be a sequence of rectangles, where $R_i = [l_i, r_i] \times [b_i, t_i]$. The contour $C(\Lambda)$ of the orthogonal region $Q(\Lambda) := \bigcup^n R_i$ is a (non connected) orthogonal polygon. Further, placing the rectangles of Λ on a plane in order of the index and eliminating hidden parts of previous placed rectangles, we get an orthogonal subdivision $S(\Lambda)$ of $Q(\Lambda)$ constructed from the visible edges of rectangles.

It is a major problem in computational geometry to construct the contour $C(\Lambda)$ and the subdivision $S(\Lambda)$ efficiently. For a general Λ, $O(k + n\log n)$ time algorithms for constructing $C(\Lambda)$ and an $O(K\log n + n\log n\log\log n)$ time algorithm for constructing $S(\Lambda)$ are known, where k and K are the complexity of $C(\Lambda)$ and $S(\Lambda)$ respectively [G], [W], [B]. However, for a special kind of sequence shown below, we can design a more efficient algorithm.

We say Λ is *regular* if the sequence $(l_1, l_2, \dots, l_n)$ is a non-decreasing sequence. A regular sequence Λ is called a *FIFO sequence* if it satisfies the following condition:

Condition 3.1 (See Figure 7) *If R_i and R_j overlap for $i<j$, then $r_i \le r_j$.*

Fact 3.2 *For a FIFO sequence Λ, the complexity of the subdivision $S(\Lambda)$ is at most $7n$.*

We explain the relation of a FIFO sequence to the previous section. We sort the horizontally extended rectangles $R_i(\alpha)$ $(i = 1,2,..,n)$ to obtain a regular sequence Γ. This process takes O(n) time, since we know the sorted list of vertical edges. We then immediately have the following:

Proposition 3.3 *Γ is a FIFO sequence.*

The proof is trivial.

From now on, we assume that Λ is a FIFO sequence. Moreover, we assume that both the sorted order of the sets of y-values of horizontal edges and that of of x-values of vertical edges are known from some oracle. We may assume that there is no pair of horizontal (resp. vertical) edges sharing a y-value (resp. x-value). Our results are as follows:

Theorem 3.4
1. $C(\Lambda)$ is constructed in $O(n \log\log n)$ time.
2. $S(\Lambda)$ is constructed in $O(n \log\log n)$ time.

Theorem 3.5
1. The complement space $P = \mathbb{R}^2 - Q(\Lambda)$ is decomposed into vertical strips in $O(n \log\log n)$ time, where $\mathbb{R}^2$ is the total plane.
2. $G(P)$ is constructed in $O(n \log\log n)$ time. We use $O(n)$ space for our algorithm.

We shall give the outline of a proof of theorem 3.5. The proof of theorem 3.4 is omitted in this version.

Plane Sweep and Set Union Structure

First, let us recall the split-union algorithm of [V-M-N]. Let S be a subset of a linearly ordered ground set U. We distinguish the elements of S from those of $U - S$ by marking each element of S . We do the following three operations -- FIND, SPLIT, and UNION.

1. FIND(u): Find the smallest marked element that is larger than or equal to the element u of U.
2. SPLIT(u): Mark the element u of U.
3. UNION(s): If s is an element of S, unmark s. (Else nop.)

Theorem 4.1 [V-K-N], [V] *An $O(n)$ mixed sequence of FIND, SPLIT, and UNION is done in $O(n \log\log n)$ time, using a data structure of $O(n)$ space and $O(n \log\log n)$ preprocessing.*

This theorem is based on the theory of priority queue. We omit the details of the data structure.

We prove theorem 3.5 by using plane sweeping. We obtain the graph $G(P)$ of the complement space P as the output of our algorithm. We sweep the plane with the vertical lines through the members of $X(vert)$ from left to right, where $X(vert)$ is the sorted set of the x-values of vertical edges. The set $\mathscr{H}$ of all horizontal edges of the rectangles is our ground set. $\mathscr{H}$ is arranged as an ordered set with respect to the y-value of each element. For convenience' sake, we add $-\infty$ and ∞ to $\mathscr{H}$ as the maximal and minimal elements

At each sweep line θ , we maintain a subset $\mathscr{H}(\theta)$ of $\mathscr{H}$, defined as follows:

Definition 4.2 *$\mathscr{H}(\theta)$ is the set of all horizontal edges whose part segments appearing in the subdivision $S(\Lambda)$ intersect the sweep line θ at the interior points or left endpoints.*

We consider FIND, UNION, and SPLIT for the pair $\mathscr{H} \supset \mathscr{H}(\theta)$. We also maintain the label $L(h)$ for each element h of $\mathscr{H}$. In the algorithm, intuitively, we assign to $L(h)$ the name of the rectangle that covers the point infinitesimally higher than the intersection of h with the current sweep line (L(h) = 0 means that the intersection is vacant.) Initially, L(h) = 0 for all h. We reset L(h) to 0 whenever UNION(h) is done. We also maintain *Queues* Q_i $i = (1,2,...,n)$, which are initially vacant. Q_i stores the horizontal edges of the rectangles that "cover" R_i in $S(\Lambda)$ when they are inserted.

At each sweep line, one of the operations INSERT and DELETE is done:
1. *If* the sweep line corresponds to the *left* edge (h_1, h_2) of a rectangle with h_1 (resp. h_2) as its lower (resp. upper) horizontal edge, *then* $INSERT(h_1, h_2)$.
2. *If* the sweep line corresponds to the *right* edge (h_1, h_2) of a rectangle, *then* $DELETE(h_1, h_2)$.

```
Procedure INSERT (u,v) ;
{ $u < v$ are elements of $\mathscr{H}$. u is the lower edge of $R_k$. }
begin
    1: SPLIT (u);
    2: L(u) := k;
    3: w = PREDECESSOR (u);
    4: if L(w) = 0 then
         begin
           5: MAKENODE (w,u);
           6: COMPLETENODE (w,SUCCESSOR(u));
           7: CONNECT ((w,SUCCESSOR(u)),(w,u));
         end
       else
         begin
           8: PUSH u in $Q_{L(w)}$
```

```
        end;
    9: w = SUCCESSOR(u);
    10. while SUCCESSOR(w) < v do
      begin
        10: if L(w) = 0 then
          begin
            11: COMPLETENODE (w,SUCCESSOR(w))
          end;
        12: UNION (w);
        13: w := SUCCESSOR(w)
      end;
    14: if L(w) = 0 then
      begin
        15: COMPLETENODE (w,SUCCESSOR(w));
        16: MAKENODE (v,SUCCESSOR(w));
        17: CONNECT ((w,SUCCESSOR(w)),(v,SUCCESSOR(w)))
      end;
    18: L(v) = L(w);
    19: SPLIT (v);
    20: UNION (w)
end.
```

```
Procedure DELETE (u,v) ;
{ $u < v$ are elements of $\mathscr{H}$. $u$ is the lower edge of $R_k$. }
begin
    1: if u is marked then
      begin
        2: COMPLETENODE( PREDECESSOR(u), u);
        3: MAKENODE( PREDECESSOR(u), SUCCESSOR(u));
        4: CONNECT ((PREDECESSOR(u),SUCCESSOR(u)),(PREDECESSOR(u),u));
        5: UNION (u)
      end;
      repeat
          6: POP w from $Q_k$;
          7: if w is marked then
            begin
              8: MAKENODE(PREDECESSOR(w), w);
              9: L( PREDECESSOR(w)) = 0
            end;
      10: until $Q_k$ is vacant;
    11: if L(v) = 0 then
      begin
        12: COMPLETENODE(v,SUCCESSOR(v));
        13: MAKENODE(PREDECESSOR (v), SUCCESSOR (v));
        14: CONNECT ((PREDECESSOR(v),SUCCESSOR(v)),(PREDECESSOR(v),v));
        15: L( PREDECESSOR (v)) = 0
      end;
    16: UNION (v)
end.
```

We explain the functions used in the procedures above.

- ***SUCCESSOR (u)*** : Find the smallest marked element that is larger than u.
- ***PREDECESSOR (u)*** : Find the largest marked element that is smaller than u.
- ***PUSH*** : Push an element into a queue.
- ***POP*** : Pop the top element from a queue.
- ***MAKENODE (u,v)*** : Make a node $\alpha(u,v)$ of G(P) corresponding to a strip $V(u,v)$ whose lower and upper horizontal edges are parts of u and v respectively. The left vertical edge of the strip is a part of the current sweep line. The right vertical edge is determined when COMPLETENODE (u, v) is done.
- ***COMPLETENODE (v,w)*** : Determine the right vertical edge of the strip $V(u,v)$ corresponding to the node $\alpha(u,v)$ that has been created by using MAKENODE(u, v). The right vertical edge is a part of the current sweep line.
- ***CONNECT ((u,v), (p,q))*** : Create a directed edge of G(P) from the node $\alpha(u,v)$ to $\alpha(p,q)$.

Lemma 4.3
1. Both SUCCESSOR and PREDECESSOR are done in $O(\log\log n)$ time
2. Each of the operations PULL, PUSH, MAKENODE, COMPLETENODE, and CONNECT is done in $O(1)$ time.

Proposition 4.4
1. During the sweeping of the plane, the number of iterations is $O(n)$ for each of the operations SUCCESSOR, PREDECESSOR, UNION, and SPLIT.
2. The number of iterations is $O(n)$ for each of the operations PUSH, POP, COMPLETENODE, MAKENODE, and CONNECT.

Proof. It suffices to show that the number of iterations of SUCCESSOR is O(n). The total number of SUCCESSOR operations executed in all INSERT operations has the same order as the complexity of $S(\Lambda)$. Hence, it follows Fact 3.2 that the total number of iterations of SUCCESSOR executed in all INSERT operations is $O(n)$. When we DELETE the rectangle R_k, the number of iteration of SUCCESSOR is less than the number of the elements stored in the Queue Q_k. Q_k stores the lower edge h of a rectangle R_i such that h was inserted on the "sheet" corresponding to R_k when the sweep line came to the left vertical edge of R_i. The number of such pairs (k,i) is at most n. This implies that the number of SUCCESSOR operations executed in all DELETE operations is O(n). This proves proposition 4.4.

Hence this plane sweep method constructs the graph G(P) as a planar graph in $O(n\log\log n)$ time, using $O(n)$ space. The proof of theorem 3.5 is now complete.

Construction of admissible region

Proposition 3.3 and theorem 3.5 ensure us that we can obtained the graph $G(P)$ for $P = \mathbb{R}^2 - \bigcup^{n} R_i(\alpha)$ in $O(n\log\log n)$ time. Our target is the admissible region $P_{\alpha,\beta}$. We obtain $G(P_{\alpha,\beta})$ from $G(P)$ in $O(n)$ time as follows:
1. For each node a of G(P), consider the intersection of W_α and the strip $V(a)$ stored in a.
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.
4. Erase both incoming edges and outgoing edges of erased nodes.

We can construct the admissible region $P_{\alpha,\beta}$ from $G = G(P_{\alpha,\beta})$ by using graph walk. We omit the details in this version.

Acknowledgement: The authors express their gratitude to Dr. N. Kubo, Sharp Corp., for his suggestion of the problem considered here in 2-dimensional memory allocation, from which this research is originated.

References

[B-C-R] B. S. Baker, E.G.Coffman, and R.L.Rivest, "Orthogonal packing in two dimensions," *SIAM J. Comput., 9-4* 1980, pp. 846-855.
[B-W] J. L. Bentley and D. Wood, "An Optimal Worst Case Algorithm for Reporting Intersections of Rectangles, " *IEEE Trans. Comput. C-29,* 1980, pp. 571-577.
[M] E. M. McCreight, "Priority Search Trees, " *SIAM J of Computing 14,* 1985, pp. 257-276.
[G] R. H. Guting, "An Optimal Algorithm for Iso-Oriented Rectangles, " *J.Algorithm 5* 1984, pp. 303-326.
[W] D. Wood, "The Contour Problem for Rectilinear Polygons, " *Information Processing Letters 19* 1984, pp. 229-236.
[B] M. Bern, "Hidden Surface Removal of Rectangles, " *Proc. 4-th ACM Computational Geometry* 1988, pp. 183-192.
[V] P. van Emde Boas "Preserving Order in a Forest in Less Than Logarithmic Time and Linear Space," *Proc. Information Processing Letters,* 1977, pp. 80- 82
[V-K-Z] P.van Emde Boas, B. Kaas, and E. Zijstra, "Design and Implementation of an Efficient Priority Queue," *Math. Syst. Theory 10,* 1977, pp. 99-127

Figures

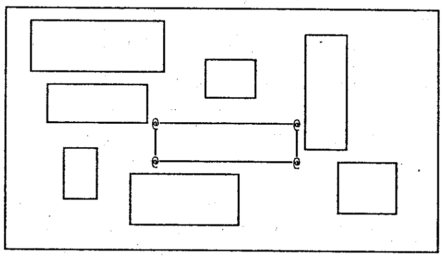

Figure 1. Intersection free allocation of rectangles. We found a placement and updated the orthogonal region.

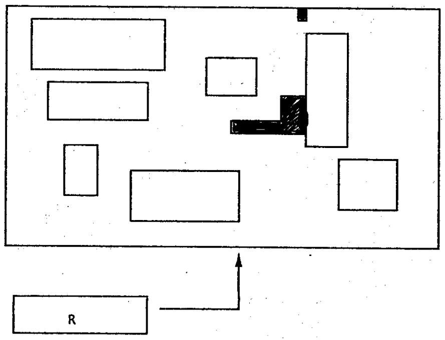

Figure 2. Admissible region. The admissible region is the union of the shaded regions.

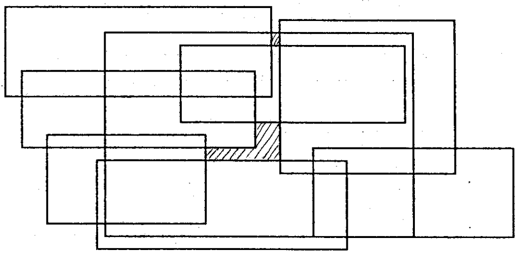

Figure 3. Arrangement of extended rectangles. The admissible region is the union of two regions of the arrangement.

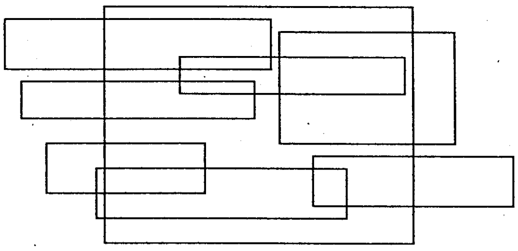

Figure 4. Arrangement of horizontally extended rectangles.

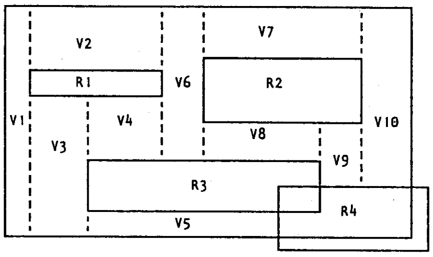

Figure 5. Vertical rectangulation. Decompose an orthogonal region $P = W - R1 \cup R2 \cup R3 \cup R4$ into 10 vertical strips.

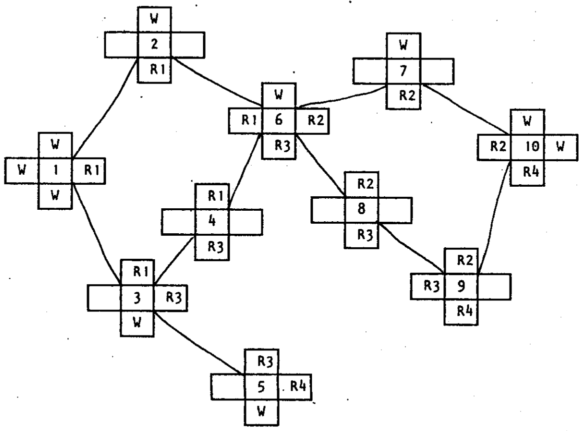

Figure 6. Graph representation of orthogonal region. The graph representation $G(P)$ corresponds to the previous figure. Each node has four edge fields.

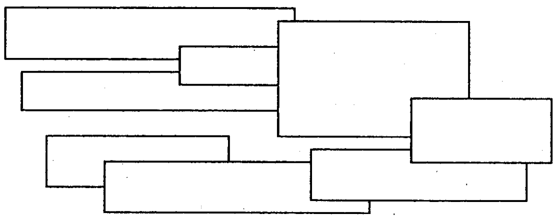

Figure 7. Orthogonal subdivision associated with a FIFO sequence