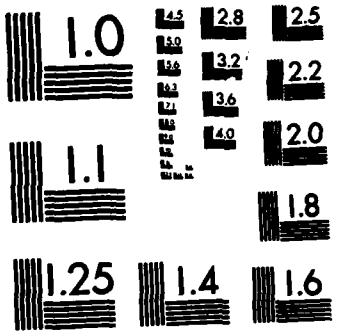
AD-A124 473 AVERAGE CASE ANALYSIS OF AN ADJACENCY MAP SEARCHING TECHNIQUE(U) ILLINOIS UNIV AT URBANA APPLIED COMPUTATION THEORY GROUP G BILARDI DEC 81 ACT-31 1/1

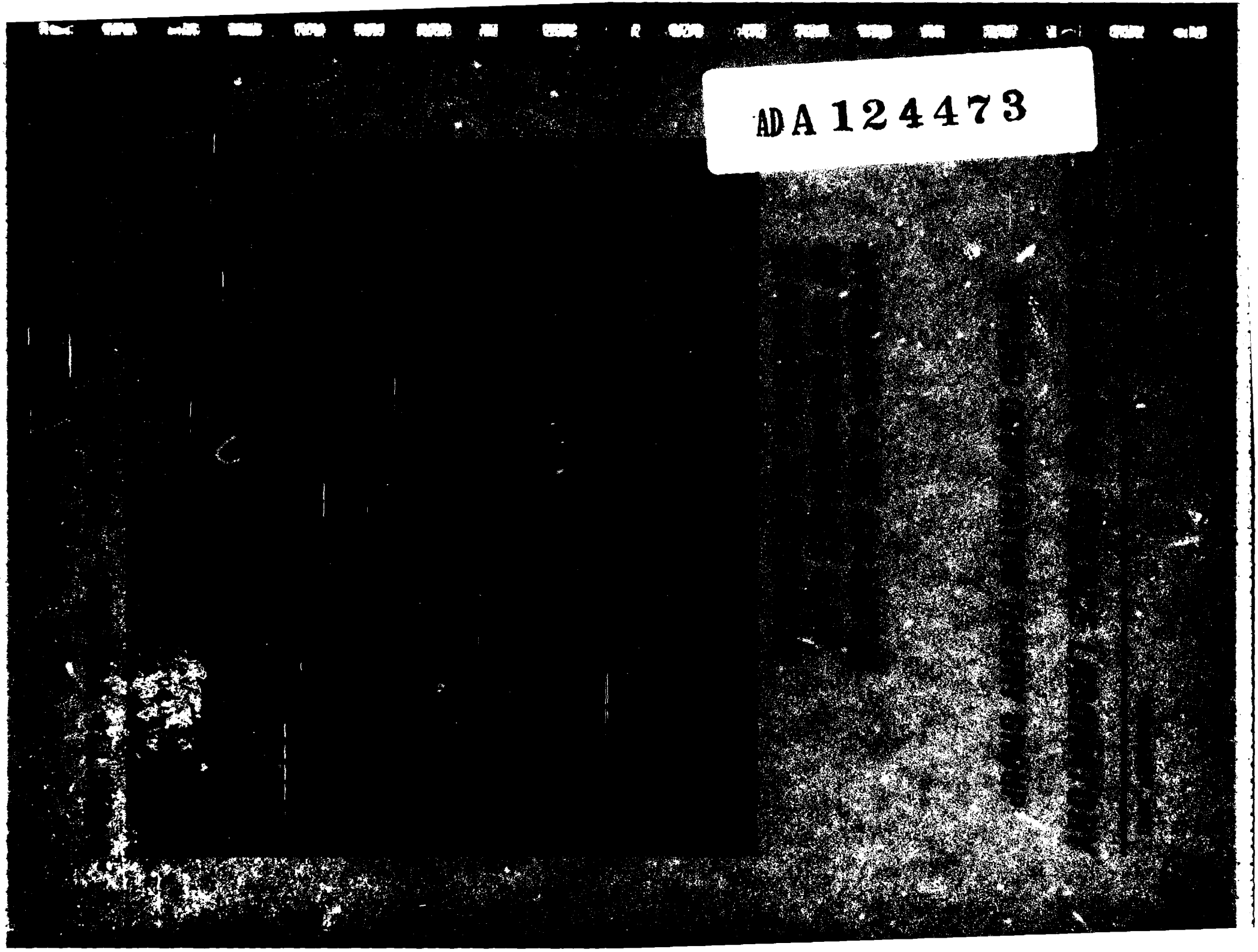
UNCLASSIFIED N00014-79-C-0424 F/G 12/1 NL

END

FILMED

DTIC





SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A124473	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) AVERAGE CASE ANALYSIS OF AN ADJACENCY MAP SEARCHING TECHNIQUE		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER R-928 (ACT-31); UILU-ENG
7. AUTHOR(s) Gianfranco Bilardi		8. CONTRACT OR GRANT NUMBER(s) ECS-81-06939; N00014-79-C-0424
9. PERFORMING ORGANIZATION NAME AND ADDRESS Coordinated Science Laboratory University of Illinois Urbana, IL 61801		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE December 1981
		13. NUMBER OF PAGES 56
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Approved for public release; distribution unlimited.

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

computational geometry, adjacency map, geometric searching, average case analysis, point location.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The adjacency map is a data structure (a tree) used to solve the following problem: given a set of parallel segments in the plane and a point p, find the segments closest to p among those intersected by the straight line through p, perpendicular to the common direction of the segments. The search is performed in the repetitive mode, so that preprocessing is convenient.

The problem considered is a particular case of planar point location for

DD FORM 1 JAN 73 1473

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

which algorithms are known (Lipton-Tarjan, Kirkpatrick), which make use of data structures constructed in time O(nlogn), searched in time O(logn), and stored in space O(n). Though asymptotically optimal, the previous algorithms are not very practical. More practical algorithms have been proposed (Preparata, Preparata-Lipski), which use O(nlogn) space.

In this thesis a modification of these algorithms is presented for the adjacency map, and the worst case analysis is performed.

The technique is easily extensible to general planar graphs. It is conjectured that, under reasonable assumptions on the input distribution, the new algorithm takes expected linear storage.

A probabilistic analysis substantiates such conjecture in the case of the adjacency map, for a wide class of input distributions. In particular, when the segments have independent endpoints, it is shown that the number of nodes in the corresponding adjacency map is about 6 times the number of segments. The results of the analysis have been confirmed by simulation.

AVERAGE CASE ANALYSIS OF AN ADJACENCY MAP SEARCHING TECHNIQUE

BY

GIANFRANCO BILARDI

Laur., Università di Padova, 1978

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1982

Urbana, Illinois

ACKNOWLEDGEMENT

I want to express my gratitude to my thesis advisor, Professor Franco Preparata, who introduced me to Computational Geometry and provided a very helpful guidance for this research.

Several discussions with my friend Sergio Verdu have been very useful.

This work was supported by the National Science Foundation under Grant ECS-81-06939 and by the Joint Services Electronics Program under Contract N00014-79-C-0424. Additional support has been provided by the International Rotary Foundation, which granted me a Fellowship for the first year of graduate study.

Finally, thanks to Mrs. Phyllis Young for the excellent typing of this thesis.

TABLE OF CONTENTS

ABSTRACT

The adjacency map is a data structure (a tree) used to solve the following problem: given a set of parallel segments in the plane and a point p, find the segments closest to p among those intersected by the straight line through p, perpendicular to the common direction of the segments. The search is performed in the repetitive mode, so that preprocessing is convenient.

The problem considered is a particular case of planar point location for which algorithms are known (Lipton-Tarjan, Kirkpatrick), which make use of data structures constructed in time O(nlogn), searched in time O(logn), and stored in space O(n). Though asymptotically optimal, the previous algorithms are not very practical. More practical algorithms have been proposed (Preparata, Preparata-Lipski), which use O(nlogn) space.

In this thesis a modification of these algorithms is presented for the adjacency map, and the worst case analysis is performed.

The technique is easily extensible to general planar graphs. It is conjectured that, under reasonable assumptions on the input distribution, the new algorithm takes expected linear storage.

A probabilistic analysis substantiates such conjecture in the case of the adjacency map, for a wide class of input distributions. In particular, when the segments have independent endpoints, it is shown that the number of nodes in the corresponding adjacency map is about 6 times the number of segments. The results of the analysis have been confirmed by simulation.

1. INTRODUCTION

The adjacency map (AM), a data structure suitable for searching a set of parallel segments in the plane, has been shown to be efficiently applicable to the solution of several problems of planar computational geometry [1]. The AM solves the following problem: given a set $\mathcal{A}$ of parallel segments in the plane and a point p, find the segments closest to p among those intersected by the straight line passing through p and perpendicular to the common direction of the segments. In the sequel we assume the y axis of the Cartesian plane to be parallel to the segments. If through each endpoint w of each member of A we trace a horizontal half-line to the right and one to the left and we make such lines terminate when they meet another segment or continue to the infinity otherwise, we partition the plane into regions. Two of these regions are half planes and all the other are rectangles, possibly unbounded in one or both horizontal directions. Each rectangle is an equivalence class of points of the plane with respect to their horizontal adjacency to vertical segments, and can be called adjacency region.

The foregoing problem is a special case of point location in a planar subdivision and therefore can be handled by one of the general techniques for such problems [2,3,4,5].

By using a method recently developed in [3], Lipski and Preparata [1] have given an efficient algorithm for point location in the adjacency regions. They make use of a static data structure that they call adjacency map, which can be searched in time O(logn), constructed in time O(nlogn), and stored in space O(nlogn), where $n = |\mathcal{A}|$ is the number of segments.

The reason to further consider the problem stems from a theoretical result due to Lipton and Tarjan [4], and Kirkpatrick [5] who showed that O(logn) searching time and O(nlogn) preprocessing time can be achieved with a data structure which only uses O(n) space. Even though the Lipton-Tarjan method is algorithmically extremely complicated (no conclusion to the contrary is available for Kirkpatrick's method), it suggests that a practical algorithm may exist with the same performance.

In this thesis we consider a new algorithm for the AM, which is a conceptually simple modification of the algorithm presented in [1]. While the worst case asymptotic performance is the same as in [1], the average performance is improved. We show that if the segments are independent of each other, under some very weak assumptions, expected linear storage is achieved, and also that except for a presorting operation, the algorithm runs in expected linear time. We also make use of a procedure different from the one used in [1,2] to balance the search tree. The new procedure is simpler and faster. Due to the nature of the procedure for building the search tree, the bound to the depth of the tree is also reduced from $\lceil 5 \log n \rceil$ to $3 \log n + 7$.

The thesis is organized as follows. In Section 2 the search tree is defined together with a recursive procedure for constructing it, and Section 3 describes the balancing of the search tree. The worst case asymptotic analysis is performed in Section 4. Section 5 is devoted to the probabilistic analysis and is the main contribution of this thesis. We begin with the definition of a suitable random model for the input of the algorithm. In this model a segment is defined by two random variables U and V, called generators, which are the (unordered) segment endpoints. A segment is

statistically described by the joint distribution of its generators. We also assume that different segments are statistically independent from each other. We show that, under this assumption, the statistical properties of the algorithm are independent of the first order generator distribution and they are only affected by the correlation properties of the endpoints. Therefore there is no loss of generality in considering uniform generators. The analysis we carry out shows that, for a broad class of generator joint-distributions, the average number of nodes in the search tree is linear in the number of segments. Numerical results are obtained for the particular case of independent endpoints. For this case we get a theoretical upper-bound of 6.07 n for the number S of nodes in the tree. Simulation completely agrees with this upper-bound and gives an average number of nodes close to 5.7 n. These results are very satisfactory, since it can be shown that the search tree must contain at least 3n nodes, and therefore the algorithm performs on the average, within a factor 2 of the absolute lower bound. Moreover the average result for this algorithm is particularly meaningful since - due to the law of large-numbers - the actual values obtained for inputs of reasonably large size (say over 300) are very close to the expected values.

The AM solves several problems related to collections of parallel segments in the plane. Some of these problems are the Nontrivial-Contour, the External-Contour, the Point-Location, and the Route-in-a-Maze, which are defined and discussed in [1]. These and probably several other problems of similar type arise in diverse fields of application such as computer aided design, large-scale-integration, operation research, data base concurrency control, etc. Of course all of the foregoing problems will take advantage of an efficient algorithm for the AM.

However we think that the implications of the present study may be wider. The algorithm that we present is in fact immediately extensible to the problem of point location in a general straight-line planar subdivision and intuition suggests that its performance will be satisfactory in the general case. But a rigorous proof of such a statement is not straightforward and we hope to perform such a task in future work.

2. DEFINITION AND CONSTRUCTION OF THE SEARCH TREE

We recall here from [1] the definition of the AM. The AM is a binary tree $\mathcal{H}$ with two types of nodes having a different typographical representation: "∇", a ∇-node or "horizontal node", is associated with a horizontal line $y = c$ and has the ordinate c as a discriminator; "O", an O-node or segment node, is associated with a straight line segment on the line $x = d$ and has the abscissa d as discriminator.

To search for a given point $p = (x,y)$ corresponds to tracing a path in $\mathcal{H}$ from the root to a leaf. The discriminator is compared against y for ∇-nodes and against x for O-nodes. The path makes a turn to the left when the point coordinate is smaller than the discriminator and to the right otherwise. Along the path the largest abscissa smaller than x, and the smallest abscissa larger than x, initialized to $-\infty$ and $+\infty$ respectively, are recorded and give, when a leaf is reached, the left and the right adjacent segments, with an infinite abscissa corresponding to the case of no adjacent segment.

Several different search trees are possible for the same set of segments. We are particularly interested in bounding the depth of the tree (to bound the query time) and the total number of nodes (to bound the storage). The latter objective is the minimization of the number of nodes in the tree having the same horizontal or vertical discriminator. The former objective requires instead a suitable balancing strategy when constructing the tree.

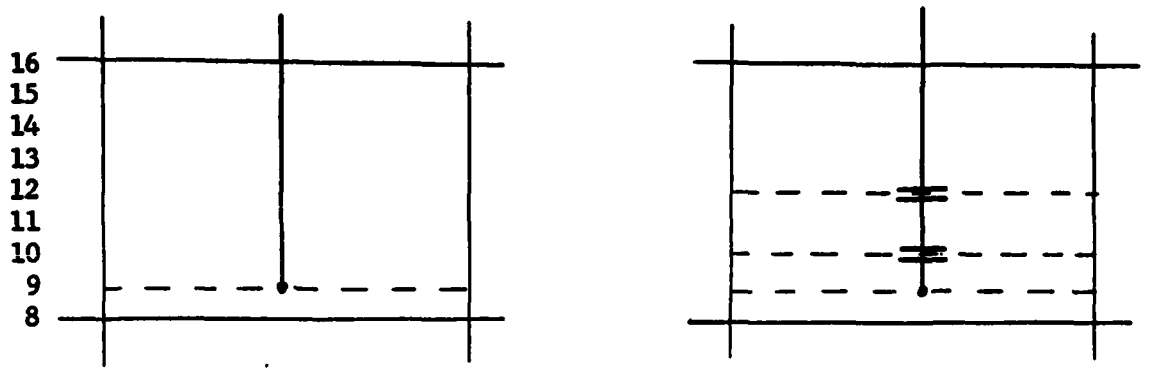
The complete definition of the AM is given by specifying the algorithm that constructs it. To describe both the Lipski-Preparata and the present algorithm let us introduce some definitions. A <u>slab [B,T]</u>, with $B < T$, is the strip of plane contained between the lines $y = B$ and $Y = T$. A vertical

segment $s = (X;Y_1,Y_2)$ spans slab $[B,T]$ if $Y_1 \leq B$, $T \leq Y_2$. Given a slab $[B,T]$ and a sequence Q of vertical segments sorted according to increasing abscissa and having a nontrivial intersection with the slab, the spanning segments of Q partition the slabs into a set of regions, which we technically call rectangles (these regions are actual rectangles, possibly unbounded on one or both sides).

The philosophy of the AM is the following. Once a point has been located within a slab, comparisons against the spanning segments of the slab locate the point in a rectangle. The rectangle is then sliced in two slabs by some line $y = M$, with $B < M < T$; a comparison against M locates the point in one of these slabs and from there on the search proceeds recursively in the same fashion. The search terminates when the rectangle examined is empty.

The foregoing ideas naturally suggest a recursive procedure for building the tree in which two main steps remain to be specified: (i) how to choose M in a given rectangle (choice of the cut point); (ii) how to organize in a binary tree the 0-nodes corresponding to a segment spanning the slab and the ∇-nodes corresponding to cut points in the rectangles. We shall describe step (i) in this section and step (ii) in the next one.

The choice of the cut point constitutes the main difference between the old and the new versions of the AM. In [1] the cut point is $M = \lfloor (B+T)/2 \rfloor$ for each rectangle in the slab $[B,T]$. We propose instead to select M as the median of the ordinates of all the segment endpoints following in the rectangle. This choice is aimed at decreasing the numbers of both vertical and horizontal nodes. As an example, consider a rectangle in the slab $[n/2,n]$ containing only one segment endpoint, say $n/2+1$, the other endpoint being in slab $[n,2n]$. The situation is illustrated in Figure 2.1 for $n = 16$.



(a) new algorithm (b) old algorithm

Figure 2.1. Comparison of different criteria for the choice of the cut point.

We can see, referring to Figure 2.1 (a), that the new algorithm constructs a tree with only one vertical node (corresponding to ordinate 9) and one horizontal node (corresponding to the segment abscissa). In Figure 2.1 (b) we see that the algorithm performs three cuts (at ordinates 12, 10, and 9); correspondingly 3 vertical nodes and 3 horizontal nodes (with the segment abscissa) will be allocated in the tree. For a generic value of n the new algorithm still uses one vertical and one horizontal node for the search tree of the given rectangle, while in the same case, the old algorithm uses O(logn) nodes of both types.

We are now ready to more formally define a procedure SEARCHTREE which recursively constructs the AM. The inputs are a slab [B,T] and a queue Q of segments, sorted from left to right, and intersecting the slab. The output is the corresponding search tree. We also assume that the endpoints have been presorted and their ordinates normalized to the set $\{1,2,\ldots,2n\}$, where n is the number of segments. Calling Q_0 the queue the queue of all the input segments, the AM is built by the call

$$\mathcal{K} \leftarrow \text{SEARCHTREE } (1,2n;Q_0) \quad . \tag{2.1}$$

The generic call decomposes the queue Q into strings σ's of consecutive spanning segments and γ's of consecutive nonspanning segments:

$$Q = \sigma_0\gamma_1\,\sigma_1\cdots\gamma_r\sigma_r \quad . \tag{2.2}$$

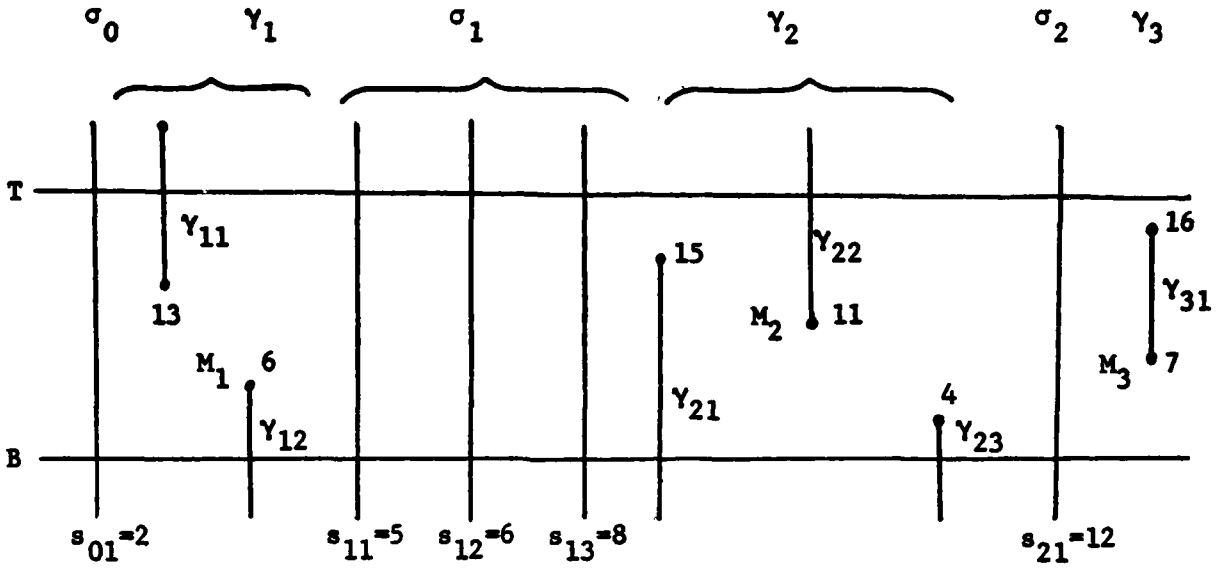
In (2.2), σ_0 and σ_r may be empty, while all the other strips are nonempty. A string γ_i corresponds to a nonempty rectangle and, after computing the cut point M_i (see Figure 2.2 (a)), two queues Q_{i1} and Q_{i2} are formed with the segments of the i-th rectangle that intersect slab $[B,M_i]$ and slab $[M_i,T]$ respectively. A ∇-node V_i with discriminator M_i is created such that

$$\text{LEFTSON } (V_i) = \text{root } (\text{SEARCHTREE } (B,M_i;Q_{i1})) \; ; \tag{2.3a}$$

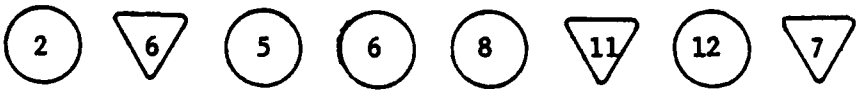
$$\text{RIGHTSON } (V_i) = \text{root } (\text{SEARCHTREE } (M_i,T;Q_{i2})) \quad . \tag{2.3b}$$

An O-node is created for each spanning segment, with discriminator equal to the abscissa of the segments. The nodes are stored in a queue

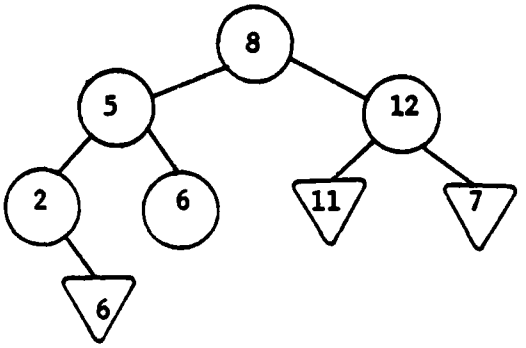
$$U = s_{01},\ldots,s_{0n_0},V_1,s_{11},\ldots,s_{1n_1},V_2,\ldots,V_r,s_{r1},\ldots,s_{rn_r} \quad , \tag{2.4}$$



(a) decomposition of a queue Q of segments in spanning segments and rectangles in slab [B,T].



(b) corresponding queue U of O-nodes and V-nodes.



(c) a possible search tree for a slab [B,T] and segments of Q.

Figure 2.2. Phases of the algorithm in processing a slab [B,T].

(see Figure 2.2 (b)). Once the queue U is completed, it is restructured into a balanced tree U by the procedure BALANCE to be described in the next section (see Figure 2.2 (c)).

A pidgin algol program for procedure SEARCHTREE is given in Table 1. Q, Q_1, Q_2, U, Z are queues; s * Q and Q * s denote the operation POP s from Q and PUSH s into Q respectively. Queue Z is used to temporarily store the segments contained in a rectangle. The subroutine CUTPOINT computes the ordinate M at which we slice the rectangle. When M is known the queues Q_1 and Q_2 of the segments that intersect the lower and upper slabs, in which the rectangle is divided, can be formed. We notice that in the Lipski-Preparata algorithm, $M = \lfloor (B + T)/2 \rfloor$ does not depend on which segments are in the rectangle and therefore queues Q_1 and Q_2 may be directly formed without using queue Z (compare with [3],[1]). This is not the case for our version of the algorithm, in which CUTPOINT computes a median of the endpoints within the rectangles.

To find the median of the points inside a given rectangle a standard median algorithm [6] can be used, which will work in time linear in the number of points. Another solution, probably faster and easier to implement, can be obtained by a simple modification of the procedure SEARCHTREE. The idea is to maintain together with the queue Q of the segments, a list P of their endpoints contained in the slab, sorted by increasing ordinate. Each segment of Q has pointers to its endpoints, when they are in P; the pointers are null otherwise. In a first scan of Q, rectangles are formed and points of P are marked with the name of the rectangle to which they belong. In a second scan, a list P_i of sorted points is easily built for

Table 1. A pidgin algol program for procedure SEARCHTREE.

```
     procedure SEARCHTREE (B,T;Q)
1    begin  Z, Q1, Q2, U ← ∅
2    while  Q ≠ ∅ do
3       begin  s = (X;Y1,Y2) ← Q
4            if (B < Y1) or (Y2 < T)  (* s does not span slab [B,T] *)
5              then Z ← s
6            if (Y1 ≤ B) and (T ≤ Y2) or (Q ≠ ∅)
7              then begin (* either s spans slab [B,T] or is the last
                             element in Q *)
8                       if  Z ≠ ∅
9                       then begin  (* the rectangle is non-empty *)
10                         M = CUTPOINT (B,T;Z);
11                         begin while (Z ≠ ∅) do
12                              z = (X;Y1,Y2) ← Z
13                              if (Y1 < M) then Q1 ← z
14                              if (Y2 > M) then Q2 ← z
15                         end (* while *)
16                         V = new horizontal node of 𝔎
17                         LEFTSON(V) = root (SEARCHTREE (B,M;Q1));
18                         RIGHTSON(V) = root (SEARCHTREE (M,T;Q2));
19                         U ← V
20                      end (* if Z ≠ ∅ *)
21           end
22           if  (Y1 ≤ B) and (T ≤ Y2)
23           then U ← s  (* s spans slab [B,T] *)
24   end  (* while Q ≠ ∅ *)
25   𝔘 = BALANCE (U)
26   return 𝔘
27 end .  (* SEARCHTREE *)
```

each rectangle with a kind of "unmerge" procedure, and its median is found. When the rectangle is cut, the corresponding list P_i is cut in two sublists P_{i1} and P_{i2} for the lower and upper slabs, respectively, by simply eliminating the median point. For instance, in the example given in Figure 2.2, the list P of points will be

$$P: 4, 6, 7, 11, 13, 15, 16 .$$

When scanning Q three rectangles are formed, the first including segments γ_{11}, γ_{12}, the second including segments γ_{21}, γ_{22}, γ_{23}, and the third including segment γ_{31}. Correspondingly the endpoints in P are marked as follows (markers are in brackets):

$$P: 4[2], 6[1], 7[3], 11[2], 13[1], 15[2], 16[3] .$$

Now a scan of P will provide the *lists of* sorted points for each of the rectangles

$$P_1: 6, 13; P_2: 4, 11, 15; P_3: 7, 16 .$$

Considering now P_2, its median $M_2 = 11$ is easily obtained as the middle point of the queue. Splitting P_2 by eliminating M_2 yields the queues P_{21}: 4, and P_{22}: 15.

To summarize the ideas introduced so far, we have shown in Figure 2.3 a set of segments with the corresponding adjacency regions. In Figure 2.4 the partition into rectangles as induced by the algorithm is shown. Note that each region of this partition is contained in an adjacency region. Finally in Figure 2.5 we give the search tree built by the algorithm for the same set of segments as in Figure 2.3. The search tree is built according to the BALANCE procedure we are going to describe in the next section.

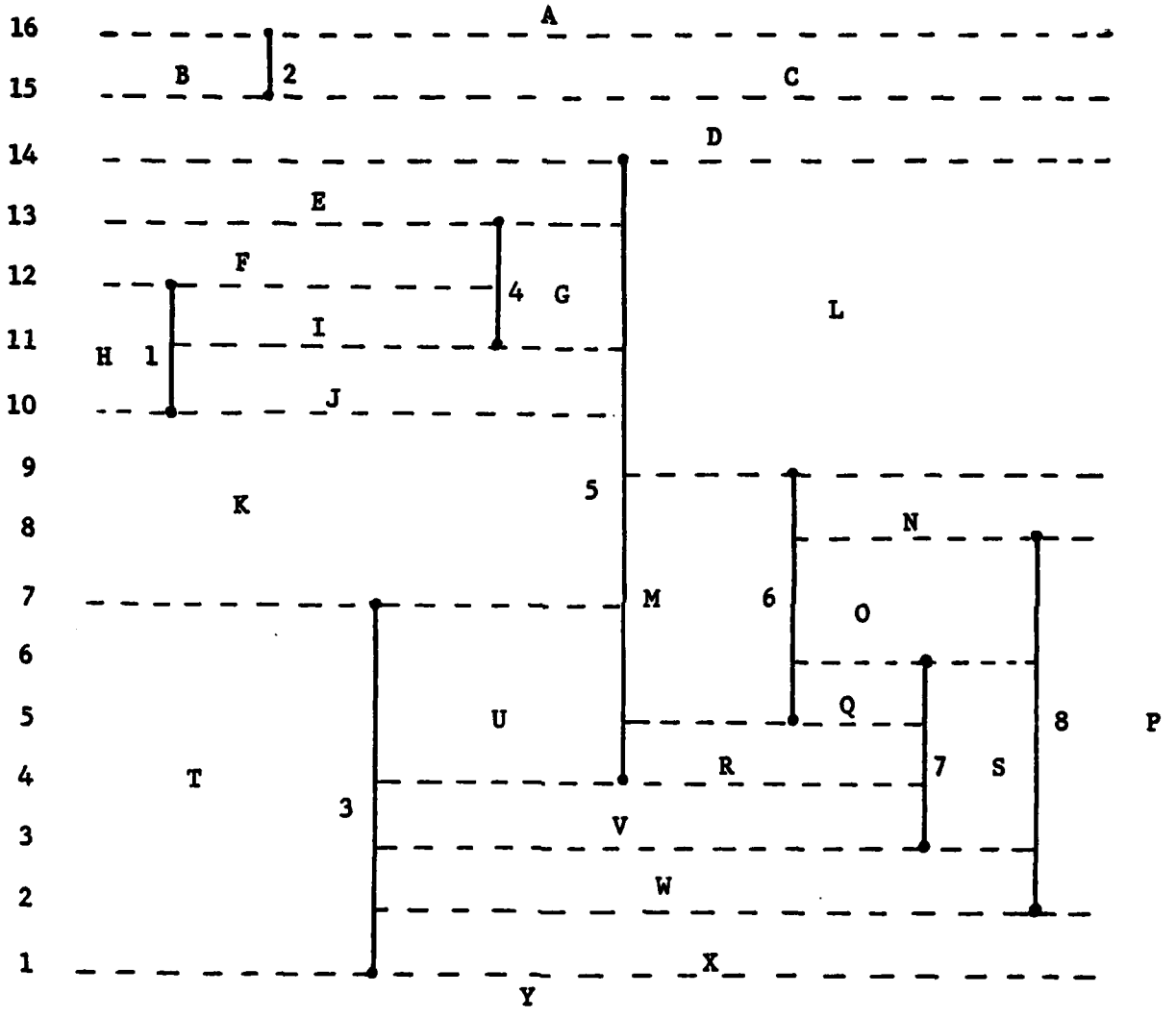


Figure 2.3. A set of 8 vertical segments and the corresponding 25 adjacency regions, which are labeled with capital letters from A to Y. The segments are numbered according to their increasing abscissa.

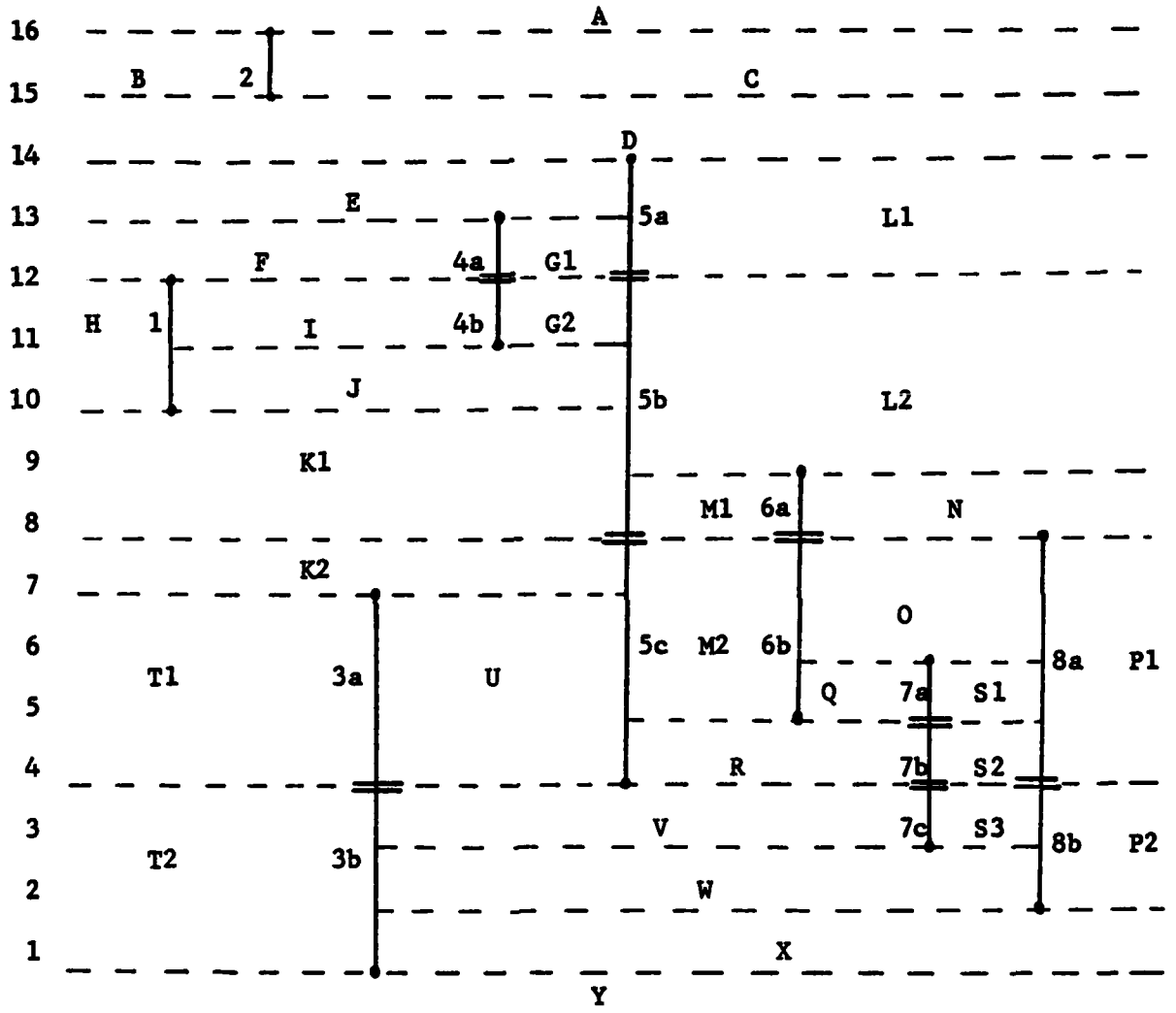


Figure 2.4. Partition of the plane created by the algorithm. Each region corresponds to a leaf in the search tree and is contained in some adjacency region; this correspondence is reflected by the notation. When a segment is cut in several parts each part is labeled with the abscissa of the original segment plus a lower case letter, serving illustration purposes only.

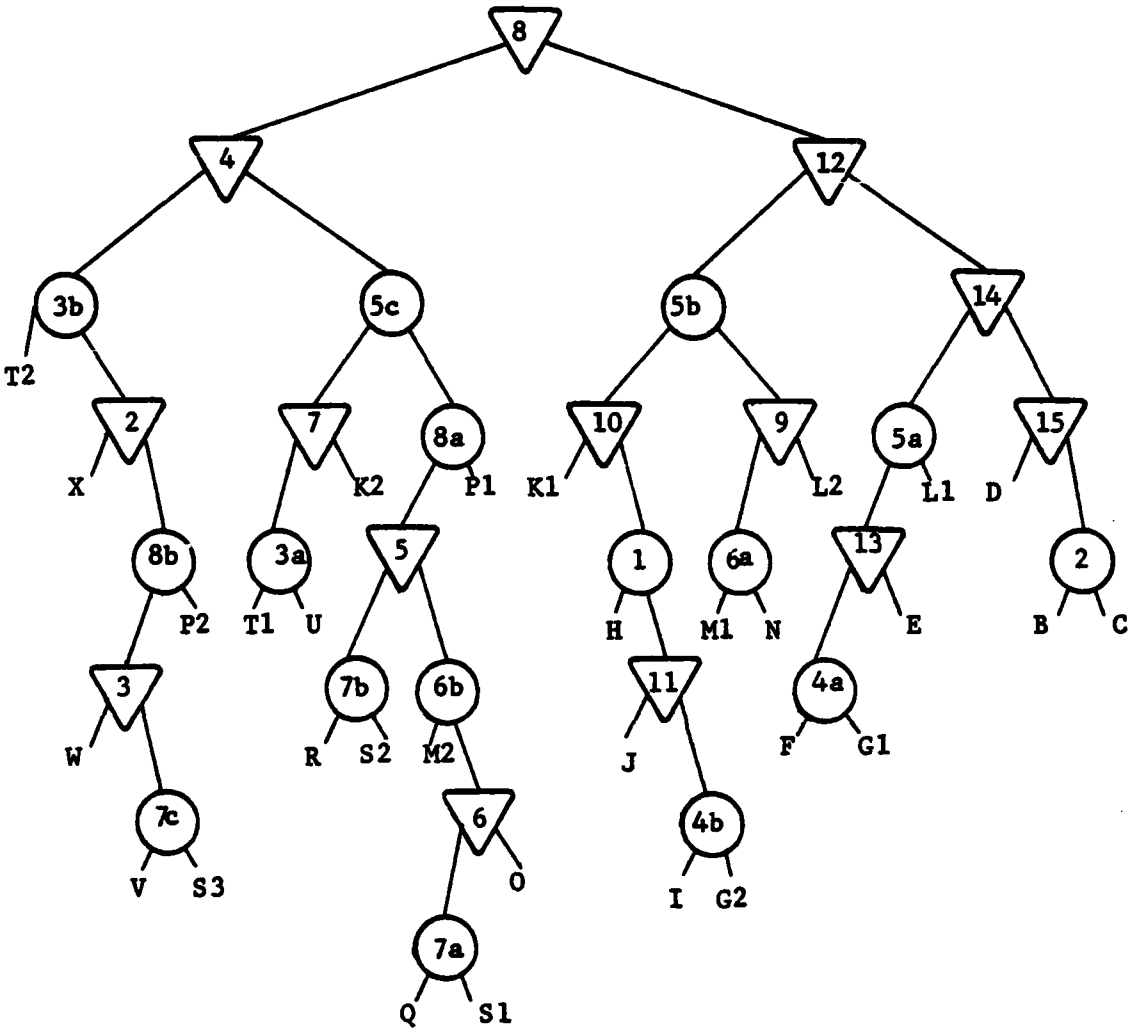


Figure 2.5. The search tree for the set of segments in Figure 2.3. Each leaf of the tree is labeled with the name of the region of points whose search terminates at that leaf. When a segment is split in several parts, there is an O-node for each such part. The lower-case letters used in this case are only to help the reader in relating the nodes of the tree to segments and regions of Figure 2.4, and do not appear in the data structure.

3. SEARCH TREE BALANCING PROCEDURE (BALANCE)

As we have seen in the last section, one call of the procedure SEARCHTREE builds a list of 0-nodes and ∇-nodes to be structured into a binary tree. In building such tree the objective is to bound its depth. The procedure BALANCE is designed to achieve an O(logn) depth. The input of BALANCE is the queue

$$U = s_{01},\ldots,s_{0n_0},V_1,\ldots,V_r,s_{r1},\ldots,s_{rn_r}$$

where the s' are 0-nodes and the V's are ∇-nodes, which are roots of partial search trees corresponding to nonempty rectangles of the current slab.

The balancing is based on the number p_i of endpoints contained in the i-th rectangle; p_i is also the number of ∇-nodes in the subtree with root V_i. In fact at each recursive call a median is selected for each nonempty rectangle of the slab, and correspondingly a ∇-node is allocated. Each endpoint is the median of some rectangle and therefore associated with a ∇-node; on the other hand when the point is selected as a median, it is no further considered by subsequent calls of SEARCHTREE, so that there is one ∇-node for each endpoint.

Procedure BALANCE works as follows on input U:

(1) if U consists of 0-nodes only, then they are arranged in a balanced binary tree;

(2) if U contains ∇-nodes let $K \triangleq p_1 + \ldots + p_r$ and let j be defined by the equations

$$p_1 + \ldots + p_{j-1} \leq K/2, \tag{3.1a}$$

$$p_1 + \ldots + p_j > K/2, \tag{3.1b}$$

which imply

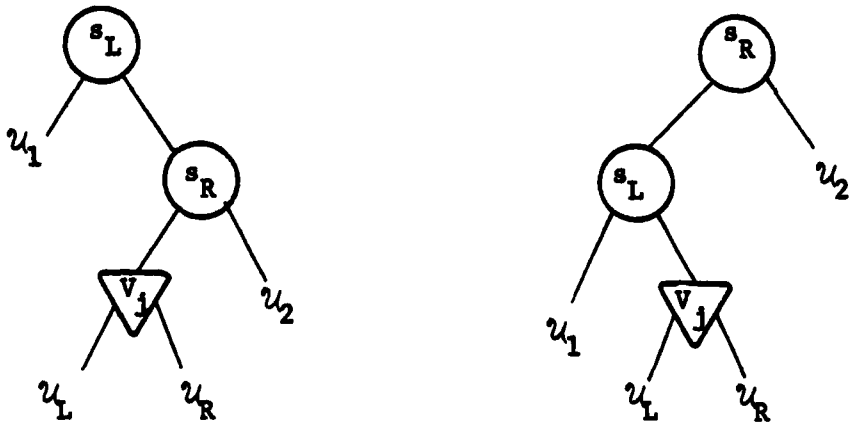
$$p_L \triangleq p_1 + \ldots + p_{j-1} \leq K/2 \tag{3.2a}$$

$$p_R \triangleq p_{j+1} + \ldots + p_s < K/2 \ . \tag{3.2b}$$

Calling s_L and s_R the left and the right spanning segments bounding the j-th rectangle, we decompose U as

$$U = U_1 \ s_L \ V_j \ s_R \ U_2 \tag{3.3}$$

$\mathcal{U}$ = BALANCE(U) is recursively defined in terms of $\mathcal{U}_1$ = BALANCE(U_1) and $\mathcal{U}_2$ = BALANCE(U_2) as shown in Figure 3.1 (the cases $p_L \geq p_R$ and $p_L < p_R$ are distinguished only to improve the average depth, while for the worst case either of the structures (a) and (b) would work). The subtrees of node V_j, $\mathcal{U}_L$ and $\mathcal{U}_R$, are the search trees corresponding to the slabs in which the j-th rectangle is sliced. They will be considered later, in the analysis of the tree.



(a) $p_L \geq p_R$ (b) $p_L < p_R$

Figure 3.1. Recursive definition of BALANCE.

In order to implement the balancing efficiently, a vector can be used to store the numbers $\sum_{i=1}^{k} p_i$, $k = 1,\ldots,r$. While this technique requires $O(r)$ time at the beginning, it allows us to find the node V_j in time logarithmic in the number of ∇-nodes involved in each recursive call of BALANCE. If there are only 0-nodes, it is also clear that they can be balanced in time linear in their number. In conclusion the BALANCE runs in time linear in the total number of nodes of the input queue U.

4. WORST CASE PERFORMANCE ANALYSIS AND STORAGE LOWER BOUNDS

In this section we analyze the worst case asymptotic performance of the method, considering the time to build the search tree (preprocessing time), the number of nodes (storage) and the depth of the tree (search time) and showing that they are O(nlogn), O(nlogn) and O(logn) respectively. In the last part of the section we introduce lower bound considerations for storage, which will be used in the next section for an appraisal of the results of the probabilistic analysis of the number of nodes.

We begin by proving a lemma showing that the number of segments and and the number of points, either in a rectangle or in a slab, are of the same order.

Lemma 4.1. Let e_R and p_R be respectively the numbers of segments and points in a rectangle, and e_S and p_S the same quantities in a slab generated by the algorithm. We have $p_R = \theta(e_R)$, and $p_S = \theta(e_S)$.

Proof. In a rectangle there are no spanning segments and therefore each segment has at least one endpoint inside the rectangle; on the other hand a segment has at most two endpoints, hence $e_R \leq p_R \leq 2\ e_R$, which proves $p_R = \theta(e_R)$. Let us now consider a slab obtained by horizontally cutting a rectangle at the median of the segment endpoints. If p_R is odd both the lower and upper slabs contain $p_S = (p_R-1)/2$ points. If p_R is even one slab contains $p_R/2$ and the other $p_R/2-1$ points. In any case $p_R/2-1 \leq p_S \leq p_R/2$ so that $p_S = \theta(p_R) = \theta(e_R)$. Since a segment in the rectangle may generate at most a segment in a slab, $e_S \leq e_R \leq p_R$. Moreover, the number of segments in the slab is at least one half of the number of points, hence $e_S \geq p_S/2 \geq (p_R/2-1)/2$. Therefore $(p_R/4 - \frac{1}{2}) \leq e_S \leq p_R$, and $e_S = \theta(p_R)$. But we have seen that $p_S = \theta(e_R)$, hence $p_S = \theta(e_S)$. □

Lemma 4.1 is useful because it shows that, in the asymptotical analysis, the work done by the algorithm in a slab or in a rectangle can be changed indifferently to the points or the segments. We now analyze separately the three performance measures.

Preprocessing time. We note that each point belongs at most to O(logn) slabs. In fact each call of the procedure SEARCHTREE processes no more than half of the number of points processed by the calling procedure. Also we note that at each call the work done to prepare subsequent calls is linear in the number of points (or segments) processed. In fact, a constant work is required for each segment to decide if it is spanning or not and, when it is not spanning, to insert it in the queues for the lower and upper slabs of the rectangle, if appropriate. Moreover the median of the points in each rectangle is found in time linear in the number of points (within the rectangle), both by using a median finding algorithm or the method proposed in Section 2. The BALANCE procedure also takes time linear in the total number of tree-nodes that it processes, and hence in the number of points of the corresponding slab. In fact the number of spanning segments is $O(e_S) = O(p_S)$ and the number r of rectangles is certainly $O(p_S)$.

In conclusion for each call O(1) work is done for each point processed by the call, resulting in O(nlogn) total time for the procedure SEARCHTREE, and therefore for the entire algorithm including presorting of segments and endpoints.

Storage. The storage is proportional to the number of nodes in the search tree. The number of ∇-nodes is 2n. The number of O-nodes for each segment is one plus the number of times the segment is cut by the median of the same rectangle. The total number of cuts can be easily bounded considering

that in a rectangle of w points at most w-1 segments are cut by the median, and that in a rectangle with 1 and 2 points the maximum number of cuts is 0 and 1 respectively. So the function f(w) recursively defined by

$$f(1) = 0, \quad f(2) = 1 \quad , \tag{4.1a}$$

$$f(2w) = f(w) + f(w-1) + 2w-1 \quad , \tag{4.1b}$$

$$f(2w+1) = wf(w) + 2w \quad , \tag{4.1c}$$

is a bound for the number of cuts when processing s points. Since $f(w) = O(w \log w)$, the total number of nodes in the tree is $O(n\log n)$.

Search time. This part of the analysis is somewhat delicate because the depth $\delta(\mathcal{U})$ of the tree $\mathcal{U}$ = BALANCE(U) depends on the level of recursion of SEARCHTREE in which the queue U has been formed (through the number of O-nodes and the weight of V-nodes), and also on the way in which the nodes are arranged in the tree. In fact, suppose a V-node V is placed at distance d_1[1] from the root of u and that the subtree rooted in V has depth d_2, then $\delta(\mathcal{U}) \geq d_1 + d_2$. The same is true when a subtree of O-nodes of depth d_2 is formed and its root is placed at distance d_1 from the root of $\mathcal{U}$. These considerations lead us to the following definition and remarks.

1) The weight of a slab is defined as $K = p_1 + \ldots + p_r$, and is the total number of endpoints contained in the slab.

[1] According to standard terminology, this distance is the number of arcs in the path from V to the root.

2) The number H of spanning segments in a slab of weight K is at most K + 2. To prove this claim, we consider the rectangle whose split generates the slab. A segment spanning the slab must have one endpoint in the rectangle; this endpoint either is the cutpoint or lies in the companion slab originating from the same rectangle, which has at most K + 1 points.

3) The level of a recursive call of BALANCE is defined as follows: BALANCE(U) has level 0; the calls made by a call at level i have level i + 1.

We are now ready to state the following lemma.

Lemma 4.2. The tree $\mathcal{U}$ constructed by the procedure BALANCE for a slab of weight K has a depth $\delta(\mathcal{U}) \leq \log K + 4$.

Proof. (By induction on K).

Basis. For K = 1, a slab may have at most one nonspanning and two spanning segments and therefore $\mathcal{U}$ has at most three 0-nodes and one ∇-node, so that $\delta(\mathcal{U}) \leq 4$.

Inductive step. We assume now that, for $K' < K$, $\delta(\mathcal{U}) \leq 3 \log K' + 4$.

(i) ∇-nodes. From the definition of BALANCE and Eqs. (3.2a) and (3.2b) it is easy to see that at each call the weight of the input is at least halved, so that the input of a call at level i has weight at most $K/2^i$. Also, if there are ∇-nodes, the i-level call allocates one of them, say V, at a distance no more than 2(i+1) from the root of $\mathcal{U}$. The subtrees of V are balanced trees of weight no more than $K/2^{(i+1)}$, and by the inductive hypothesis they have a depth $\leq 3 \log(K/2^{i+1}) + 4 = 3 \log K - 3(i+1) + 4$; therefore the distance of the leaves of such subtrees from the root of $\mathcal{U}$ is

$\leq 3 \log K - 3(i+1) + 4 + 2(i+1) + 1 \leq 3 \log K + 4$.

(ii) 0-nodes. If the input of a call at level i+1 has no ∇-nodes, but the calling procedure at level i has some, we argue as follows. Since the input of the calls at level i has weight less than $K/2^i$, i cannot be larger than $\lceil \log K \rceil$. Therefore the root of the tree of 0-nodes built by the call at level (i+1) has a distance from the root of $\mathcal{U}$ which is at most $2(\lceil \log K \rceil + 1)$ and has a depth at most $\log\lceil K+2 \rceil$ since there are less than K + 2 nodes in the input. In conclusion the distance of the leaves of the 0-node tree from the root of $\mathcal{U}$ is at most $2(\lceil \log K \rceil + 1) + \log\lceil k+2 \rceil = \leq 3 \log K + 4$. This completes the proof of the lemma.

Considering that the search tree is constructed by a call of BALANCE on an input of weight K = 2n, we prove the following.

Theorem 4.1. The entire search tree has a depth bounded by 3 logn + 7.

Lower bounds. We have already said in the introduction that there are point-location algorithms linear in the storage and we will show in the next section that our algorithm uses expected linear storage. It is also trivial to see that linear storage is asymptotically optimal, i.e., within a multiplicative constant of the minimum. Now we would like to know something more about such a constant. We obtain the following simple, but interesting result: the number of nodes in the search tree for a set of n segments with distinct endpoint ordinates is at least 3n. Notice that this is a lower bound not only for the worst case, but for all the instances of the problem, and therefore applies also to average results. We give here two segments to establish the stated result.

The first argument is almost trivial. We observe that each segment is specified by 2 endpoint ordinates and one abscissa; therefore it will generate at least 2 ∇-nodes and one O-node in any search tree able to solve the adjacency problem. In fact by changing only one of these parameters we change at least some adjacency region and therefore the parameter must appear in the tree to account for this change.

Another argument stems from different considerations. The search tree is a binary tree and therefore the number of different search-paths (including the exit from the last node traversed which can be left or right) equals the number of nodes in the tree plus one. Each path corresponds to a region (see Figure 2.5 as an example) of the partition reflected in the adjacency map. We can conclude that the number S of nodes in the tree, and the number A of adjacency regions, must satisfy $S \geq A - 1$. In order to complete our reasoning we need to estimate A. If all the endpoint ordinates are distinct, there are $A = 3n + 1$ adjacency regions [1], and therefore $S \geq 3n$.

We will reconsider the 3n lower bound for S in the next section, when analyzing the average behavior of our algorithm.

5. PROBABILISTIC ANALYSIS

In this section we derive some results on the average performance of the algorithm, the main purpose being to show that the expected storage is linear in the number of segments. We also show that the expected time for the procedure SEARCHTREE is linear.

5.1. Random Model

To obtain average-case performance results we need a probabilistic model for the input of our algorithm, i.e. for the set $\mathcal{A} = \{s_1,\ldots,s_n\}$ of segments. We have to consider that while we are dealing with segments whose endpoints are real numbers, the only feature of the input which is relevant to the algorithm is the relative order of the endpoints of the input segments. In other words, all the inputs that result in the same set of normalized segments are equivalent. Therefore the number of possible inputs, for a given input size n, is essentially finite.

In principle the input $\mathcal{A}$ is probabilistically described by the joint distribution of the endpoints. From this distribution the probability of each set of normalized segments can be computed and, for each normalized input, the size of the search tree built by the algorithm can be calculated. The expected size of the tree could be obtained by averaging the tree size according to the computed distribution of normalized inputs.

In practice the approach outlined above would result in a very heavy combinatorial problem and can hardly be used. To overcome this difficulty a suitable model of the set $\mathcal{A}$ will be introduced that, while simplifying the analysis, will still preserve the main features of the original problem.

The difficulties in analyzing the average behavior of the algorithm, when operating on a finite input, arise mainly from two facts: (1) the cutpoints are random variables; (2) there are some "boundary effects" that make some statistical properties, e.g. the average number of cuts affecting a given segment s_j, dependent upon j, or, in other words, not stationary. On the other hand the median of a very large number of random variables has generally very small variance and, for a reasonably large number of segments, the "boundary effects" should be all but negligible.

The foregoing considerations suggest that the analysis would be greatly simplified by considering the case of an infinite number of segments, and therefore modeling the input by a pair-valued random process

$$s_j = (B_j, T_j) \ , \ B_j < T_j \ , \ j \in Z \ , \tag{5.1}$$

where B_j and T_j are the bottom and top ordinates of segment s_j, Z is the set of the integers, and the abscissa X_j of s_j is increasing with j. Notice that the actual value of X_j is irrelevant to the algorithm as long as the order of the segments does not change.

In the following we derive results using model (5.1) for the input of our algorithm, with further assumptions on the process $\{s_j, j \in Z\}$. The entire analysis will be carried out under:

Assumption A1: the process s_j is a sequence of mutually independent and identically distributed random pairs.

According to Assumption A1 the process s_j will be probabilistically specified in a complete manner by the segment-endpoint joint distribution

$$F_{BT}(b,t) \triangleq P[B \leq b, T \leq t] \tag{5.2}$$

which is, by hypothesis, independent of j. Here and in the sequel, we follow the convention of denoting a random variable by a capital letter and the values it assumes by the corresponding lower case letter.

Generators. While the bottom B and the top T are a natural description of a vertical segment, for our purposes it is convenient to consider the segment endpoints U and V to be an unordered pair from which B and T can be recovered as

$$B = \min\{U,V\}, \tag{5.3a}$$

$$T = \max\{U,V\} \quad . \tag{5.3b}$$

We call U and V generators of the segment and we describe them probabilistically by the joint distribution

$$F_{UV}(u,v) \triangleq P[U \leq u,\ V \leq v] \quad . \tag{5.4}$$

There are several advantages in working with generators. One is that generators can be assumed symmetrically distributed, i.e. $F_{UV}(u,v) = F_{UV}(v,u)$, and therefore identically distributed, i.e. $F_U(u) = F_V(u)$. Here $F_U(u) \triangleq P[U \leq u]$ and $F_V(v) \triangleq P[V \leq v]$ are the marginal distribution functions of U and V. Moreover, for the generators we may assume independence together with identical distribution, while this is not possible for B and T, since $B \leq T$.

It is also important to notice that there is no loss of generality in considering generators. In fact, given B and T we can always construct some generators U and V with the desired properties of symmetry by letting

$$U = QB + (1-Q)T, \tag{5.5a}$$

$$V = (1-Q)B + QT, \tag{5.5b}$$

when $Q \in \{0,1\}$ is a random variable, independent of B and T, with $P[Q = 0] = P[Q = 1] = \frac{1}{2}$. It is easy to show that for such U and V the joint distribution is

$$F_{UV}(u,v) = \tfrac{1}{2}(F_{BT}(u,v) + F_{BT}(v,u)) \quad , \tag{5.6}$$

and therefore $F_{UV}(u,v) = F_{UV}(v,u)$.

<u>A robustness result</u>. A first advantage of the generator symmetry stems from a fact we have already stated in different form: the adjacency map is invariant under any continuous one-to-one transformation of the plane along the direction of the segments. Therefore we can study the problem by replacing the original ordinate y with a new ordinate z defined as

$$z = F_U(y) \quad . \tag{5.7}$$

If F_U is continuous (i.e. $P[U = y] = 0$ for all y's), then the new generators

$$U_z = F_U(U) \qquad V_z = F_V(V) \quad , \tag{5.8}$$

are uniform in the interval [0,1]. This is easily seen when F_U is strictly increasing and therefore invertible. In fact $U_z \in [0,1]$, (see Figure 5.1), and if $u \in [0,1]$ then

$$F_{U_z}(u) \triangleq P[U_z \le u] = P[F_U(U) \le u]$$

$$= P[U \le F_U^{-1}(u)] \triangleq F_U(F_U^{-1}(u)) = u \quad ,$$

which is the uniform distribution. If F_U is constant on some interval, the inverse F_U^{-1} is not properly defined, but the proof will still go thru with F_U^{-1} replaced by $\hat{F}_U^{-1}(u) \triangleq \sup\{x | F_U(x) \leq u\}$.

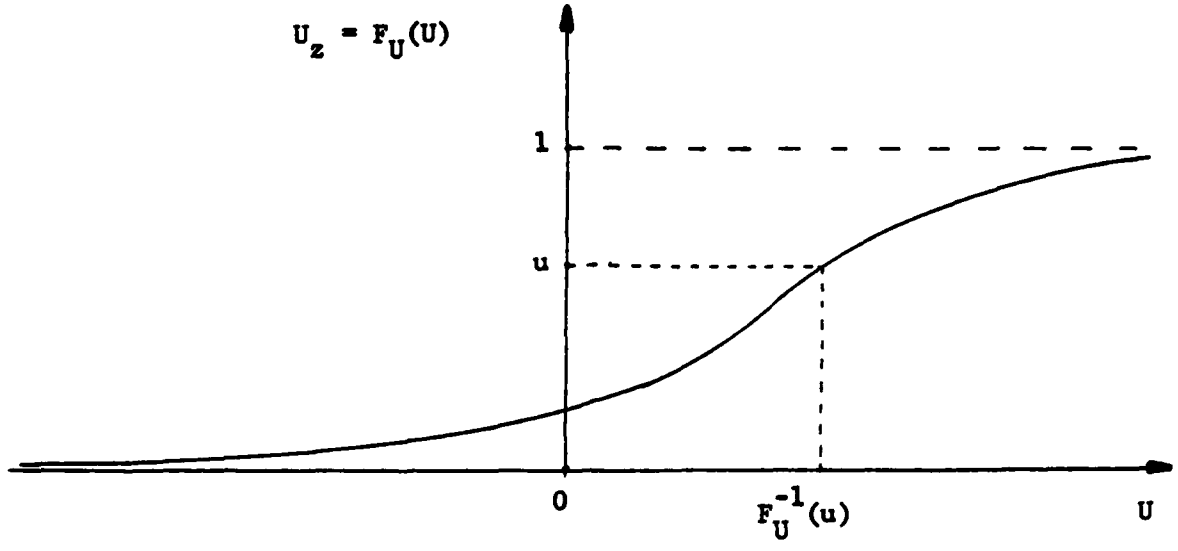


Figure 5.1. A random variable U transformed according to its own distribution F_U becomes uniform in [0,1].

We conclude that our algorithm presents a robustness property, its complexity being independent of the first order distribution of the generators.

Without loss of generality with respect to Assumption A1 we will develop the analysis under

Assumption B1: the process $s_j = (U_j, V_j)$ is a sequence of mutually independent and identically distributed random pairs such that:

(i) the joint distribution $F_{UV}(u,v) = F_{UV}(v,u)$ is symmetric;

(ii) the marginal distribution $F_U(u)$ and $F_V(v)$ are uniform in [0,1].

5.2. The Goals of the Analysis: Space and Time

The first quantity we want to analyze is the number S of nodes in the search tree. For the case of n segments, if the algorithm cuts segment s_j by means of c_j medians, we have

$$S = 3n + \sum_{j=1}^{n} c_j \quad . \tag{5.9}$$

In fact there are 2n ∇-nodes, one for each endpoint, and (c+1) O-nodes for a segment cut c times. Denoting by E the expectation operator on random variables, our goal is to compute $E[c_j]$ to get

$$E[S] = 3n + \sum_{j=1}^{n} E[c_j] \quad . \tag{5.10}$$

In our model where n is ∞, Eqs. (5.9) and (5.10) have no sense, but we can still compute $E[c_j]$. If this quantity turns out to be finite we can say that our algorithm has asymptotically linear storage, and we can estimate the storage, at least for large values of n, by the formula

$$E[S] = (3 + E[c])n \quad . \tag{5.11}$$

Here and in the following we drop the subscript j in segment s_j and related quantities such as c_j, since their statistical properties are independent of j due to the stationarity of the input process.

As far as the time analysis is concerned, we have seen that the worst-case performance is O(nlogn) both for presorting and SEARCHTREE. However the work per endpoint in each recursive call of SEARCHTREE that processes that point is bounded by a constant. Therefore if we can prove that the average number of calls in which a point is processed is constant we can conclude that SEARCHTREE runs in expected linear time.

5.3. Principal Slabs and Principal Medians

In order to analyze the algorithm we take a look at the way it works and introduce a suitable terminology to describe it. A slab whose y-interval is of the kind [0,y] or [y,1] is called external. The probability that there is a spanning segment in an external slab is zero. Hence the algorithm will process an external slab by cutting it at the median, which, with probability 1, is the arithmetic mean of bottom and top of the slab. For example the first cut will be at $y = \frac{1}{2}$, and will generate the slabs $[0,\frac{1}{2}]$ and $[\frac{1}{2},1]$; slab $[0,\frac{1}{2}]$ will be cut at $y = \frac{1}{4}$ and so on. Cuts of external slabs will generate the following family of medians we call principal medians:

$$\mathcal{M} \triangleq \{m_k \mid k \in Z\} \tag{5.12}$$

where $m_0 \triangleq \frac{1}{2}$, $m_k \triangleq 2^{k-1}$ for $k < 0$, $m_k \triangleq 1 - 2^{-(k+1)}$ for $k > 0$. Correspondingly the following principal slabs are formed:

$$\text{slab}(k) \triangleq \begin{cases} \text{slab}[m_k, m_{k+1}] & , \quad k \leq 1, \\ \text{slab}[m_{k-1}, m_k] & , \quad k \geq 1 \quad . \end{cases} \tag{5.13}$$

The principal slabs are shown in Figure 5.2.

It is convenient to analyze the procedure SEARCHTREE separating the stage in which the principal medians are formed from the subsequent processing of the interiors of the principal slabs. One reason for this approach is that (depending on the distribution F_{UV}) the principal slabs will generally be partitioned into rectangles by spanning segments. Medians in rectangles have only a local effect and need to be analyzed differently.

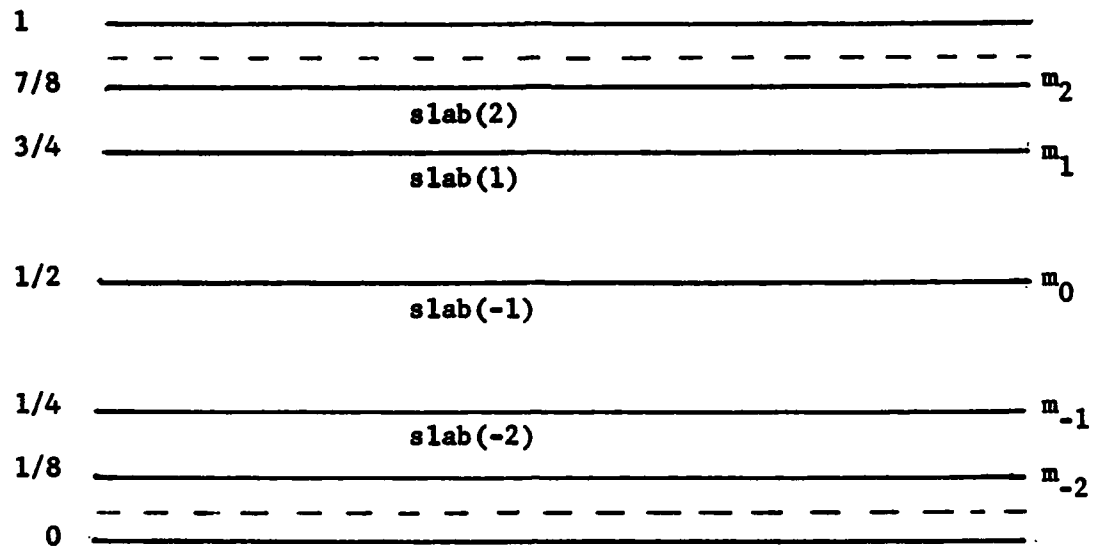


Figure 5.2. Principal slabs and principal medians.

According to this approach, let us consider a segment s = (U,V). The general situation is shown in Figure 5.3, where slab (α) and slab (β) are the principal slabs containing U and V respectively. We can express the number c of cuts performed by the algorithm on segment s as

$$c = c_0 + c(U) + c(V), \tag{5.14}$$

where c_0 is the number of cuts due to the principal medians, c(U) and c(V) are the numbers of cuts occurring in the subsequent processing of slab (α) and slab (β), respectively. When slab (α) = slab (β), or in other words U and V are in the same principal slab, clearly $c_0 = 0$, but we still have

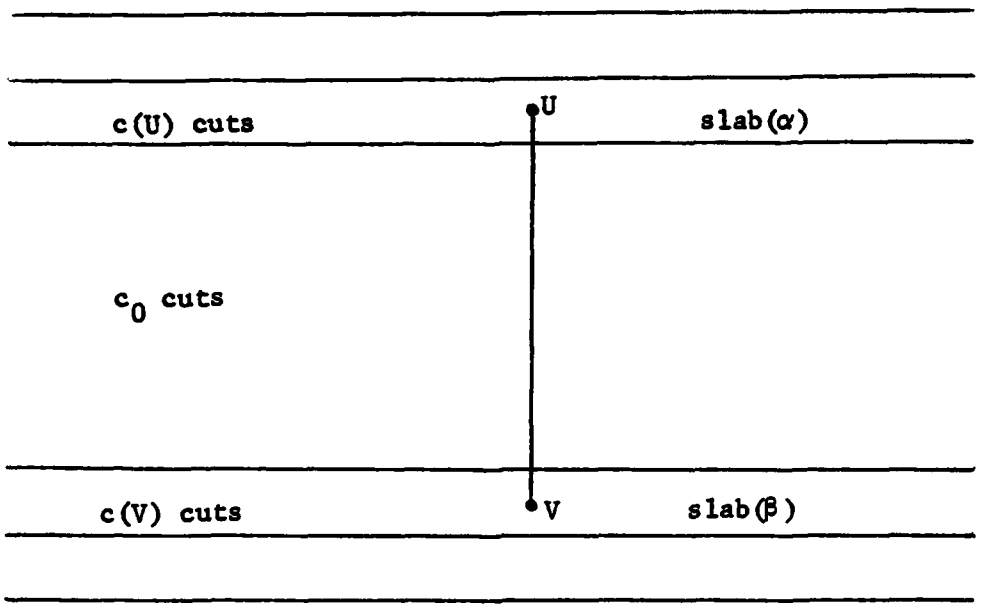


Figure 5.3. Decomposition of the set of cuts affecting a given segment.

to specify how to charge the cuts to each endpoint. A reasonable convention is to refer to the first median M cutting s. The cut due to M can be charged ½ for each endpoint. Subsequent cuts are charged to V for medians in the interval [V,M] and to U for medians in the interval [M,U], when $U > V$. Similarly, when $V > U$.

5.4. Mean Value of c_0

In this section we first present a worst-case argument that shows that $E[c_0]$ is finite, and actually less than 3, for any distribution F_{UV}. Then we give an exact expression for $E[c_0]$ in terms of F_{UV}.

Proposition 5.1. For any set of n segments the total number of cuts due to the principal medians is less than 3n.

Proof. The claim follows by considering that, if on one side of median m there are q points, median m cuts at most q segments. Therefore m_0 cuts at most n segments, m_{-1} and m_1 cut at most n/2 segments each, and so on. Of course only a finite set $\mathcal{M}^*$ of principal medians is to be considered. Thus

$$\sum_{j=1}^{n} c_{0j} = \sum_{j=1}^{n} \sum_{m \in \mathcal{M}^*} c_{0j}(m) \leq n + 2(\tfrac{1}{2} n + \tfrac{1}{4} n + \ldots) = 3n \ , \qquad (5.15)$$

where $c_{0j}(m) = 1$ if m cuts segment s_j, and $c_{0j}(m) = 0$ otherwise.

From (5.15) we get

$$\frac{1}{n} \sum_{j=1}^{n} E[c_0] \leq 3 \ , \qquad (5.16)$$

which is the stationary model, where $n \to \infty$ and $E[c_{0j}] \to E[c_0]$, becomes

$$E[c_0] < 3 \ . \qquad (5.17)$$

To compute $E[c_0]$ exactly we write c_0 in terms of $c_o(m)$ as

$$c_0 = \sum_{m \in \mathcal{M}} c_0(m) \ . \qquad (5.18)$$

Since $c_0(m) \in [0,1]$, we have

$$E[c_0(m)] = P[c_o(m) = 1] = 2(F_{UV}(m,1) - F_{UV}(m,m)) \ . \qquad (5.19)$$

The last expression for $P[c_0(m) = 1]$ is obtained by observing that (see Figure 5.4)

$$P[c_0(m) = 1] = P[U < m, V > m] + P[U > m, V < m] \ ,$$

and

$$P[U < m, V > m] = P[U < m] - P[U < m, V \leq m] = F_{UV}(m,1) - F_{UV}(m,m).$$

Hence, by the symmetry of generators, Eq. (5.19) follows.

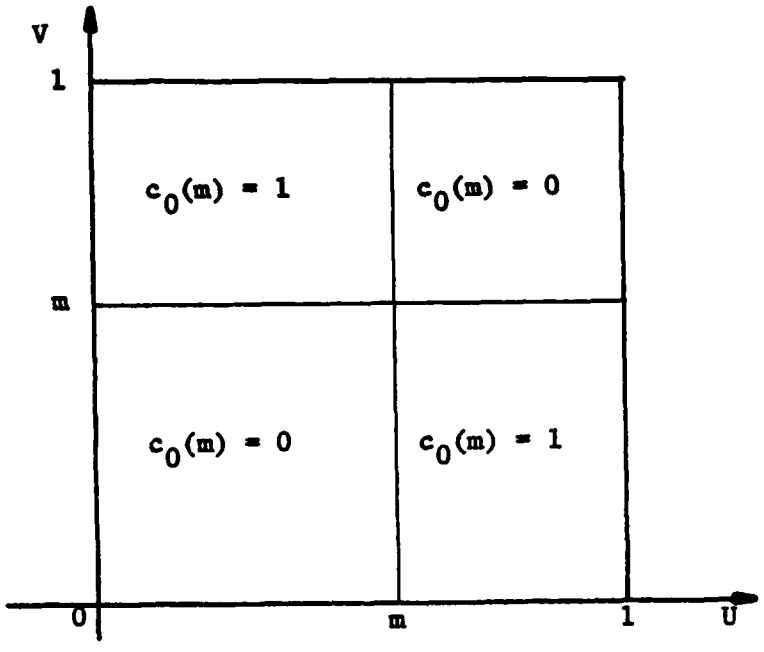


Figure 5.4. Set of generator pairs resulting in segments cut by median m.

In Eq. (5.19) we may use the fact that $F_{UV}(m,1) = m$, because of uniformity in [0,1] of U and V.

Finally, if we take the expected value of both sides of Eq. (5.18) and we use Eq. (5.19) we get a series expansion for $E[c_0]$. We summarize the main results of this section in the following proposition.

Proposition 5.2. Under Assumption B1, the average number of cuts due to the principal medians is

$$E[c_0] = \sum_{m \in \mathcal{M}} 2(m - F_{UV}(m,m)) \tag{5.20}$$

and, for every distribution F_{UV},

$$E[c_0] < 3 \quad . \tag{5.21}$$

Incidentally, we note that bound (5.21), derived here as a consequence of Proposition 1, can be obtained directly from Eq. (5.20). For this purpose in Eq. (5.20) the range of summation $\mathcal{M}$, defined in Eq. (5.12), can be split in sets $\{m_k | k \leq 0\}$ and $\{m_k | k > 0\}$. The sum is then upper bounded using the fact that in the first set $m_k - F_{UV}(m_k, m_k) \leq m_k = 2^{-k}$, and, in the second set, $m_k - F_{UV}(m_k, m_k) \leq (1 - m_k) = 2^{-(k+1)}$. The result (5.21) then follows by the sum of two geometric series.

5.5. Analysis of Rectangles in Principal Slabs

We still have to analyze the contribution of c(U) and c(V) to c, in Eq. (5.14). For this task we need to consider the work done by the algorithm when processing the interiors of the principal slabs. In general a given generator, say U, will be contained in a rectangle $\mathcal{R}$ formed by the boundary medians of a principal slab together with two spanning segments. Since the algorithm processes the rectangles independently of each other, c(U) is determined only by the contents of $\mathcal{R}$. Intuition suggests that, on the average, c(U) increases with the total number W of points in $\mathcal{R}$. This we shall now analyze.

Analysis of segments. We begin by studying the relative position of a segment and a generic slab [m,m']. We define the set of segments that respectively: (i) span the slab, (ii) have one endpoint in the slab, (iii) have two endpoints in the slab, or (iv) are outside the slab. More formally

$$\mathcal{S} \triangleq \{s | B < m,\ m' < T\} \quad , \tag{5.22a}$$

$$\mathcal{S}_1 \triangleq \{s | B < m < T < m' \text{ or } m < B < m' < T\} \tag{5.22b}$$

$$\mathcal{B}_2 \triangleq \{s \mid m < B,\ T < m'\} \quad , \tag{5.22c}$$

$$\mathcal{O} \triangleq \{s \mid T < m \text{ or } B > m'\} \quad . \tag{5.22d}$$

It is also useful to let

$$\mathcal{J} \triangleq \mathcal{I} \cup \mathcal{B}_1 \cup \mathcal{B}_2 \tag{5.22e}$$

as the set of segments which have non-trivial intersection with the slab [m,m']. Of course all the sets defined by Eqs. (5.22a)-(5.22e) are functions of m and m', but, for simplicity, we do not show it in the notation.

In Figure 5.5 typical examples are given for segments in all classes, and in Figure 5.6 the classes are shown in the (U,V) plane.

The probability that a segment s belongs to one of the classes we have just defined can be expressed in terms of F_{UV} as follows.

$$P[s \in \mathcal{I}] = 2(F_{UV}(m,1) - F_{UV}(m,m')) = 2(m - F_{UV}(m,m')) \quad , \tag{5.23a}$$

$$P[s \in \mathcal{B}_1] = 2(F_{UV}(m',1) - F_{UV}(m,1)) - 2P[s \in \mathcal{B}_2] = 2(m-m'-P[s \in \mathcal{B}_2]) \quad , \tag{5.23b}$$

$$P[s \in \mathcal{B}_2] = F_{UV}(m',m') - 2F_{UV}(m,m') + F_{UV}(m,m) \quad , \tag{5.23c}$$

$$\begin{aligned} P[s \in \mathcal{O}] &= F_{UV}(m,m) + F_{UV}(m',m') - 2F_{UV}(m',1) + 1 \\ &= 1 - 2m' + F_{UV}(m',m') + F_{UV}(m,m) \quad , \end{aligned} \tag{5.23d}$$

$$P[s \in \mathcal{J}] = 1 - P[s \in \mathcal{O}] = 2m' - F_{UV}(m',m') - F_{UV}(m,m) \quad . \tag{5.23e}$$

When we focus our attention on the slab [m,m'] we are interested only in segments belonging to $\mathcal{J}$. So we introduce the following conditional probabilities, which are well defined since $P[s \in \mathcal{J}] > 0$:

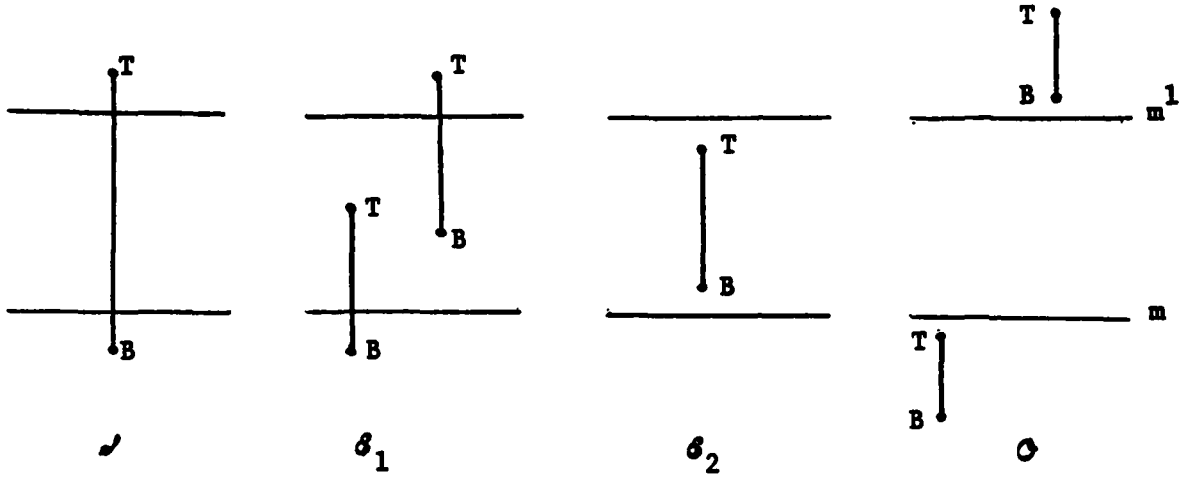


Figure 5.5. Relative position of a segment and a slab.

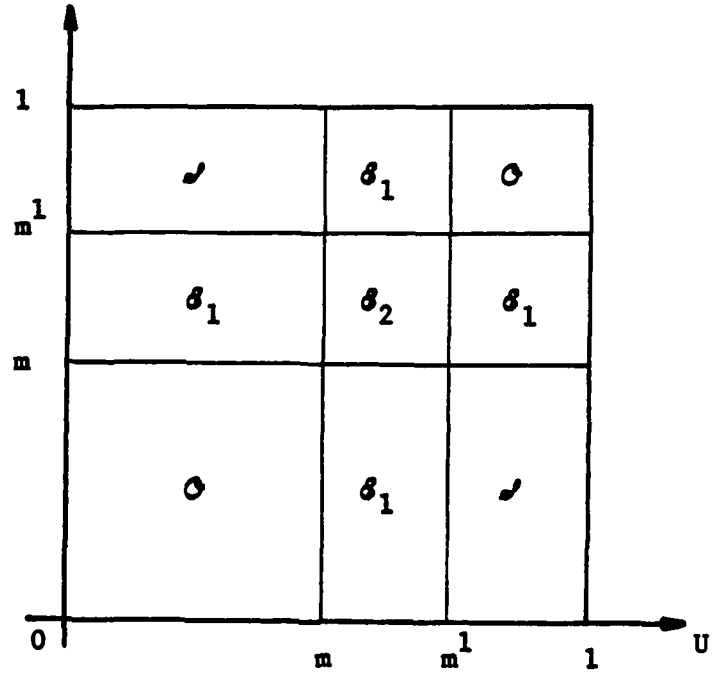


Figure 5.6. Classes of segments in terms of generators pairs.

$$p_0 \triangleq P[s \in \mathcal{A} | s \in \mathcal{A}] = P[s \in \mathcal{A}]/P[s \in \mathcal{A}] \quad , \tag{5.24a}$$

$$p_1 \triangleq P[s \in \mathcal{B}_1 | s \in \mathcal{A}] = P[s \in \mathcal{B}_1]/P[s \in \mathcal{A}] \quad , \tag{5.24b}$$

$$p_2 \triangleq P[s \in \mathcal{B}_2 | s \in \mathcal{A}] = P[s \in \mathcal{B}_2]/P[s \in \mathcal{A}] \quad . \tag{5.24c}$$

To guarantee the existence (with probability 1) of rectangles in the principal slabs, we introduce the following assumption, that is satisfied by most reasonable distributions.

Assumption A2: the distribution F_{UV} is such that probability p_0 is nonzero for all the principal slabs.

Analysis of rectangles. Let us now consider a generator of s, say $U \in$ slab(i). Let s_L and s_R the left and the right segments closest to U, among those which span slab(i), and call $\mathcal{R}$ the rectangle closed by these two segments. The number W of segment endpoints in node $\mathcal{R}$ can be written as

$$W = N_L + N_R + 1 + \phi(V) \quad , \tag{5.25}$$

where N_L is the number of points in $\mathcal{R}$ to the left of U, N_R is the number of those to the right, the third term, 1, accounts for U itself, and $\phi(V)$ is 1 if V - the other generator of s - is also in slab(i) and 0 otherwise. An example is given in Figure 5.7, where $V \notin$ slab(i) so that $\phi(V) = 0$, $N_L = 2$, $N_R = 4$, and $W = 2+4+1+0 = 7$.

Now we want to find the probability mass function (p.m.f.) of W. It is useful to partition the sample space into regions where U belongs to a given slab. The theorem on total probability for this partition yields

$$P[W=w] = \sum_{|i|=1}^{+\infty} P[W=w | U \in \text{slab}(i)] P[U \in \text{slab}(i)] \quad . \tag{5.26}$$

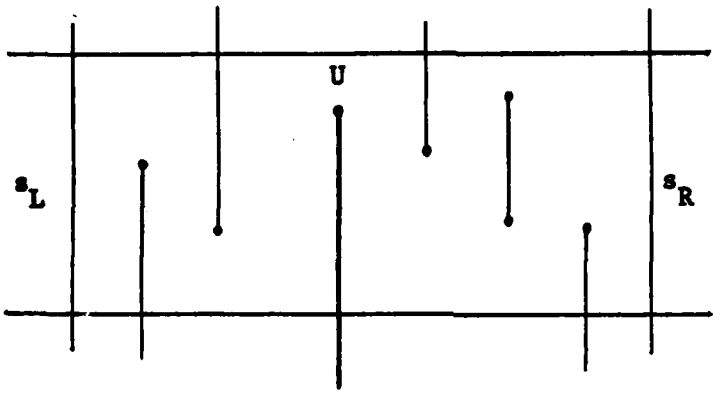


Figure 5.7. The rectangle $\mathcal{R}$ including generator U.

Since U is uniform in [0,1], $P[U \in \text{slab}(i)]$ equals the width $2^{-(|i|+1)}$ of the slab, and we only need to compute $P[W=w|U \in \text{slab}(i)]$. First let us observe that, under the condition $U \in \text{slab}(i)$, N_L, N_R and $\psi(V)$ are independent random variables. Also let us denote for brevity $P[\ |U \in \text{slab}(i)]$ as $P_i[\ ]$. Then we have

$$P_i[W=w] = P_i[N_L+N_R=w-2]P_i[V \in \text{slab}(i)] + P_i[N_L+N_R=w-1]P_i[V \notin \text{slab}(i)]. \quad (5.27)$$

Now we have

$$\begin{aligned} P_i[V \in \text{slab}(i)] &= P[V \in \text{slab}(i)|U \in \text{slab}(i)] \\ &= P[U,V \in \text{slab}(i)]/P[U \in \text{slab}(i)] \\ &= P[s \in \mathcal{S}_1]2^{|i|+1} \ ; \end{aligned} \quad (5.28)$$

$$P_i[V \notin \text{slab}(i)] = 1 - P_i[V \in \text{slab}(i)] \ . \quad (5.29)$$

It remains to evaluate $P_i[N_R + N_L = w]$. Since N_L and N_R are independent of each other, the p.m.f. of the sum is the discrete convolution of the p.m.f's of the addends N_L and N_R. So let us compute $P_i[N_R = w]$, which also equals $P_i[N_L = w]$ by the symmetry of the configuration. Referring to Figure 5.7, we can define L_1 and L_2 as the numbers of segments in classes $\mathcal{S}_1$ and $\mathcal{S}_2$ of slab(i) that are between U and s_R. The numbers of endpoints at the right of U is obviously $N_R = L_1 + 2L_2$, and therefore

$$P_i[N_R = w] = \sum_{h=0}^{\lfloor w/2 \rfloor} P_i[L_1 = w - 2h, L_2 = h] \quad . \tag{5.30}$$

Since there are $\binom{w-h}{h}$ ways to form a string of $w - 2h$ segments of $\mathcal{S}_1$ and h segments of $\mathcal{S}_2$, followed by a segment of $\mathcal{S}$,

$$P[L_1 = w-2h, L_2 = h] = p_0 \binom{w-h}{h} p_1^{(w-2h)} p_2^h \quad . \tag{5.31}$$

Thus, by Eqs. (5.30) and (5.31) $P_i[N_R = w]$ can be expressed in terms of the probabilities p_0, p_1, p_2 of slab(i). Finally

$$P_i[N_L + N_R = w] = \sum_{h=0}^{w} P_i[N_L = h] P_i[N_R = w - h] \tag{5.32}$$

is the announced convolution for the p.m.f. of $N_L + N_R$.

Summary. Equations (5.23) and (5.24) give the probabilities p_0, p_1, p_2, for a given slab, in terms of F_{UV}. Equations (5.30) (5.31) and (5.32) give the p.m.f. of $N_L + N_R$. Equations (5.28), (5.29), (5.30) substituted in (5.27) yield the conditional p.m.f. $P[W = w | U \in \text{slab(i)}]$. Finally Eq. (5.26) is the desired p.m.f. of W.

On the number of segments in a rectangle. While we will use the probabilistic characterization of W, the number of points in $\mathcal{R}$, for a close estimate of E[c(U)], the number of segments in $\mathcal{R}$ turns out to be more manageable when we are only concerned with asymptotic performance. On the other hand, Lemma 4.1 allows us to interchange points and segments when we can neglect multiplicative constants.

Given $U \in$ slab(i), we consider the number Q of segments that are in class $\mathcal{J}$ for slab(i) and that are between s_L and U, or U and s_R, (see Figure 5.7), the segment s generated by U and V is not counted, for simplicity. If we define Q_L and Q_R to be the numbers of segments in $\mathcal{R}$ respectively to the left and to the right of s, we can write

$$Q = Q_L + Q_R \quad . \tag{5.33}$$

In Figure 5.7 $Q_L = 2$ and $Q_R = 3$. It is easy to see that Q_R is a geometrically distributed random variable, i.e.

$$P_i[Q_R = q] = p_0(1 - p_0)^q, \quad q = 0,1,\ldots \quad , \tag{5.34}$$

since $Q_R = q$ if and only if s is followed by q nonspanning segments and then by one which is spanning. Q_L is independent of, and distributed as Q_R. Therefore the p.m.f. of Q is

$$P_i[Q = q] = \sum_{h=0}^{q} P_i[Q_L = h]P_i[Q_R = q - h]$$

$$= p_0^2(q + 1)(1 - p_0)^q \quad . \tag{5.35}$$

5.6. Bounds for E[c(U)]

The exact calculation of E[c(U)] would require us to solve quite difficult combinatorial problems, so that we limit ourselves to obtain bounds on either side. First we give a general upper bound which can be used, for a broad class of generator distributions F_{UV}, to show that E[c(U)] is finite and therefore that the algorithm achieves expected linear storage. Second we derive a tighter upper bound and a lower bound for the case of independent generators.

All our bounds will be based on the total probability expansion of E[c(U)] with respect to the possible values of W:

$$E[c(U)] = \sum_{w=1}^{+\infty} E[c(U)|W=w]P[W=w] \quad . \tag{5.36}$$

We already know how to compute P[W=w], and we will use suitable upper and lower bounds for E[c(U)|W=w].

A first upper bound. We have shown in Section 4 that the total number of cuts in processing a rectangle with w points is bounded by f(w), where f is defined by Eqs. (4.1a)-(4.1c). Since the total number of cuts in a rectangle is the sum of the cuts associated with all the endpoints within the rectangle, and c(U) has the same statistical properties for all generators, we conclude that

$$E[c(U)|W=w] \leq f(w)/w \quad . \tag{5.37}$$

Use of bound (5.37) in Eq. (5.36) yields

$$E[c(U)] \leq \sum_{w=1}^{+\infty} P[W=w]f(w)/w \quad . \tag{5.38}$$

Discussion of convergence. It can be easily shown that $f(w)/w = O(\log w)$, and actually that $f(w)/w < \log_2 w$. Then, considering that the series $\sum_{w=2}^{+\infty} 1/(w(\log w)^{\lambda})$ converges if $\lambda > 1$ and diverges if $\lambda \leq 1$, we see that a sufficient condition for the right-hand side of (5.38) to converge is that $P[W = w] = O(1/(w(\log w)^{2+\epsilon}))$ for some $\epsilon > 0$. The intuitive meaning of this condition is that rectangles with many points should not be too likely. This statement, however, holds only in a very weak sense. For, we could have $E[W] = +\infty$, and still a finite $E[c(U)]$! (One instance of this case is $P[W = w] = \theta(1/w(\log w)^3)$.) We have seen in the last section that the dependence of $P[W = w]$ on the distribution of the generators is not straightforward, and this makes it difficult to restate the convergence condition on $P[W = w]$ in terms of $F_{UV}(u,v)$.

We can get more insight by reasoning in terms of the number of segments Q defined in (5.33), whose p.m.f. has a simple expression. We can write

$$E[c(U)] = \sum_{q=0}^{+\infty} P[Q = q] E[c(U) | Q = q]$$

$$= \sum_{|i|=1}^{+\infty} P[U \in \text{slab}(i)] \sum_{q=0}^{+\infty} P_1[Q = q] E[c(U) | Q = q] \quad , \tag{5.39}$$

where we have expanded $P[Q = q]$ according to the total probability theorem, and we have interchanged the order of summation. The last operation is allowed since we deal with series of positive terms. Now, if $Q = 0$, $\mathcal{R}$ contains only one segment and therefore there are no cuts ($E[c(U) | Q = 0]$). For $q > 0$, the average number of cuts $E[c(U) | Q = q]$ is obviously $O(\log q)$ and therefore it can be upper bounded by kq, where $k > 0$ is a suitable constant. So, after using Eq. (5.35), we get

$$\sum_{q=0}^{+\infty} P_i[Q=q]E[c(U)|Q=q] \leq k\, p_{0i}^2 \sum_{q=1}^{+\infty} (q+1)q(1-p_{0i})^q \quad . \tag{5.40}$$

The last series can be summed in closed form yielding

$$\sum_{q=0}^{+\infty} P_i[Q=q]E[c(U)|Q=q] \leq k\, p_{0i}^2 \frac{2}{p_{0i}^3} = \frac{2k}{p_{0i}} \quad . \tag{5.41}$$

Note that we have added the subscript i to p_0, to stress its dependence on the slab. Substitution of bound (5.41) in Eq. (5.39) gives

$$\begin{aligned} E[c(U)] &\leq 2k \sum_{i=1}^{+\infty} 2^{-(|i|+1)} \left(\frac{1}{p_{0i}} + \frac{1}{p_{0(-i)}} \right) \\ &= k \sum_{i=1}^{+\infty} \frac{2^{-i}}{p_{0i}} + k \sum_{i=1}^{+\infty} \frac{2^{-i}}{p_{0(-i)}} \quad . \end{aligned} \tag{5.42}$$

A sufficient condition for convergence of the last term, and therefore for finiteness of E[c(U)], is that

$$p_{0i} > k'\, 2^{-i}\, i(\log i)^{1+\epsilon} \quad , \quad i \geq 1 \tag{5.43}$$

for some $k' > 0$, and some $\epsilon > 0$, and a similar condition for $i \leq 1$. Condition (5.43) is not very restrictive at all. It is intuitive that a lower bound on the probabilities of spanning segments prevents rectangles with many segments. The lower bound is decreasing with $|i|$ because slabs with fewer points have less weight in the contribution to the number of cuts.

Independent generators. To get less crude bounds than the one expressed by Eq. (5.38), we need to further specify the joint distribution F_{UV} of the generators. We introduce

Assumption A3 (independence). U and V are statistically independent, with joint distribution

$$F_{UV}(u,v) = F_U(u)F_V(v) = u\,v, \qquad u,v \in [0,1] \quad . \tag{5.44}$$

The hypothesis of independence considerably simplifies the analysis. Here we see another advantage of defining generators; in fact, as already noted, we could not directly assume independence together with indntical distribution for the bottom B and the top T of segment s, since $B \leq T$.

Let $\mathcal{R}$ be a rectangle processed by the algorithm in slab [m,m'], not necessarily a principal one. In any case $\mathcal{R}$ will be completely below ($m' \leq \frac{1}{2}$) or completely above ($m \geq \frac{1}{2}$) the median $m_0 = \frac{1}{2}$. The median M of the points in $\mathcal{R}$ divides $\mathcal{R}$ into two parts: we call internal the one closer to $m_0 = \frac{1}{2}$, and external the other one. So the internal part is the one above M for rectangles below m_0, and the ones below M for rectangles above m.

Let $z_1,\ldots,z_w$ be the endpoints inside $\mathcal{R}$. Without loss of generality suppose that $\mathcal{R}$ is above m_0, as shown in Figure 5.8. If $z_j \in \mathcal{R}$ is a generator of segment $(z_j,\bar{z}_j)$, this segment is cut by median M if and only if $z_j < M$ and $\bar{z}_j > M$, or $z_j > m$ and $\bar{z}_j < M$. If we call c(z,M) the number of cuts due to M and charged to z, we have:

$$P[c(z;M) = 1] = \begin{cases} M \;, & z > M \\ 0 \;, & z = M \\ 1\text{-}M \;, & z < M \end{cases} \tag{5.45}$$

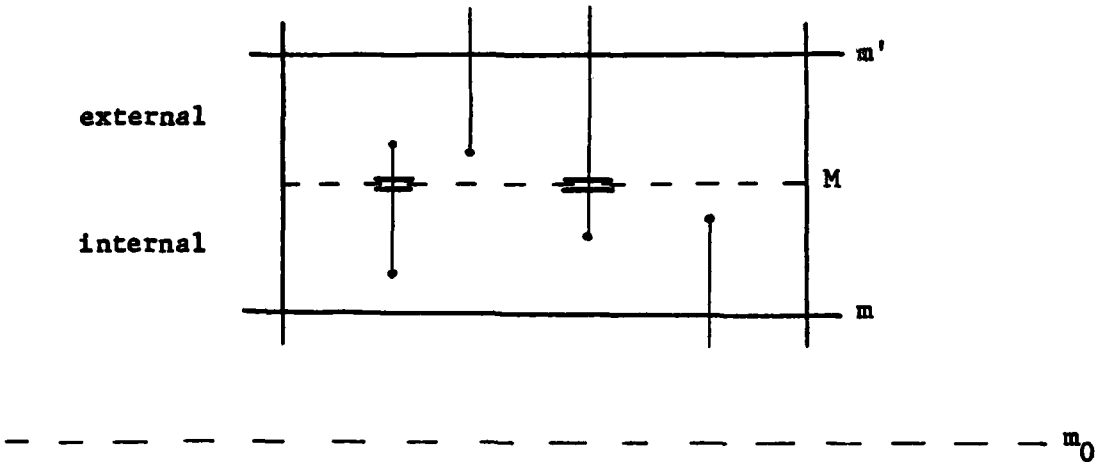


Figure 5.8. A rectangle $\mathcal{R}$ above m_0 divided in internal and external part by median M. A point z forms a segment cut by M if the companion point $\bar{z}$ is on the other side of the median.

In general we have

$$P[c(z;M) = 1] \begin{cases} p_E & , \ z \text{ external} \\ 0 & , \ z = M \\ p_I & , \ z \text{ internal} \end{cases} \tag{5.46}$$

where $p_I \leq p_E$, and $p_I + p_E = 1$. The variable

$$D \triangleq \sum_{j=1}^{w} c(z_j;M) \tag{5.47}$$

is an upper bound to the number of cuts due to M, since, if z and $\bar{z}$ are both in $\mathcal{R}$, the cuts of segment $(z,\bar{z})$ are counted twice in our census scheme. To compute E[D], let us suppose that $\mathcal{R}$ contains w points, n_I in the internal part and n_E in the external one ($w = n_I + n_E + 1$). With this notation

$$E[D] = \sum_{j=1}^{w} E[c(z_j;M)]$$

$$= \sum_{j=1} P[c(z_j;M) = 1] = n_I p_I + n_E p_E \quad . \tag{5.48}$$

If we choose the median M such that $n_I = n_E = (w-1)/2$ when w is odd, and $n_I = n_E + 1 = w/2$ when w is even, we have that

$$E[D] = \begin{cases} (w-1)/2 & , \text{ w odd} \\ w/2 - 1 + p_I, & \text{w even} \end{cases} \tag{5.49}$$

and in general, being $p_I < \frac{1}{2}$,

$$w/2 - 1 \leq E[D] \leq (w-1)/2 \quad . \tag{5.50}$$

The number of cuts in a given rectangle $\mathcal{R}$ equals the number of cuts due to the median M plus the number of cuts occurring in the two regions obtained by cutting $\mathcal{R}$ with M. In the worst case such regions will consist of only one rectangle, while in general spanning segments will form several rectangles in each region. Based on these considerations we can recursively define an upper bound g(w) for the average number of cuts occurring in a rectangle with w points.

$$g(0) = 0, \quad g(1) = 0 \ , \tag{5.51a}$$

$$g(w) = (w-1)/2 + g(n_I) + g(n_E) \quad , \tag{5.51b}$$

where, as we have said, $n_I = \lfloor (w-1)/2 \rfloor$, $n_E = \lceil (w-1)/2 \rceil$. Using the bound $g(w)/w$ for $E[c(U)|W=w]$ in Eq. (5.36) we get

$$E[c(U)] \leq \sum_{w=1}^{+\infty} P[W=w]g(w)/w \quad . \tag{5.52}$$

To get a lower bound for $E[c(U)|W=w]$ we count only the cuts due to the first median in rectangles of principal slabs. We already have a lower bound for E[D], but since D accounts twice for cuts of segments with both endpoints in the same rectangle, we have to subtract this contribution. Since $P[s \in \mathcal{S}_2] = 4^{-(i+1)}$ for slab (i), the probability of segments with both endpoints in the same slab is

$$2 \sum_{i=1}^{+\infty} 4^{-(i+1)} = 1/6 \quad . \tag{5.53}$$

So, with probability 1/6, in E[D] we charge to an endpoint ½ cuts more than due, and we have

$$(w/2 - 1)/w - 1/24 \leq E[c(U)|W=w] \quad , \tag{5.54}$$

whence

$$E[c(U)] \geq \frac{1}{2} - \frac{1}{24} - \sum_{w=1}^{+\infty} P[W=w]/w \quad . \tag{5.55}$$

5.7. Independent Generators: Numerical Results and Simulations

We now specialize the preceding analysis to the case of independent generators in order to get numerical results. We also compare such results with those obtained by actually running the algorithm on suitable random inputs.

In the sequel, Assumptions B1 and A3 are used. Assumption A2 is satisfied as a consequence.

First we plug Eq. (5.44) into Eq. (5.20), and, by summing the resulting geometric series, we get

$$E[c_0] = \sum_{m \in \mathcal{M}} 2(m - m^2) = 13/6 \quad . \tag{5.56}$$

Then, from Eqs. (5.23) and (5.24), or from a direct argument, we can see that, for slab ($\pm$ i), (for brevity we use the symbol ℓ defined as $\ell \triangleq 2^{-(1+|i|)}$): we have

$$p_0 = 2(1 - 2\ell)/(4 - 5\ell) \quad , \tag{5.57a}$$

$$p_1 = 2(1 - \ell)/(4 - 5\ell) \quad , \tag{5.57b}$$

$$p_2 = \ell/(4 - 5\ell) \quad . \tag{5.57c}$$

Using these values for p_0, p_1, p_2 we can compute P[W = w], according to the procedure outlined in Section 5.5. Then we proceed to the numerical evaluation of bounds (5.52) and (5.55) obtaining

$$0.18 = E[c(U)] \leq 0.45 \quad . \tag{5.58}$$

Using these results in Eq. (5.14) we get

$$2.53 \leq E[c] \leq 3.07 \quad , \tag{5.59}$$

and correspondingly Eq. (5.11) becomes

$$5.53 \leq \frac{E[s]}{n} \leq 6.07 \quad . \tag{5.60}$$

The algorithm has been coded, and run on random inputs. The independent endpoints, U and V, have been generated by a congruential method [7]. Particular care has been taken in choosing generators of the same segment, taking them far apart from each other in the "random" sequence given by the congruential method, in order to avoid a possible correlation.

The results of the simulation completely agrees with the theoretical analysis. For n ranging from 300 to 3000, S/n has been always very close to 5.7, so that experiments suggest $S \approx 5.7\ n$.

Another interesting result of simulation is that, on the average, the depth of the tree is about $2 \log_2 n$, actually with a very small variance.

5.8. Average Time

We have seen in Section 5.2 that the average running time of SEARCHTREE is linear if the average number of calls in which a given endpoint is processed, is a (finite) constant.

Let H denote the number of calls in which the generator U is involved. If $U \in$ slab(i), then it will be processed $H_1 = |i| + 1$ times during the formation of principal slabs, and H_2 times during the processing of the interior of slab(i). So $H = H_1 + H_2$ and $E[H] = E[H_1] + E[H_2]$. $E[H_1]$ is finite and can actually be exactly computed as

$$E[H_1] = \sum_{|i|=1}^{+\infty} E[H_1 | U \in \text{slab}(i)] P[U \in \text{slab}(i)]$$

$$= \sum_{i=1}^{+\infty} (i+1)2^{-i} = 3 \qquad (5.61)$$

To show that $E[H_1]$ is in general finite we use the expansion

$$E[H_2] = \sum_{w=1}^{+\infty} E[H_2|W=w]P[W=w] \quad . \tag{5.62}$$

Now, $E[H_2|W=w] = O(\log w)$, since at any call involving the same point the number of points processed is at least halved. Therefore we can apply to the series (5.62) considerations similar to those made in Section 5.6 about Eq. (5.38).

In conclusion, for a wide class of generator distribution, including the case of independent generators, the expected time of SEARCHTREE is linear in the number of segments.

6. CONCLUSIONS

In this thesis we have defined and analyzed a new algorithm for the adjacency map, using a technique that can be easily extended to solve the point location problem in a planar subdivision induced by an embedded straight-line planar graph.

An advantage of this technique lies in the simplicity of the basic idea: the extension of binary search to planar structures. However, dealing with two dimensions requires some care in the implementation of the binary search to keep both the storage and the search time bounded.

From a practical point of view, our algorithm is quite attractive, since it builds in time $O(n\log n)$ a data structure that can be stored in space $O(n\log n)$ and searched in time $O(\log n)$. Moreover all the constants involved are small.

Theoretically though, we know that $O(n)$ space is achievable, and therefore the algorithm is not asymptotically optimal in the worst case. However, while it is possible to find cases in which $S(n) = \theta(n\log n)$, (the reader may try with the set $\{s_j = (-j,j) | j = 1,\ldots,n\}$), such cases require a strong correlation in the position of all the segments. This suggests that if the segments are random, for instance independent of each other, the expected value of $S(n)$ should be linear in n.

The analysis developed in Section 5 confirms the foregoing conjecture and also allows us to estimate the constant k in $E[S(n)] = kn$, when the statistical description of the segments is given. In the case of segments with independent endpoints, we have seen that $k \approx 6$. Comparing this result with the lower bound discussed in Section 4, that implies $k \geq 3$, we may conclude that the storage performance of the algorithm is quite good.

The preprocessing time is also satisfactory, since, after presorting the procedure SEARCHTREE runs in average linear time. Finally the search time is already good ($< 3 \log n + 7$) in the worst case, and simulation suggests it is slightly better ($\approx 2 \log n$) on the average.

Beyond the details of the analysis we may like to capture, on an intuitive basis, the essential features of the algorithm that make its average time and space be linear. We can see the action developing as follows. First the region to be searched is partitioned in $O(\log n)$ strips of plane, the principal slabs in the infinite model, and a point is located in a slab. Then the search proceeds in the interior of the slab. A segment must be represented in the search structure of each of the slabs that intersects, but this can be done at the expense of a small amount of extra storage, since on the average a segment intersects less than 4 slabs. Each principal slab is in turn partitioned in rectangles by the spanning segments. Comparison against spanning segments allows point location in a rectangle. At this point we only need a search structure for each rectangle, since different rectangles do not interfere with each other. The size of such structures depends on the number of points in the rectangle. The key point is now that when the size n of the problem increases, the number of rectangles increases proportionally, but the distribution of the size of the rectnagles does not change. Therefore the average time to build, and the average space to store, the related search-tree are constant. Thus globally, the average complexity is linear. The constant of proportionality is of course related to the distribution of the rectangle size, and this in turn to the frequency of spanning segments. When the endpoints are independent there are many spanning segments, and this accounts for the fact that the constants are small in this case.

As we have seen, the probabilistic analysis of the adjacency map has given considerable insight on the main features of the point-location technique used. The next step should be to extend the analysis to the general case of point location in planar graphs. If, as we conjecture, the complexity of the algorithm is the same in the general case, the proposed technique can be considered a basic tool in computational geometry.

REFERENCES

1. W. Lipski and F. P. Preparata, "Segments, rectangles, contours," J. Algorithms 2, (1981) pp. 63-76.

2. D. T. Lee and F. P. Preparata, "Location of a point in a planar subdivision and its applications," SIAM J. Comput. 6, No. 3 (1977) pp. 594-606.

3. F. P. Preparata, "A new approach to planar point location," SIAM J. Comput., No. 3 (1981) pp. 473-482.

4. R. J. Lipton and R. E. Tarjan, "Application of a planar separator theorem," Proc. of the 18th Symp. on Found. of Comp. Sci., Providence, RI, October 1977, pp. 162-170.

5. D. G. Kirkpatrick, "Optimal search in planar subdivision," University of British Columbia, Vancouver, British Columbia, Department of Computer Science; Manuscript, 1979.

6. E. M. Reingold, J. Nievergelt and N. Deo, Combinatorial Algorithms, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1977.

7. D. E. Knuth, The Art of Computer Programming, Vol. 2, Addison-Wesley, 1969.

END

FILMED

3-83

DTIC