Intern. J. Computer Math., Vol. 34, pp. 129–144
Reprints available directly from the publisher
Photocopying permitted by license only

RANKING INTERVALS UNDER VISIBILITY CONSTRAINTS*

HERBERT EDELSBRUNNER

Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801, USA

MARK H. OVERMARS

Department of Computer Science, University of Utrecht, P.O. Box 80.089, 3508 TB Utrecht, The Netherlands

EMO WELZL

Institut für Informatik, Fachbereich Mathematik, Freie Universität Berlin, Arnimallee 2–6, D-1000 Berlin 33, FRG

IRITH BEN-ARROYO HARTMAN and JACK A. FELDMAN

IBM Scientific Center, Haifa, Israel, 32000

(*Received 12 September 1989*)

Let S be a set of n closed intervals on the x-axis. A ranking assigns to each interval, s, a distinct rank, $\rho(s) \in \{1, 2, \ldots, n\}$. We say that s can *see* t if $\rho(s) < \rho(t)$ and there is a point $p \in s \cap t$ so that $p \notin u$ for all u with $\rho(s) < \rho(u) < \rho(t)$. It is shown that a ranking can be found in time $O(n \log n)$ such that each interval sees at most three other intervals. It is also shown that a ranking that minimizes the average number of endpoints visible from an interval can be computed in time $O(n^{5/2})$. The results have applications to intersection problems for intervals, as well as to channel routing problems which arise in layouts of VLSI circuits.

KEY WORDS: Algorithms, data structures, intervals, ranking, vertical visibility, partial orders, maximum matching in a bipartite graph, one sided channel routing.

C.R. CATEGORIES: F.2.2, G.2.1, G.2.2.

1. INTRODUCTION

Let S be a set of n closed intervals on the real line. We define a *ranking*, ρ, as a bijective mapping from S to $\{1, 2, \ldots, n\}$. A ranking of S can be visualized by

*Part of this research was performed when the first and the third author were still with the Institute für Informationsverarbeitung, Technische Universität Graz, Austria. Research of the first author was supported by the National Science Foundation under grant CCR-8714565. Research of the third author was supported by the Deutsche Forschungsgemeinschaft under Grant We 1265/1-1.

drawing each interval, s, as a horizontal line segment, $\bar{s}$, with height $\rho(s)$ and so that s is its vertical projection onto the x-axis.

We define the notion of visibility in a ranked set of intervals. For s and t in S, we say that $\bar{s}$ *sees* $\bar{t}$ (or $\bar{t}$ is *visible* from $\bar{s}$) if $\rho(s)<\rho(t)$ and there exists a point $p \in s \cap t$ such that $p \notin u$ for all u with $\rho(s)<\rho(u)<\rho(t)$. More specifically, $\bar{s}$ *sees* the left endpoint of $\bar{t}$ (the left endpoint of $\bar{t}$ is *visible* from $\bar{s}$) if $\rho(s)<\rho(t)$, s contains the left endpoint of t, and no interval u with $\rho(s)<\rho(u)<\rho(t)$ contains the left endpoint of t. Analogously for the right endpoint of t.

Intuitively, "$\bar{s}$ sees $\bar{t}$" means that there is a position on $\bar{s}$ such that if one stands at this position and looks vertically upward then one sees $\bar{t}$. See Figure 1.1 for an example.

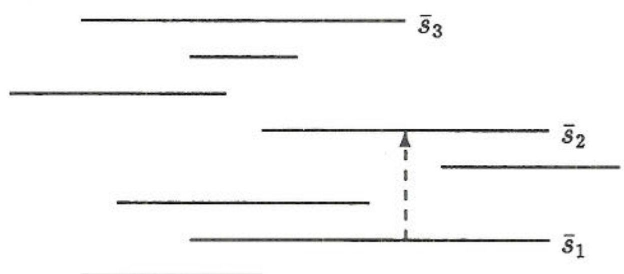

Figure 1.1 Vertical visibility in a ranking: $\bar{s}_1$ sees $\bar{s}_2$ but cannot see $\bar{s}_3$.

In this paper we study some aspects of rankings and visibility. In Section 2 we look at rankings where the maximum number of line segments any one line segment sees is small. We show that for every finite set of intervals there exists a ranking such that any line segment sees at most three other line segments. An algorithm will be given that computes such a ranking in time $O(n \log n)$ if the number of intervals is n. In Section 3 we apply this result to obtain a new data structure for the interval intersection problem: store a finite set of intervals so that for a query interval the intervals it intersects can be reported efficiently. In Section 4 we give a method for minimizing the average number of visible endpoints from a line segment. We show that this problem has applications to one sided channel routing. We prove that the problem can be reduced to computing a maximum matching in a bipartite graph which is known to be solvable in time $O(n^{5/2})$. Finally, Section 5 gives some concluding remarks and directions for further research.

Throughout this paper we will use the following notation. S is a set of n intervals. The intervals are denoted by s, t, u and v, with or without subscripts; the corresponding horizontal line segments are denoted by $\bar{s}$, $\bar{t}$, $\bar{u}$ and $\bar{v}$. It will be important to distinguish between the left endpoint of s, ℓ_s, and the left endpoint of $\bar{s}$, $\ell_{\bar{s}}$. For the right counterparts we write r_s and $r_{\bar{s}}$. Note that if $s \neq t$ then $\ell_{\bar{s}} \neq \ell_{\bar{t}}$ because $\rho(s) \neq \rho(t)$ but $\ell_s = \ell_t$ may hold. So, in a ranking, a line segment may see $\ell_{\bar{s}}$ but not $\ell_{\bar{t}}$, although $\ell_s = \ell_t$.

2. MAXIMUM INDIVIDUAL VISIBILITY

The goal of this section is to describe a way to rank a set of intervals so that the visibility is constant for each line segment. We do this in three stages. First, we describe a certain way to rank intervals, then we prove that in a thus obtained ranking any line segment sees at most three other line segments, and finally we describe an algorithm that implements the construction in time $O(n \log n)$.

2.1 A Lower Bound

If a line segment sees k endpoints then it can see at most $k+1$ other line segments. We show that $k \leqq 2$ is always achievable. Let us first make sure that this is best possible.

LEMMA 2.1 *There exists a set of six intervals so that in every ranking of this set there is a line segment that sees at least two endpoints.*

Proof See Figure 2.1 for the set of six intervals for which we prove the

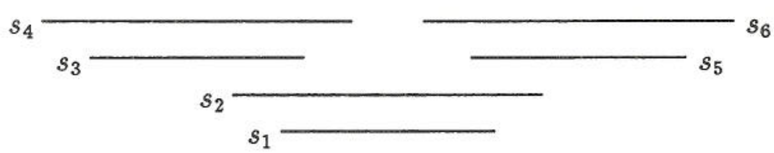

Figure 2.1 An example of six intervals so that every ranking has one line segment that sees at least two endpoints.

assertion. Assume that ρ is a ranking in which no line segment sees more than one endpoint. Then $\rho(s_3) < \rho(s_4)$, since otherwise $\bar{s}_4$ sees ℓ_3 and some other endpoint (either r_3 or the leftmost endpoint of the line segments blocking the view to r_3).* By a similar consideration we obtain $\rho(s_1) < \rho(s_2)$. If $\rho(s_2) < \rho(s_3)$, then $\bar{s}_2$ sees r_3 and r_4. So the only possibilities for s_2 to be ranked among s_3 and s_4 are

$$\rho(s_3) < \rho(s_2) < \rho(s_4) \quad \text{and} \quad \rho(s_3) < \rho(s_4) < \rho(s_2).$$

The ranking of s_1 is now determined; in the first case we have $\rho(s_1) < \rho(s_3) < \rho(s_2) < \rho(s_4)$ and in the second case $\rho(s_3) < \rho(s_1) < \rho(s_4) < \rho(s_2)$. This can be seen by simply going through the two possibilities for the first case and the three possibilities for the second case. In either case, $\bar{s}_1$ sees r_3 or r_4. By a symmetric argument for $\{s_1, s_2, s_5, s_6\}$ it follows that $\bar{s}_1$ sees ℓ_5 or ℓ_6; so $\bar{s}_1$ sees two endpoints after all. □

Remarks

1) By checking all possibilities with a computer, one can show that six intevals are necessary to prove the lemma, that is, for five or fewer intervals it is

*We write ℓ_i and r_i short for $\ell_{\bar{s}_i}$ and $r_{\bar{s}_i}$.

always possible to rank them so that no line segment sees more than one endpoint.

2) Lemma 2.1 does not quite imply that there is a set of intervals so that every ranking contains an interval that sees at least three other intervals. Still, this is true but we leave the argument to the interested reader.

2.2 Constructing a Ranking

The ranking we design consists of a sequence of layers, $(\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_m)$, where each $\mathcal{L}_i$ is interchangeably treated as a set or sequence of intervals or of line segments, as is convenient. The intervals in layer $\mathcal{L}_i$ will be assigned higher ranks than the intervals in $\mathcal{L}_j$ for $i<j$. The idea is that we make the layers such that a line segment in layer $\mathcal{L}_i$ can only see endpoints in its own layer and in one other layer above it.

The first layer, $\mathcal{L}_1$, is constructed as follows. Let s_1 be the interval in S with $\ell_{s_1} = \min\{\ell_s | s \in S\}$ and r_{s_1} maximal among these; s_1 is the first interval in $\mathcal{L}_1$. Assume we have found the first j intervals, $s_1, s_2, \ldots, s_j$, in $\mathcal{L}_1$. Now we consider the set of intervals $s \in S$ with $\ell_s \in s_j$ and $r_s \notin s_j$. If this set is empty the first layer is complete. Otherwise, take s_{j+1} from this set such that $r_{s_{j+1}}$ is maximal. In this way we continue until we can no longer extend the first layer. Assume $\mathcal{L}_1 = (s_1, s_2, \ldots, s_k)$. We define $\rho(s_i) = n-k+i$ for $1 \leqq i \leqq k$. So the intervals forming the first layer are assigned the highest ranks, where the first such intervals gets the lowest of those ranks and the last gets the highest. Setting $S' = S - \mathcal{L}_1$, we construct the next layers in the same way from S'. See Figure 2.2 for an example of the ranking we get.

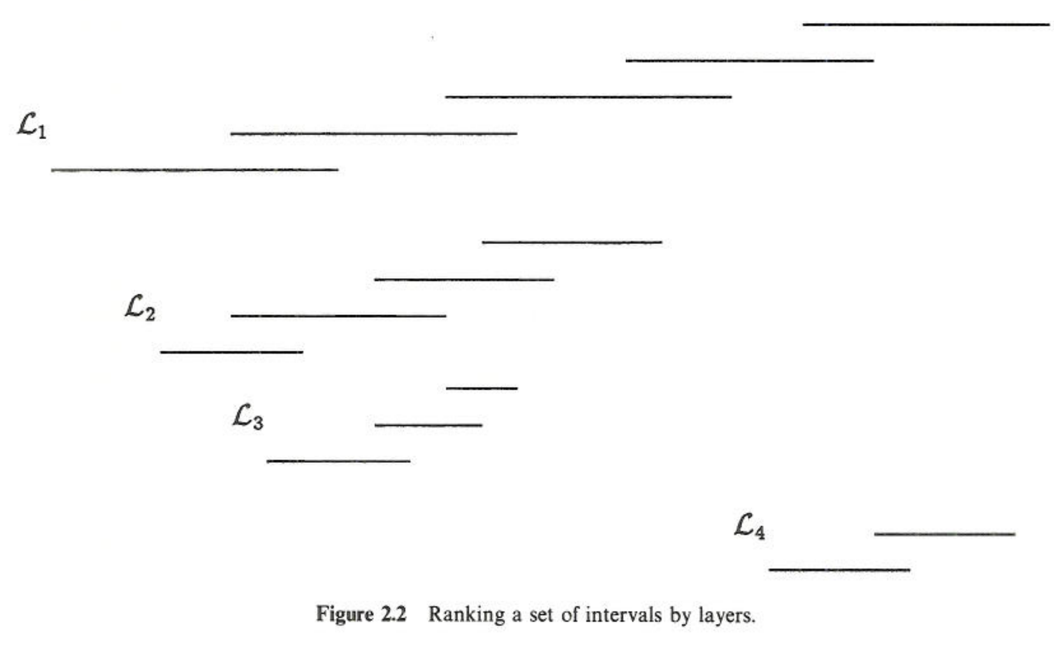

Figure 2.2 Ranking a set of intervals by layers.

2.3 Analysis of the Ranking

To prove that every line segment sees at most two endpoints we need a few simple observations. First, note that for each layer $\mathscr{L}_i$ the union $I_i = \bigcup_{s \in \mathscr{L}_i} s$ is a single interval. Furthermore, we have $I_i \subseteq I_j$ or $I_i \cap I_j = \emptyset$ if $j < i$. In the latter case, I_i lies to the right of I_j. This implies that a line segment $\bar{s}$ in layer $\mathscr{L}_i$ can only see line segments in its own layer and in at most one other layer, namely layer $\mathscr{L}_j$ with largest j so that $j < i$ and $I_i \subseteq I_j$.

Let us now consider a layer $\mathscr{L}_i = (s_1, s_2, \ldots, s_{k_i})$. Clearly every $\bar{s}_l$, $1 \leqq l \leqq k_i - 1$, sees only one endpoint in the same layer, namely the left endpoint of $\bar{s}_{l+1}$, and $\bar{s}_{k_i}$ sees no endpoint in $\mathscr{L}_i$. Let $\mathscr{L}_j = (t_1, t_2, \ldots, t_{k_j})$ be the lowest layer above $\mathscr{L}_i$ with $I_i \subseteq I_j$ (if it exists). The only left endpoint of a line segment in $\mathscr{L}_j$ visible from below is that of $\bar{t}_1$, followed by the right endpoints of $\bar{t}_1, \bar{t}_2, \ldots, \bar{t}_{k_j}$, in this order from left to right. Thus, if a line segment $\bar{s}$ in $\mathscr{L}_i$ sees at least two endpoints in $\mathscr{L}_j$ then it must either see the left and the right endpoint of $\bar{t}_1$ or two consecutive right endpoints. The former contradicts the way t_1 is chosen. In the latter case, let $\bar{s}$ see the right endpoints of $\bar{t}_l$ and $\bar{t}_{l+1}$. By the way t_{l+1} is chosen after t_l it follows that $r_{t_{l+1}} = r_s$. But then, $\bar{s}$ can see the right endpoint of $\bar{t}_{l+1}$ only if $\bar{s}$ is the last line segment in $\mathscr{L}_i$, and therefore it sees no endpoint within its own layer. Consequently, $\bar{s}$ sees at most two endpoints in either case. This proves the main result of this section.

THEOREM 2.2 *For every finite set of intervals there exists a ranking in which every line segment sees at most two endpoints.*

2.4 Implementing the Construction

In the remainder of this section we show how to compute such a ranking efficiently. To construct the layers we need to be able to perform two operations efficiently:

1) find an interval with smallest left endpoint, and if there are several of them, find among those the one with largest right endpoint, and

2) find the interval with left endpoint in some interval s and right endpoint as large as possible.

Both questions can be efficiently answered using a minimum height binary tree which we now define. The leaves of the tree are in one-to-one correspondence with the intervals so that the inorder of the leaves gives the sequence of intervals sorted from left to right by left endpoint, and if two intervals have the same left endpoint then the one with the larger right endpoint precedes the other. Each internal node, κ, stores an interval, $i(\kappa)$, with largest right endpoint stored in a leaf below κ. More precisely, if κ has children μ and ν then $i(\kappa) = i(\mu)$ if $r_{i(\nu)} < r_{i(\mu)}$, $i(\kappa) = i(\nu)$ if

$r_{i(\mu)} < r_{i(\nu)}$, and $i(\kappa)$ is any one of the two if $r_{i(\mu)} = r_{i(\nu)}$. This tree can be constructed in time $O(n \log n)$.

As to operation 1, the interval with smallest left endpoint (and if this is ambiguous, the one with largest right endpoint) is stored in the leftmost leaf of the tree and can thus be retrieved in constant time. To understand how operation 2 can be executed efficiently notice that all intervals whose left endpoints lie in some query interval s are stored in a consecutive list of leaves. These leaves define $O(\log n)$ subtrees, and for each such subtree an interval with largest right endpoint is stored in its root. In time $O(\log n)$ we can find these roots and select one interval, among the $O(\log n)$ intervals, with largest right endpoint.

The algorithm works as follows. As initialization we construct the above tree for the set S of intervals. We continue constructing layers as long as the tree is not yet empty. Each layer is constructed as follows. Using operation 1 we identify the interval, s_1, with smallest left endpoint and remove it from the tree; s_1 is the first interval in the layer. Assume we have constructed the layer up to interval s_j. Using operation 2 we determine the interval with largest right endpoint whose left endpoint falls into s_j. If such an interval does not exist then the current layer is complete. Otherwise, call this interval s_{j+1} and add it to the current layer. We delete s_{j+1} from the tree and continue.

It remains to explain how an interval, s, can be deleted from the tree. Structurally, we delete the leaf that stores s plus we delete its ancestors bottom-up until we arrive at an ancestor, μ, that still has one child that is not an ancestor of the deleted leaf. Starting at μ we continue moving up, and for each node on the way we recompute the stored interval from the intervals of its children until we reach the root or a node whose interval remains unchanged. Clearly, a deletion does not take more than $O(\log n)$ time and does not increase the future search time. Altogether we do at most $2n$ search operations (only n are successful) and n deletions which implies the following result.

THEOREM 2.3 *Given a set of n intervals, a ranking where no interval sees more than two endpoints can be constructed in time* $O(n \log n)$ *using* $O(n)$ *storage.*

3. AN ALGORITHMIC APPLICATION

In this section we show how a ranking of a set of intervals can be used to solve the following search problem:

> store a given set of n intervals, S, in some data structure, and for each later specified query interval, q, report the intervals in S it intersects.

A data structure for this problem can be based on the ranking method of Section 2. When we explain how the intervals that intersect q can be determined, using this data structure, we first consider the special case where q is a point and later extend the search algorithm to intervals.

3.1 *The Data Structure*

Let ρ be a ranking of S as described in the previous section and call a point p on the real line a *breakpoint* if p is an endpoint of an interval s so that no interval t with $\rho(t)<\rho(s)$ contains p. Based on ρ we construct a directed graph $\mathscr{R}=(S,A)$ with $(s,t)\in A$ if and only if s sees t. The data structure consists of the graph $\mathscr{R}$ (arcs are represented by pointers) in addition to a linear array that stores the breakpoints sorted from left to right. Let $p_1,p_2,\ldots,p_m, m\leqq 2n$, be the sorted sequence of breakpoints. With each gap between two adjacent breakpoints, (p_i,p_{i+1}), we store a pointer to the interval (node in $\mathscr{R}$) with lowest rank that contains it (if such an interval exists). The lowest rank interval that contains a breakpoint p_i is either the interval of gap (p_{i-1},p_i) or that of (p_i,p_{i+1}), if they exist.

Here are some properties of this data structure. By Theorem 2.2, the outdegree of each node in $\mathscr{R}$ is at most three. This implies that $O(n)$ storage suffices for $\mathscr{R}$. Since the linear array also takes only $O(n)$ storage this is true for the entire data structure. $\mathscr{R}$ is planar because arcs in $\mathscr{R}$ correspond to vertical visibilities, but this will not be important.

3.2 *Constructing the Data Structure*

Given ρ, the data structure can be constructed in time $O(n)$ as follows. Process the intervals in order of decreasing rank. At any point in time we store a linked linear list of the breakpoints for the current set of intervals, plus a pointer for each gap between adjacent breakpoints as described for the linear array above. To add a new interval, s, we locate its two endpoints in the list—both are new breakpoints that need to be added to the list. The old breakpoints that are contained in s (there are at most two) have to be deleted from the list. At the same time we find the intervals visible from s and add appropriate arcs to $\mathscr{R}$.

Unless the interval is the rightmost in its layer, constant time suffices to locate its endpoints if we start at the breakpoint that corresponds to one of the endpoints of the preceding interval. We use the following procedure to locate the right endpoint, r, of the rightmost interval in some layer $\mathscr{L}_i$—the left endpoint takes only constant time once the right endpoint is located. Start at the leftmost breakpoint, b, that corresponds to an endpoint of the preceding layer (we assume that a special access pointer to b is set up at the time it is created) and walk to the right in the list until the location of r is found.

We argue that the procedure for locating rightmost right endpoints takes only $O(n)$ time in total. Consider the list between b and r; it is split into a left and a right part by the leftmost breakpoint b' of an endpoint in $\mathscr{L}_i$. The left part will remain untouched for the rest of the algorithm which implies that the total size of all left parts is $O(n)$. The right part will be deleted completely when the intervals of $\mathscr{L}_i$ are added. Since we cannot delete more than we construct, the total size of all right parts is also $O(n)$.

After processing all intervals the graph $\mathscr{R}$ is complete and we just need to copy the linked list of breakpoints to a linear array.

3.3 *Searching in the Data Structure*

First we consider the problem of reporting all intervals of S that contain a point q;

let their number be k and denote them by $s_1, s_2, \ldots, s_k$ in order of increasing rank. The first step is to locate q in the linear array of breakpoints, that is, to determine the largest breakpoint $p_i \leqq q$. If p_i does not exist then $k=0$ and we are done. Otherwise, the lowest rank interval that contains q, s_1, is either stored with (p_{i-1}, p_i) or with (p_i, p_{i+1}). Now we use $\mathscr{R}$ to report all the other intervals that contain q in order of increasing rank. Assume that we reached some interval s_j. After reporting s_j we examine the (at most three) outgoing arcs of node s_j in $\mathscr{R}$. We are done if s_j has no outgoing arc or if no interval visible from s_j contains q—in these cases $k=j$. Otherwise, determine the interval of the at most three visible from s_j that contains q and has lowest rank of those that contain q; this is s_{j+1}.

Next, let $q=[a,b]$ be an interval and consider the problem of reporting all intervals in S that intersect q. Such an interval

i) contains at least one of the two endpoints of q, or

ii) both of its endpoints lie in q.

To find the intervals that satisfy (i) we apply the algorithm of the previous paragraph for points a and b and mark the nodes of $\mathscr{R}$ thus visited. Each interval that satisfies (ii) is reachable in $\mathscr{R}$ by a path starting at a marked node or by a path starting at a gap in the linear array that is contained in q. To find all such intervals we thus go through the sequence of gaps contained in q and mark all nodes of $\mathscr{R}$ pointed to by these gaps. In the final step, we put all marked nodes onto a stack and process each one as follows until the stack is empty. Take an interval s off the stack, report it, and test each of the at most three intervals visible from s. If such an interval is yet unmarked and intersects q then mark it and push it onto the stack.

3.4 The Analysis

As mentioned above, the data structure for a set S of n intervals takes $O(n)$ storage and $O(n)$ time for construction if the ranking, ρ, is given. By Theorem 2.3, it takes $O(n \log n)$ time to construct it from S. The time to find all k intervals that contain a point is $O(\log n)$ for searching the linear array plus $O(k)$ to walk through $\mathscr{R}$. Similarly, the time to report all k intervals that intersect a query interval $q=[a,b]$ is $O(\log n + k)$, but a brief argument is required.

First, we mark the intervals that contain a or b or both; this takes time $O(\log n)$ plus time proportional to the number of marked intervals (each interval is marked at most twice). Next, we mark the intervals identified by the gaps contained in q. In this step, a single interval can be marked an arbitrary number of times. Still, the total number of different intervals marked in this step is at least half the number of gaps visited and thus at least half the total number of marks applied. The rest of the algorithm takes $O(k)$ time because only intervals that intersect q are pushed onto the stack, and each such interval is processed only once and in constant time. This implies the following result.

THEOREM 3.1 *The above data structure stores a set of n intervals in $O(n)$ storage* *and can be constructed in time* $O(n \log n)$*; it can be used to report the* k *intervals that intersect a query interval* q *in time* $O(\log n + k)$.

Remark The $\log n$ term in the query time is caused solely by the initial binary search step that locates the endpoints of the query interval in the linear array of breakpoints. In applications where the endpoints are chosen from some bounded universe $U = \{0, 1, \ldots, u-1\}$ of integers faster search methods exist. For example, if u is reasonably small we can construct a linear array with index set U in which each gap is represented by a sequence of entries all with the same pointer to a node in $\mathscr{R}$. With direct access in this array we improve the query time to $O(k+1)$ with storage going up to $O(u+n)$. Complexities between this extreme and the one of Theorem 3.1 can be obtained using the y-fast trie of Willard [14] or the q-fast trie of Willard [15]. The y-fast trie gives query time $O(\log\log n + k)$ and storage $O(n)$, but a large amount of preprocessing is required because the method is based on perfect hashing. The q-fast trie solves the problem in query time $O(\sqrt{\log n} + k)$ and storage $O(n)$ and takes only $O(n \log n)$ time for construction. These results improve the $O(\frac{\log u}{\log \log u} + k)$ solution of [6] which works for the special case where the query interval is a point.

4. MINIMIZING THE AVERAGE ENDPOINT VISIBILITY

We have seen that for every set of intervals we can find a ranking such that at most two endpoints are visible from each line segment. In this section we address the problem of minimizing the average number of endpoints visible from each line segment. Of course, this is equivalent to minimizing the total number of endpoint visibilities and, since each endpoint is seen at most once, to maximizing the number of endpoints that are not visible from any line segment.

We say that for a ranking ρ of a set S of intervals, an endpoint $\ell_{\bar{s}}$ (or $r_{\bar{s}}$) is *exposed* if no interval t with $\rho(t) < \rho(s)$ contains ℓ_s (or r_s). Note that if $\ell_{\bar{s}}$ (or $r_{\bar{s}}$) is exposed then ℓ_s (r_s) is a breakpoint in the terminology of Section 3. Define $L_S = \{\ell_{\bar{s}} | s \in S\}$ and $R_S = \{r_{\bar{s}} | s \in S\}$, the sets of left and right endpoints of the line segments. A subset I of $L_S \cup R_S$ is *exposed* if all elements in I are exposed, and I is *exposable* if there exists a ranking ρ of S in which I is exposed. In this terminology, this section studies the problem of finding a maximum exposable subset of $L_S \cup R_S$.

4.1 Motivation from One-sided Channel Routing

The problem of finding a maximum exposable subset of $L_S \cup R_S$ arises in the *layout* of *VLSI* circuits. Layouts of VLSI circuits are composed of placed modules and their interconnections by wires. The routing of these wires can be done in two layers, where the change from one layer to another necessitates a *via.* A good layout practice is to minimize the number of vias.

Take, for example, the routing situation shown in Figure 4.1(a). The rectangles symbolize modules. The modules contain several labelled points (which are called

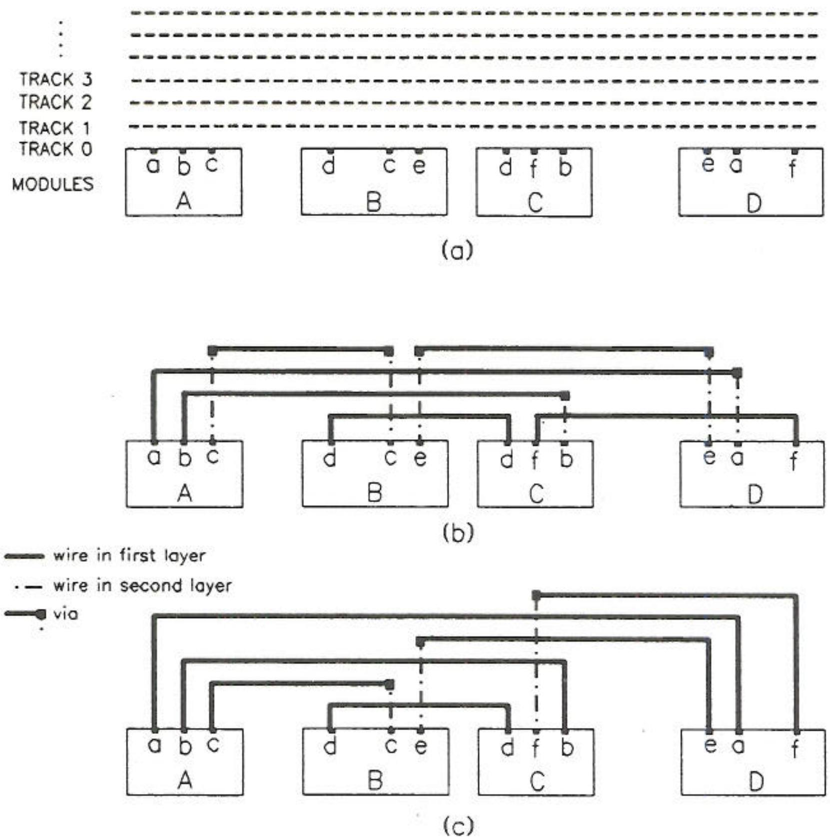

Figure 4.1 (a) One sided channel routing problem, (b) a possible solution, and (c) an optimal solution.

terminals and are labelled in the figure with lower case letters). A set of terminals which have a common label is called a *net*. All terminals belonging to the same net must be electrically interconnected by wires. These interconnections must take place on the tracks running above the modules, where horizontal wires are allowed in one layer, and vertical wires are allowed in both. Wires that belong to distinct nets may not cross at the same layer. A net may contain any number of terminals. The most common distinction is between *2-point* nets as opposed to nets with more than two terminals, that are often referred to as *multi-point* nets. This problem is known as the *one sided channel routing problem* (see, for example, [8] or [13] or [1]). We deal here with the special case where all nets are 2-point nets. Figure 4.1(b) illustrates a possible solution requiring 6 vias. The optimal solution, however, needs fewer vias as shown in Figure 4.1(c). Note that if we represent each 2-point net by an interval, then every unexposed endpoint in a given ranking requires a via. Therefore, the problem of via minimization for this situation is equivalent to the problem of finding a ranking with a maximum exposed subset.

We remark that the intervals which correspond to 2-point nets all have distinct endpoints, but in the following sections we do not make this assumption.

4.2 *Average Versus Worst-case Visibility*

It turns out that the problem of this section is quite different from minimizing the maximum individual visibility. Indeed, the algorithm in Section 2 can produce rankings with far fewer than the maximum number of exposed endpoints. Consider three sets of intervals, $T=\{t_1, t_2, \ldots, t_n\}$, $U=\{u_1, u_2, \ldots, u_n\}$ and $V=\{v_1, v_2, \ldots, v_n\}$, such that no two endpoints are the same, and

1) $t_i \subseteq t_{i+1}$, $u_i \subseteq u_{i+1}$, and $v_i \subseteq v_{i+1}$, for $1 \leqq i < n$,
2) $t_n \cap u_n = \emptyset$, and
3) for all $1 \leqq i \leqq n$, $\ell_{v_i} \in t_1$ and $r_{v_i} \in u_1$

(see Figure 4.2). Now apply the algorithm of Section 2 to compute a ranking for $S=T \cup U \cup V$ (see Figure 4.3). In such a ranking the number of exposed endpoints is $2n+2$ while the ranking indicated in Figure 4.2 implies that a set of $4n$ endpoints is exposable. Conversely, if we remove an interval from T and U each, every ranking that maximizes the number of exposed endpoints is equivalent to the one in Figure 4.2. This is because the only way to expose another endpoint (a right endpoint in T or a left endpoint in U) is to move a line segment of T or U below all line segments of V. If we do this with a line segment in T, say, then we lose n endpoints at once (the left endpoints in V), and this cannot be compensated for even if we move all line segments of T below the ones of V. But in a ranking as indicated in Figure 4.2, the topmost line segment of V sees $2n-2$ endpoints.

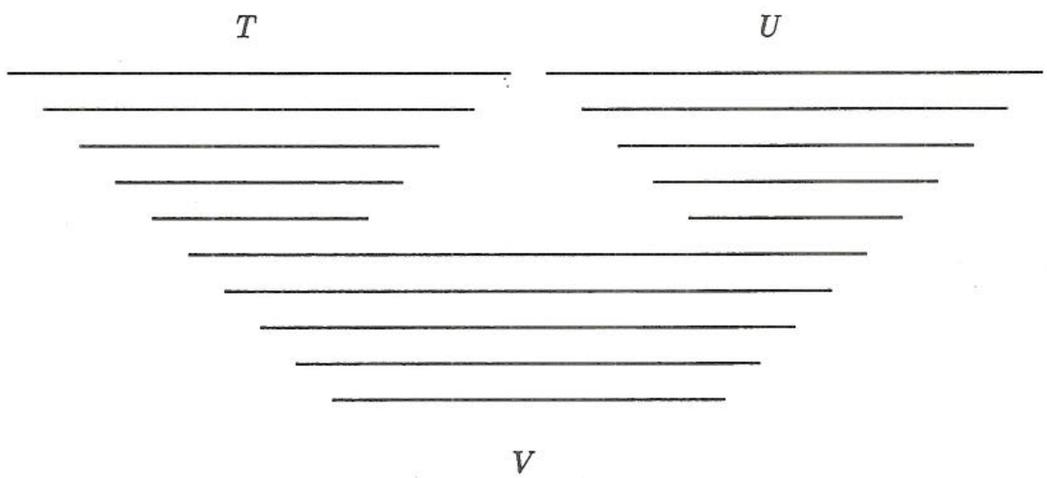

Figure 4.2 The sets T, U and V for $n=5$.

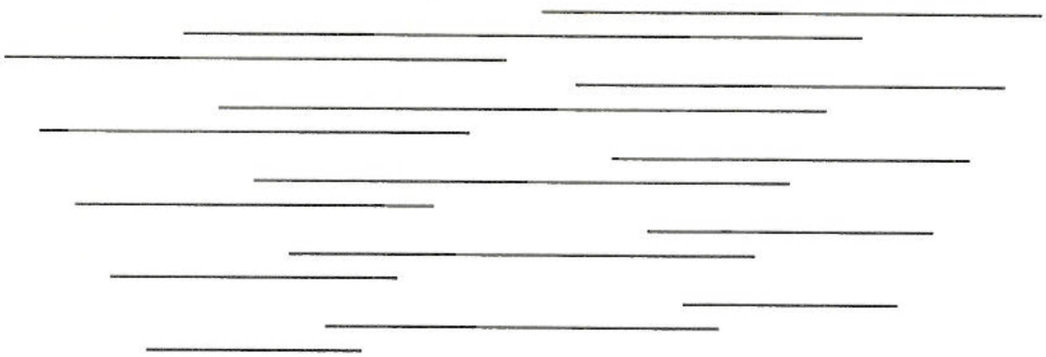

Figure 4.3 The ranking of $T \cup U \cup V$ as produced by the algorithm in Section 2.

4.3 *Excluding Endpoints*

For the remainder of the section, let S be a set of intervals as usual. We start with two basic observations.

Figure 4.4 An excluding pair.

First, consider two intervals, $s, t \in S$, so that $\ell_s < \ell_t < r_s < r_t$ (see Figure 4.4). If t follows s in the ranking (the case shown in Figure 4.4), then $\ell_{\bar{t}}$ is not exposed independent of the ranking of the other intervals. If s follows t, then $r_{\bar{s}}$ is not exposed. Hence there is no ranking with both $\ell_{\bar{t}}$ and $r_{\bar{s}}$ exposed; we say that $\ell_{\bar{t}}$ and $r_{\bar{s}}$ *exclude* each other.

Second, let s and t be two intervals with $s \subseteq t$ and let ρ be a ranking with $\rho(s) > \rho(t)$. If we move $\bar{s}$ right below $\bar{t}$ (that is, s gets the rank of t and the ranks of intervals u with $\rho(t) \leqq \rho(u) < \rho(s)$ increase by one), then the number of exposed endpoints does not decrease. Hence, we may as well let s precede t. A ranking ρ for which $\rho(s) \leqq \rho(t)$ if $s \subseteq t$ is true is called *inclusion consistent.*

Of course, when we restrict ourselves to inclusion consistent rankings, we immediately exclude some of the endpoints from the game. If $s \subseteq t$ and $\ell_s = \ell_t$ then $\ell_{\bar{t}}$ is not exposed in any inclusion consistent ranking; we say that $\ell_{\bar{t}}$ is *shy.* Analogously, if $s \subseteq t$ and $r_s = r_t$ then $r_{\bar{t}}$ is shy. Let $L \subseteq L_S$ and $R \subseteq R_S$ be the sets of endpoints that are not shy. The *exclusion graph* of S is the bipartite undirected graph $\mathscr{G} = (L \cup R, A)$ with $\{x, y\} \in A$ if $x \in L$, $y \in R$, and x and y exclude each other.

Since two endpoints, x and y, that exclude each other cannot be exposed in the same ranking of $S, I \subseteq L \cup R$ is independent* in $\mathscr{G}$ whenever I is exposable. We demonstrate below that also the reverse is true, that is, every independent node set I of $\mathscr{G}$ is exposable. In addition, we give a description of all inclusion consistent rankings for which a given independent set I is exposed.

4.4 *Partial Order Constraints for Rankings*

We start with a definition. For I a subset of $L \cup R$ we define the relation $\mathscr{I} \subset S \times S$ so that $(s,t) \in \mathscr{I}$ if

i) $s \subseteq t$, or

ii) $\ell_{\bar{s}} \in I$ and $\ell_{\bar{s}}$ and $r_{\bar{t}}$ exclude each other, or $r_{\bar{s}} \in I$ and $\ell_{\bar{t}}$ and $r_{\bar{s}}$ exclude each other.

A pair (s,t) that satisfies (i) is called an *inclusion pair* and one that satisfies (ii) is called an *exclusion pair*. Note that no pair in $\mathscr{I}$ can be an inclusion and an exclusion pair at the same time.

It is clear that the definition of $\mathscr{I}$ captures the necessary condition for an inclusion consistent ranking with I exposed since t must follow s in any inclusion consistent ranking that exposes I if $(s,t) \in \mathscr{I}$. We continue by first showing that $\mathscr{I}$ is acyclic if and only if I is independent. Second, we prove that I is exposable if and only if $\mathscr{I}$ is acyclic.

LEMMA 4.1 *$\mathscr{I}$ is acyclic if and only if $I \subseteq L \cup R$ is independent in $\mathscr{G}$.*

Proof ($\Rightarrow$) If I is not independent then there are nodes $\ell_{\bar{s}}$ and $r_{\bar{t}}$ in I with $\{\ell_{\bar{s}}, r_{\bar{t}}\} \in A$. By definition of A, $\ell_{\bar{s}}$ and $r_{\bar{t}}$ exclude each other. It follows that both (s,t) and (t,s) are exclusion pairs which thus constitutes a cycle in $\mathscr{I}$.

($\Leftarrow$) In a first step we show that if $(s,t) \in \mathscr{I}$ and (t,u) is an inclusion pair in $\mathscr{I}$, then $s \neq u$ and $(s,u) \in \mathscr{I}$. Assume first that $s=u$. But then $t \subseteq s$ which contradicts $(s,t) \in \mathscr{I}$ because $s \neq t$. If (s,t) is an inclusion pair, then (s,u) is also an inclusion pair and therefore in $\mathscr{I}$. Finally, if (s,t) is an exclusion pair, then either $\ell_{\bar{s}} \in I$ and $\ell_t < \ell_s < r_t < r_s$, or $r_{\bar{s}} \in I$ and $\ell_s < \ell_t < r_s < r_t$; assume the former without loss of generality. If $s \subseteq u$, then (s,u) is an inclusion pair. Otherwise, $\ell_u \leqq \ell_t < \ell_s < r_t \leqq r_u < r_s$, and (s,u) is an exclusion pair. In both cases $(s,u) \in \mathscr{I}$ as claimed.

We assume that I is a set of independent nodes in $\mathscr{G}$ and that $\mathscr{I}$ is not acyclic. Consider a minimal cycle in $\mathscr{I}$; this is a sequence $s_1, s_2, \ldots, s_k, s_{k+1} = s_1$ so that $(s_i, s_{i+1}) \in \mathscr{I}$ for $1 \leqq i \leqq k$ but $(s_i, s_j) \notin \mathscr{I}$ unless $j = i+1$. This cycle contains no inclusion pair; otherwise, we could shorten the cycle by the previous observation. Therefore, the cycle consists only of exclusion pairs. Thus either $\ell_{\bar{s}_1} \in I$ and $r_{\bar{s}_2} \notin I$, or $r_{\bar{s}_1} \in I$ and $\ell_{\bar{s}_2} \notin I$; assume without loss of generality that the latter is the case. Then we have $r_{\bar{s}_i} \in I$ for all $1 \leqq i \leqq k$ and therefore $r_{s_i} < r_{s_{i+1}}$ for all $1 \leqq i \leqq k$. But then $r_{s_1} < r_{s_1}$, a contradiction. □

Lemma 4.1 implies that if I is exposable then $\mathscr{I}$ is acyclic. In this case, the transitive closure of $\mathscr{I}$ is a partial order. The lemma below considers linear extensions of such partial orders.

*A subset of nodes is *independent* if no two of its nodes are adjacent.

LEMMA 4.2 *A subset I of $L \cup R$ is exposed in an inclusion consistent ranking ρ of S if and only if ρ is a linear extension of $\mathscr{I}$.*

Proof ($\Leftarrow$) Suppose $x \in I$ is not exposed in a ranking ρ of S that is a linear extension of $\mathscr{I}$. Assume without loss of generality that $x = \ell_s$. Then there is an interval t with $\rho(t) < \rho(s)$ and $x \in t$. Hence, either $t \supseteq s$, which is not possible since then (s, t) is an inclusion pair, or r_t and $x = \ell_s$ exclude each other, which is not possible either because (s, t) is an exclusion pair.

($\Rightarrow$) Let I be exposed in an inclusion consistent ranking, ρ, which is not a linear extension of $\mathscr{I}$. That is, there are s and t so that $\rho(s) < \rho(t)$ and $(t, s) \in \mathscr{I}$. Hence, either $r_t \in I$ and $r_t \in s$, or $\ell_t \in I$ and $\ell_t \in s$. In either case an endpoint in I (r_t or ℓ_t) is not exposed in ρ—a contradiction. □

Remark Lemma 4.2 implies that I is exposable if and only if $\mathscr{I}$ is acyclic. This together with Lemma 4.1 proves the earlier claim that I is exposable if and only if it is independent in $\mathscr{G}$. Notice that Lemma 4.2 talks only about inclusion consistent rankings. However, because I is a subset of $L \cup R$, and not only of $L_S \cup R_S$, I is exposable if and only if it can be exposed by an inclusion consistent ranking.

4.5 Computing an Optimal Ranking

By Lemma 4.2, $\mathscr{I}$ provides a description of *all* inclusion consistent rankings in which I is exposed. We summarize:

1) There is a maximum exposable set of endpoints, I, which is a subset of $L \cup R$, where L and R are the sets of left and right endpoints that are not shy.
2) A set I is a maximum exposable subset of $L \cup R$ if and only if I is a maximum independent node set in $\mathscr{G}$.
3) A linear extension of $\mathscr{I}$ is an inclusion consistent ranking of S.

By (2), computing a maximum exposable set amounts to computing a maximum independent set in the exclusion graph $\mathscr{G}$ of S which is bipartite. As is well known, a maximum independent set is the complement of a minimum covering node set* and, by a theorem of König [7] (see e.g. [2, Thm. 5.3]), the size of a minimum covering in a bipartite graph is equal to the size of a maximum matching.† Furthermore, given a maximum matching of $\mathscr{G}$, time proportional to the number of nodes and arcs of $\mathscr{G}$ suffices to construct such a minimum covering or its complement, a maximum independent set. Now, a maximum matching in a bipartite graph with $N \leqq 2n$ nodes can be found in time $O(n^{5/2})$ time (see [5] or [4]); this step is the bottleneck of our algorithm.

Note that the graph $\mathscr{G}$ might have up to $\binom{n}{2}$ edges. However, we never have to

*A subset of the nodes in a graph is a *covering* if every arc is incident to at least one node in the set.

†A subset M of the arcs in a graph is a *matching* if no two arcs in M are incident to a common node.

store $\mathscr{G}$ explicitly—rather the set of intervals constitutes an O(n) storage implicit representation of $\mathscr{G}$ from which the adjacency of two given nodes (two endpoints) can be deduced in constant time. An inspection of the maximum matching algorithm in [5] and of the transformation from a maximum matching to a maximum independent set (see e.g. [2, chapter 5]) shows that O(n) working storage suffices.

Given I, we can compute a linear extension of $\mathscr{I}$ in time O(n^2), again without explicitly storing $\mathscr{I}$. One way to do this is to store with each interval t the number of intervals s with $(s,t)\in\mathscr{I}$. To compute these counters we simply consider each pair of intervals, (s,t), decide in constant time whether it is in $\mathscr{I}$, and if it is we increment t's counter by one. After this initialization we repeatedly remove an interval with counter equal to 0. When we remove s we also test pairs (s,t), for all t still in the structure, and decrement t's counter if $(s,t)\in\mathscr{I}$. The sequence of removed intervals is the linear extension of $\mathscr{I}$. This implies the main result of this section.

THEOREM 4.3 *A maximum exposable set, I, of endpoints of a set S of n intervals (with a ranking ρ in which I is exposed) can be computed in time* O($n^{5/2}$) *time and storage* O(n).

5. DISCUSSION AND OPEN PROBLEMS

In this paper we introduced the notions of ranking intervals and of visibility in a ranked set of intervals. We showed that any set of intervals has a ranking so that each interval sees at most two endpoints and therefore at most three other intervals. This result has application to an interval intersection problem discussed in Section 3. We also showed how to compute a ranking that minimizes the average number of endpoints seen per interval. This problem has application to the one-sided channel routing problem.

Still, there are a number of open problems that remain. For some sets it is possible to rank the intervals so that each interval sees at most one endpoint. Such a ranking would simplify the search algorithm for the interval intersection problem because it replaces a ternary decision per interval (it sees up to three other intervals) by a binary decision (only two intervals are visible). To the best of the author's knowledge it is still open whether there is a polynomial time algorithm that computes a ranking with each interval seeing at most one endpoint, if such a ranking exists.

Another open problem is concerned with sets of intervals that change over time. Is it possible to maintain a ranking in which each interval sees only a constant number of endpoints and where the insertion or deletion of an interval requires only few changes in the ranking? A positive solution to this problem could lead to a simple and efficient dynamic data structure for the above mentioned interval intersection problem.

There are various open problems related to the one sided channel routing problem. One of them is to solve the one-sided channel routing problem with multi-point nets. One possible approach to this problem is to break up each interval which corresponds to the span of a net into subintervals, and find a ranking with a maximum exposed set. However, this does not necessarily correspond to a solution requiring a minimum number of vias. Another open problem is when the number of available tracks is a given constant t. In this case we would like to find an *assignment*, α, which is a mapping from an interval set to the tracks $\{1, 2, \ldots, t\}$ such that intervals mapped to the same track are disjoint, and the number of exposed endpoints is maximized.

The notions of ranking, assignment and visibility can be extended to objects in two and higher dimensions. A *ranking*, ρ, is a bijective map from a set of objects, S, to $\{1, 2, \ldots, n\}$, where $n = |S|$, and s *sees* t if there is a point $p \in s \cap t$ that does not lie in any $u \in S$ with $\rho(s) < \rho(u) < \rho(t)$. We can thus ask the question whether or not the results of this paper can be generalized to two and higher dimensions. In general, S cannot be ranked so that each object sees only few other objects—take for example S as a set of n vertical and n horizontal lines in the plane. On the other hand, n half-planes can be ranked so that each half-plane sees only $O(\log n)$ other half-planes—is $O(1)$ possible? Positive results along these lines would imply new algorithms for intersection problems in two and higher dimensions. We refer to [9] for some results in the planar case.

References

[1] I. Ben-Arroyo Hartman and J. A. Feldman, Optimal interval visibility, Techn. Rept. TR88.277, IBM Scientific Center Israel, Haifa, Israel, 1989.

[2] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, Macmillan, Hong Kong, 1976.

[3] H. Edelsbrunner, A new approach to rectangle intersections—Part I, *Intern. J. Comput. Math.* **13** (1983), 209–219.

[4] Sh. Even and R. E. Tarjan, Network flow and testing graph connectivity, *SIAM J. Comput.* **4** (1975), 507–518.

[5] J. E. Hopcroft and R. M. Karp, An $n^{5/2}$ algorithm for maximal matchings in bipartite graphs, *SIAM J. Comput.* **2** (1973), 225–231.

[6] R. G. Karlsson and M. H. Overmars, Scanline algorithms on a grid, *BIT* **28** (1988), 227–241.

[7] D. König, Graphs and matrices (in Hungarian), *Mat. Fiz. Lapok* **38** (1931), 116–119.

[8] E. Kuh, T. Kashiwabara and T. Fujisava, An optimum single row routing, *IEEE Trans. Circuits Systems* (1979), 361–368.

[9] M. van Kreveld, M. H. Overmars and M. Sharir, in preparation.

[10] E. M. McCreight, Efficient algorithms for enumerating intersecting intervals and rectangles, Report CSL-80-9, Xerox Palo Alto Research Center, 1980.

[11] E. M. McCreight, Priority search trees, *SIAM J. Computing* **14** (1985), 257–276.

[12] F. P. Preparata and M. I. Shamos, *Computational Geometry—An Introduction*, Springer-Verlag, New York, 1985.

[13] R. Raghavan and S. Sahni, The complexity of single row routing, Techn. Rept., Dept. Comput. Sci., Univ. Minnesota, 1981.

[14] D. E. Willard, Log-logarithmic worst-case range queries are possible in space $\Theta(n)$, *Inform. Process. Lett.* **17** (1983), 81–84.

[15] D. E. Willard, New trie data structures which support very fast search operations, *J. Comput. Syst. Sci.* **28** (1984), 379–394.