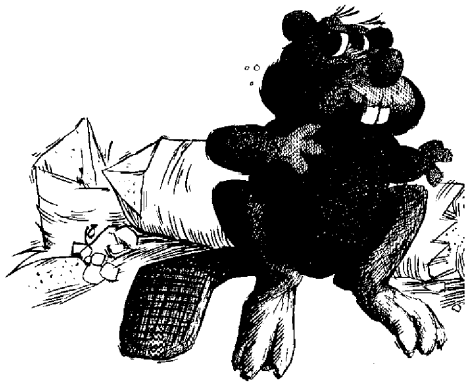

Space-Economical Plane-Sweep Algorithms

Thomas Ottmann
Derick Wood

Data Structuring Group
CS-84-32

October, 1984

SPACE-ECONOMICAL PLANE-SWEEP ALGORITHMS[1]

Thomas Ottmann[2]

Derick Wood[3]

ABSTRACT

Solving geometrical problems using plane sweep is now a well understood paradigm. Often plane-sweep solutions go hand in hand with the use of semi-dynamic data structures. One serious drawback of such structures is that they typically have size linear in the number of edges or line segments in the problem at hand. In practical situations this linear storage requirement is often prohibitive. For this reason we present two general techniques which ameliorate the situation when the geometrical problem satisfies a sub-linear distribution condition. An example of such a condition is: For VLSI designs consisting of n line segments experience shows that at most $\sqrt{n}$ cut the sweep line at any position. In this case our results imply that $O(\sqrt{n})$ and $O(\sqrt{n} \log n)$ space plane-sweep solutions are achievable for many problems.

1. INTRODUCTION

Hon [H, p. 3] states that it is now common for VLSI designs to consist of more than one million rectangles. This observation delineates yet another gap between theory and practice. On the one hand theoretical investigations into the complexity of various computational-geometric problems, abstracted from the area of VLSI design as well as other areas, have assumed that the input data is

1 The work of the first author was carried out under DFG Grant No. Ot 64/4-2, while that of the second was carried out under Natural Sciences and Engineering Research Council of Canada Grant No. A-5692 during a visit to the University of Karlsruhe.

2 Institut für Angewandte Informatik und Formale Beschreibungsverfahren, Universität Karlsruhe, D-7500 Karlsruhe, West Germany.

3 Data Structuring Group, Department of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada.

held wholly in core. This has meant that space linearly proportional to the size of the input data is the best that can be achieved, that is it is optimal. Clearly such an assumption is indefensible in light of Hon's statement. On the other hand practitioners assume the input data is stored externally and attempt to design algorithms which do not assume the input data is wholly in core, for example see [L].

Fortunately this gap can be closed quite simply by assuming off-line variants of computational models are used rather than their on-line variants. This means that the storage for the input data itself is not taken into account, only the space needed for data held or produced during processing is accounted for. This assumption means that computational geometry problems should be re-examined to see if they can be solved with sub-linear space requirements while retaining good time bounds. This paper provides a starting point for such a general investigation by examining plane-sweep algorithms, which abound in various aspects of VLSI design tasks. See [NP] for a general introduction to plane-sweep. During a plane-sweep the space requirements are of two distinct kinds. First, space is always required for the objects currently intersected by the sweep line, we call this *sweep-line space* and the associated structure the *sweep-line structure*. Second, additional space may be required for the sweep history, that is about the objects already swept over, this we call *sweep-history space* and the associated structure the *sweep-history structure*. In region reporting, for example, storage of both kinds is, apparently, necessary, see [NP], but in a recent paper [SvW] have shown that only sweep-line space is needed if a two-sweep algorithm is used. The importance of this result stems from the 'square root rule' for VLSI designs, that is any vertical or horizontal cross section reveals proportional to $\sqrt{n}$ objects if the whole design consists of n objects. Hence at each position of the sweep-line there are only $\sqrt{n}$ objects intersecting it, for example if $n = 1{,}000{,}000$ then we only need space proportional to 1,000 rather than to the initial pessimistic bound of 1,000,000. [SvW] demonstrate that region reporting can, indeed, be carried out using only $O(\sqrt{n})$ space, when it is assumed that the square root rule holds. Once they have solved the, by no means simple, problem of removing the need for sweep-history space, the sweep-line space bound is immediate since a fully-dynamic sweep-line data structure is used. However there are plane-sweep algorithms in which the sweep-line data structure is semi-dynamic rather than fully dynamic. The distinction between the two kinds of structures is that under updating semi-dynamic structures have a pre-set fixed number of nodes, for example vectors and hash tables, while fully-dynamic structures have a variable number of nodes, for example a binary search tree. An example of the use, in a plane-sweep algorithm, of a semi-dynamic structure, the segment tree, is seen in [BW], while the use of a fully-dynamic structure is seen in [BO]. The semi-dynamic sweep-line structures used in various plane-sweep algorithms have space requirements $O(n)$, or even $O(n \log n)$, see [BO], [BW], [Wo], [OWo], and [GO] for example.

This distinction between sweep-history and sweep-line structures and the associated space requirements leads to two separate, but inter-connected avenues of investigation. Since [SvW] have already examined sweep-history structures to some extent we turn to the study of sweep-line structures and, hence semi-dynamic sweep-line structures.

We present two similar techniques for saving space in sweep-line structures. The first depends on the *cross-section number* C, the maximum number of objects intersecting the sweep-line during the plane sweep. For VLSI designs $C = O(\sqrt{n})$ as we have already mentioned. We give a simple and generally applicable technique which ensures that only two sweep-line structures need be kept requiring $O(f(C))$ space when the usual semi-dynamic sweep-line structure require $O(f(n))$ space, for n the number of objects. The second depends on the partial order $above^+$ among the objects. It only requires one sweep-line structure requiring $O(f(M))$ space, where M is the length of the longest maximal chain in the partial order and $O(f(n))$ space is required by the usual semi-dynamic structure. To illustrate these ideas we use as a running example, the *Line Segment Intersection Problem* (*LSIP*), but we emphasize that this is not restrictive, it has been chosen since it is a basic problem underlying other more complex problems, and, at the same time, is simple enough not to obscure the ideas we wish to present. We first treat these ideas for the isothetic variant of the *LSIP*, that is only horizontal and vertical line segments are allowed. In Section 2 we present a solution to the *Isothetic LSIP* (*ILSIP*) using a semi-dynamic sweep-line structure, then in Section 3 we present the first space-saving technique and in Section 4 the second. In Section 5 we discuss how these ideas can be carried over to the general, nonisothetic, *LSIP* and, finally, in Section 6 we discuss some open problems and the relationship of our work to that of [SvW].

Notation: Throughout the paper we use n to denote the size of input data and k to denote the size of the output data.

Assumptions: Throughout the paper we assume:

(i) The endpoints of the *LSIP* are presented off-line in sorted order. Hence their sorting time and their size is ignored in analyzing algorithms.

(ii) We only treat worst-case time and space asymptotic complexity measures.

(iii) We assume the comparison-based model of computation [AHU] throughout.

2. THE ISOTHETIC LINE-SEGMENT-INTERSECTION PROBLEM

The *Isothetic Line-Segment-Intersection Problem* (*ISLIP*) of size $n \geq 1$ can be stated as follows:

Given n horizontal and vertical line segments in the plane with no two line segments being co-linear, determine all intersecting pairs consisting of a vertical and a horizontal line segment.

This problem was first posed and solved in [BO]; their solution is time- and space-optimal for a comparison-based model of computation. First we sketch their solution in order to recall the plane-sweep paradigm and second, we modify it to introduce semi-dynamic sweep-line data structures.

A vertical line which sweeps through the plane from left to right or vice versa, is said to be a *(vertical) sweep-line.* Given an instance of the *ILSIP*, a sweep line cuts, at each position, a number of horizontal line segments and overlaps some vertical line segments — the *set of active objects.* This set only changes during the sweep when an endpoint is met. The n line segments provide at most $2n$ endpoints and, therefore, at most $2n$ positions at which the set of active objects changes, see Figure 2.1.

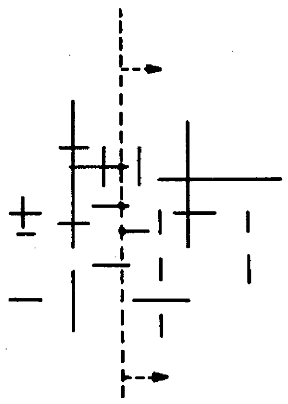

Figure 2.1
The plane sweep approach

A horizontal-vertical intersection is obtained whenever a vertical line segment is met which intersects an active horizontal line segment. Moreover, since active horizontal line segments appear as points on the sweep line intersections can be detected by means of a range query with a newly active vertical line segment on the set of active points. In other words a two-dimensional problem has been replaced by at most $2n$ one-dimensional problems. A high level version of this solution is:

{The sweep points are the x-values of the left and right endpoints of horizontal line segments and the x-values of the vertical line segments sorted by their x-values}

```
while  Sweep points not exhausted do
begin  Obtain next endpoint $(x,E)$, say, where $x$ is the $x$-value of the end-
       point and $E$ the segment to which it belongs.
    if $x$ is a left endpoint of $E$ then
       Add $E$ to set of active horizontal line segments else
    if $x$ is a right endpoint of $E$ then
       Remove $E$ from the set of active horizontal line segments else
    {$E$ is a vertical line segment}
       Perform a range query with $E$ on the set of active horizontal line
```

```
        segments reporting all intersecting pairs discovered.
end
```

To convert this high level solution into an efficient algorithm requires a sweep-line data structure which can represent points on a line, allows efficient updating, and provides efficient range querying. The solution in this case is straightforward — a balanced search tree, for example the AVL tree [AHU]. This is because the horizontal line segments are represented, for our purposes, by their y-values. A range query is easily accommodated in such a search tree, see Figure 2.2, and for an AVL tree with n nodes it requires $O(\log n + k)$ time, where k is the number of line sequents falling within the query interval. Overall the algorithm requires $O(n \log n + k)$ time and $O(n)$ space, where k is the number of intersecting pairs. This is because there are at most $2n$ actions which are either updates of the sweep-line structure or a query with respect to it.

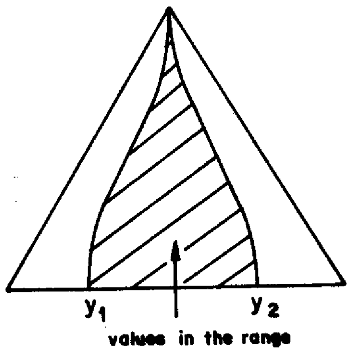

Figure 2.2
A range query

An AVL tree is a *fully dynamic data structure* or *FD data structure*, that is it grows and shrinks as a result of insertions and deletions; its size is always of the same order as the number of items it represents. We now replace this structure with a *semi-dynamic data structure* or *SD data structure* which does not grow and shrink in this fashion. We will discuss later why such a structure can be a 'a good thing.' We refer to the above algorithm as the *FD algorithm* when an *FD* structure is used and as the *SD algorithm* when an *SD* structure is used.

The y-values associated with the given set of horizontal line segments are known in advance, they do not take us by surprise during the plane sweep. Therefore we can construct a search tree for the whole set before the sweep commences. Then rather than inserting and deleting nodes we merely mark items as present or absent. (Recall that we have assumed the y-values of the horizontal line segments to be pairwise distinct.) See Figure 2.3. It is easy to arrange that the search tree is of minimal height, hence such updating is again logarithmic. A range query, at first sight, also appears to be logarithmic modulo the reporting time. Unfortunately this is not so as a little thought shows.

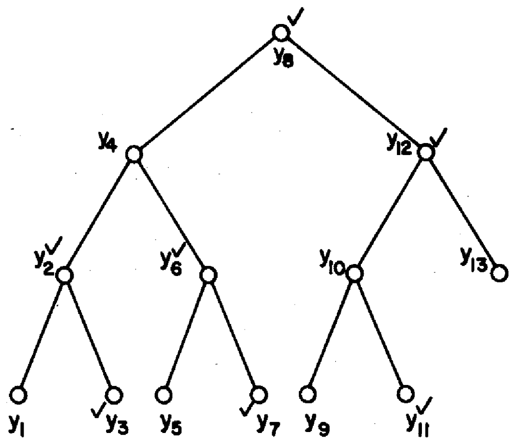

Figure 2.3
A semi-dynamic search tree

The reason is that a range query may include within it many nodes whose associated y-values are absent. So a range query may take time linear in the size of the tree. To avoid this difficulty we associate with each node the set of all present values in its subtree. We call these *present sets*. In Figure 2.4 the present sets corresponding to Figure 2.3 are shown. First observe that insertion and deletion in such a modified tree, while not as simple as before is not difficult. On insertion of a y-value y_i, y_i is added to all present sets on the search path as well as being marked as present. On deletion of a y-value y_i, y_i is removed from all present sets on the search path and also marked as absent. Implementing the present sets as doubly-linked lists ensures that the removal of a item takes constant time — once it has been found. To find the appearances of y_i in all present sets we simply link the appearances together on insertion, linking them to the node containing y_i. Then, on deletion of y_i the node containing y_i is first found, when its appearances are traced and removed one by one. Thus updating is still a logarithmic operation. A range query is now answered by using the present sets to avoid traversing subtrees. For example the query $[y',y'']$ where $y_2 < y' < y_3$ and $y_9 < y'' < y_{10}$ visits the blocked-out and hacheéd nodes shown in Figure 2.5. The blocked-out nodes fall within the range, while the hacheéd nodes do not. The right subtrees of the blocked-out nodes that are on the search path of y' are within the range also, as are the left subtrees for y''. Thus the answer to the range query is the union of the present sets at nodes y_3, y_6, and y_9 which is $\{y_3, y_6, y_9\}$ that is the y-values at the blocked-out nodes together with the present sets associated with the roots of the subtrees in the range. Since $O(\log n)$ nodes are visited and no subtrees are investigated the query time is $O(\log n + k)$ once more, where k is the number of reported values. Readers familiar with the segment tree of [B1] ([BW] is more easily accessible) will see much similarity. We call the semi-dynamic structure we have described the semi-dynamic *range tree*. Observe that it requires $O(n \log n)$ space in the worst case since each present value is added to $O(\log n)$ present sets and $O(n)$ values are present. Thus our modified algorithm requires $O(n \log n + k)$ time and $O(n \log n)$ space — a deterioration in performance over the original algorithm. Therefore it is natural to ask: Why use semi-dynamic structures? First we must point out that such a structure has been introduced for the *ILSIP* for pedagogic rather than algorithmic purposes. It is a bad choice here. However there are problems for which no fully dynamic structure with logarithmic performance is known to exist, for example the measure problem in 3-space [vLW], the rectangle containment problem [VW], the contour problem [LP, Wo], the connected components problem [EvLOW], the region reporting problem [NP, SvW], etc. In these cases we are, at present, forced to use semi-dynamic sweep-line structures. For other problems a semi-dynamic structure turns out to be a better choice, in terms of simplicity and space utilization, for example in hidden-line elimination [OWW, OW2].

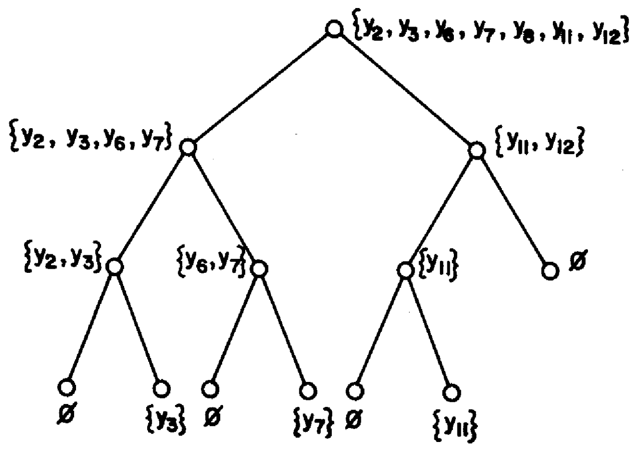

Figure 2.4
The semi-dynamic range tree

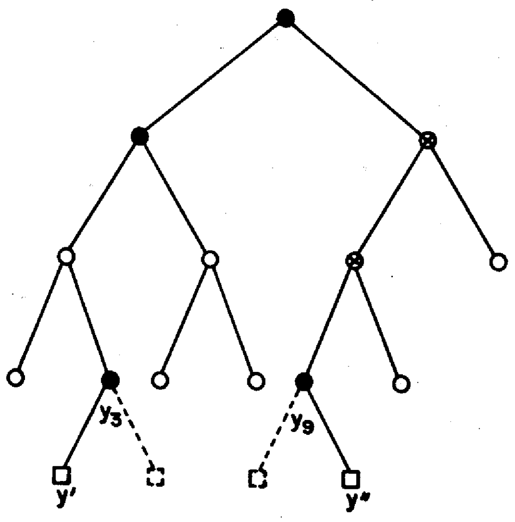

Figure 2.5

In the next two sections we discuss how the space requirements for the semi-dynamic range tree can be lowered, often substantially, when used as a sweep-line structure.

3. SPACE ECONOMY — THE FIRST APPROACH

In this section we assume that C, *the cross-section number*, is the maximum number of active horizontal line segments over all positions of the sweep line for the given instance of *ILSIP*.

First, note that C is easily calculated in $O(n)$ time and $O(1)$ space. Simply maintain, for each position of the sweep line, the number of active horizontal line segments, updating the current maximum if necessary.

Second we wish, if possible to modify the *SD* algorithm for *ILSIP* to use $O(C \log C)$ space rather than $O(n \log n)$ space. The *FD* algorithm uses only $O(C)$ space, but the *SD* algorithm, as explained in Section 2, does not satisfy the $O(C \log C)$ space requirement. We present a generally applicable technique to do this. It is worthwhile, at this stage, commenting on the space-economization technique introduced in [SvW]. They are also concerned with reducing the storage used by data structures during a plane sweep, however they attack a different problem. A number of plane-sweep algorithms require that information about objects already swept over be kept, that is the sweep-history structure, for example in computing connected components [EvLOW], closure [SSW], or region reporting [NP, SvW]. This information typically requires $O(n)$ space even though the active objects may require only $O(C)$ space. [SvW] avoid the space for this extra information by use of a two-sweep algorithm for region reporting. Because their basic data structures are fully dynamic this ensures that $O(C)$ space is required over all. Thus the approaches of [SvW] and ourselves are complementary rather than overlapping.

Third, we emphasize once more that for $n > 1{,}000{,}000$ $O(n)$ and $O(n \log n)$ space algorithms are prohibitively space consuming, since most computer systems give low priority to programs with such space requirements, if they have this amount of core available. However if $C = \sqrt{n}$ as is the case in VLSI circuits, then $O(C)$ and even $O(C \log C)$ space *are* acceptable.

Let us turn to consider our general technique for *ILSIP*. The idea is simple, we keep two semi-dynamic sweep-line tree structures each having C nodes. We call them *OLD* and *NEW*; they are both of minimal height. Now when the plane-sweep begins we examine the first C horizontal line segments and arrange for *OLD* to be an *SD* range tree for them. This means that the plane sweep must look ahead in order to accumulate the information. For this purpose we assume the horizontal and vertical line segment endpoints are kept in separate files. Modifying the algorithm of Section 2 to accommodate this change is straightforward. When the sweep line meets the $(C+1)$st horizontal line segment the look-ahead technique is used to make *NEW* a *SD* range tree for the $(C+1)$st to $2C$th horizontal line segments. The sweep line during this stage first performs insertions into *OLD*, then insertions into *NEW*, and deletions from both. When a vertical line segment is met its associated range query takes place on both *OLD* and *NEW*; clearly the reports from *OLD* and *NEW* are disjoint and the range query performs correctly since the query is decomposable [B2].

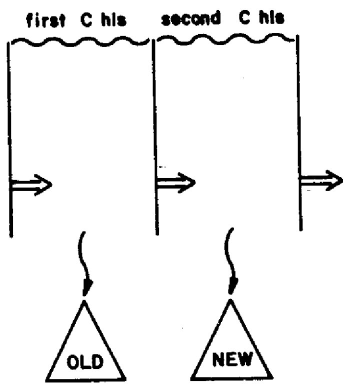

Figure 3.1

For now we ignore the time taken by this process, concentrating solely on the $O(C \log C)$ space that is required in the worst case. On reaching the third set of C horizontal line segments our approach is somewhat different. First we merge *OLD* and *NEW* back into *OLD*. This is necessary since we cannot guarantee that either *OLD* or *NEW* is empty at this stage. Schematically we have Figure 3.2. We perform the merge by, literally, merging the present sets at the roots of both *OLD* and *NEW* into one sorted list of at most C items. This is then used as the basis for re-constructing *OLD*. When re-building *OLD*, all the y-values

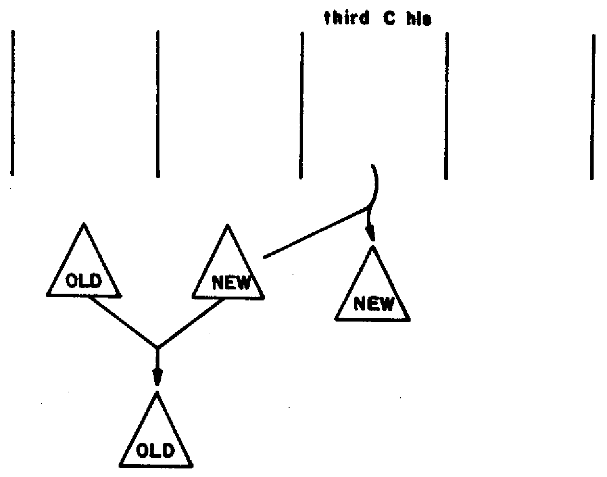

Figure 3.2

are present so the present sets are particularly easy to construct. This process requires $O(C \log C)$ time over all, as does the construction of each *SD* range tree for C items. Finally the third set of C horizontal line segments is associated with *NEW*, hence this stage is similar to that obtaining on meeting the $(C+1)$st horizontal line segment. This approach keeps two sweep-line structures throughout the plane sweep each requiring $O(C \log C)$ space, thus our space requirements have been met.

It remains to analyze the time taken by this new approach.

Since there are at most $\lceil n/C \rceil$ sets of C horizontal line segments the time taken to tailor the structures to current needs is:

$$O(\lceil n/C \rceil \cdot C \log C) = O(n \log C)$$

since at each stage at most two structures are tailored.

The time taken for the foreground tasks of insertion and deletion each take $O(\log C)$ time, while a range query requires $O(\log C + k)$ time. Therefore the algorithm requires, over all,

$$O(n \log C + k) \text{ time} \quad \text{and} \quad O(C \log C) \text{ space.}$$

If $C = \sqrt{n}$ this yields $O(n \log n + k)$ time and $O(\sqrt{n} \log n)$ space, while if $C = \log n$ we obtain $O(n \log\log n + k)$ time and $O(\log n \log\log n)$ space.

In the next section we consider a different approach.

4. SPACE ECONOMY — THE SECOND APPROACH

While the approach taken in the previous section is an eminently practical technique it leaves open the question of whether or not a single sweep-line structure with C nodes can be used. If only C slots are available at each sweep-line position, then during the plane sweep many horizontal line segments must be identified with the same slot. This in turn implies that the total order of horizontal line segments at each sweep-line position must be maintained and, further, maintained efficiently. We first demonstrate that such maintenance can require $O(C \log C)$ time for an *SD* range tree. In Figure 4.1 we have a set of n horizontal line segments for which $C = 2$ and which have the property that a new line segment is always added below the active ones to maintain the total order and a line segment is always deleted when it is topmost in the current total order. This figure can easily be generalized, for any value of C, which implies that for all i, segments i, $i+1, \ldots, i+C-1$ must be in slots $1,2, \ldots, C$, respectively. But this means that the addition of segment $i+C$ and the removal of segment i forces a shift of the items so that segments $i+1, \ldots, i+C$ are in slots $1, \ldots, C$, respectively. This takes at least $\Omega(C)$ time for many sweep-line structures and for an *SD* range tree requires $O(C \log C)$ time since the present sets need to be recomputed. Clearly such a time bound for one update is prohibitively expensive, thus we wish to avoid re-assigning values to slots when once they have been assigned. If each incoming segment never changes its slot assignment during its lifetime, then this implies that at each sweep-line position the segments must occur in the sweep-line structure in an order which is their total ordering with respect to the sweep-line. Clearly n slots are always sufficient to guarantee this, however we wish to economize on space. This gives rise to the *Minimal Slot Assignment Problem* (*MSAP*) which can be stated as:

Given a set of n nonoverlapping horizontal line segments in the plane find the smallest positive integer N which satisfies the condition that the segments can be assigned slots in the range 1..N such that the total order of line segments at each sweep-line position equals the total order given by their slot assignments.

The above argument shows that the cross-section number C is an inappropriate measure when requiring a single semi-dynamic sweep-line structure, that is when solving the *MSAP*. To see what the appropriate measure is we define a relation *above* for horizontal line segments as follows:

For two horizontal line segments L_1 and L_2 L_1 *above* L_2 holds iff L_1 and L_2 have an x-value x, in common and the y-value of L_1 at x is greater than the y-value of L_2 at x.

In Figure 4.1 i *above* $i+1$ holds for all i, $1 \leq i < n$. The transitive closure of this relation, denoted by *above*$^+$, is a partial order on the set of horizontal line segments. In Figure 4.1 i *above*$^+$ j, for all i, $1 \leq j < n$. Now in Figure 4.1 the set of n line segments forms a *maximal chain* in *above*$^+$, since

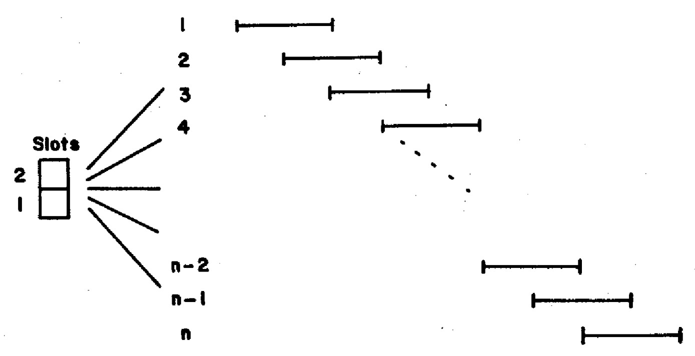

Figure 4.1

$1\,above^+\,2\,above^+\,3 \cdots above^+\,n$. This is the reason that 2 slots are insufficient as a solution to this instance of the *MSAP*. The relation $above^+$ was introduced in [GY] and has been studied in [OW1] and [ChOSSW]. In the partial order of Figure 4.1 there is one and only one maximal chain however, in general, there are many. The length of the longest maximal chain M, provides a lower bound on the minimal number N of slots required, which is clearly n in Figure 4.1. However it is possible that the longest maximal chain is much smaller than n. In Figure 4.2 we have $n = 10$, $C = 3$, and $M = 5$ via $a\ above^+\ d\ above^+\ e\ above^+\ i\ above^+\ j$. A possible slot assignment is:

1	a, f, b, c
2	d, h
3	e, g
4	i
5	j

and another is:

1	a
2	d, c
3	e, b, h
4	i, f, g
5	j

Clearly the longest maximal chains must be assigned slots $1, \ldots, M$ when $M = N$, but apart from these assignments there is considerable freedom with the assignment of the other segments. There isn't complete freedom, since the assignment must be consistent with the partial order, so i and h, for example cannot be assigned the same slot, since this implies j and g must share a slot,

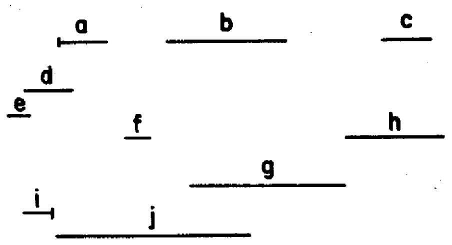

Figure 4.2

which is inconsistent with the partial order, *above*$^+$. Indeed the *MSAP* can be rephrased to require only that the slot assignment is consistent with the partial order *above*$^+$.

In Figure 4.2 the number of slots is exactly M. Therefore it is natural to ask whether or not M slots are always achievable. To demonstrate this we present an inductive slot assignment algorithm which we prove always requires exactly M slots. Assume light is shining vertically down on the line segments from above. Then the line segments a, f, b, and c of Figure 4.2 are completely illuminated, while the other segments are either partially illuminated or in complete shadow, see Figure 4.3. The fully illuminated segments are assigned slot 1 and are then removed. Observe that this reduces the length of the maximal chain of the partial order *above*$^+$ for the remaining set by 1. The process is repeated for slot 2, and so on until all segments have been assigned slots. Again this requires, by our previous arguments, at least M slots. That it requires at most M steps to remove all elements is immediate, otherwise a maximal chain of length greater than M would have been found.

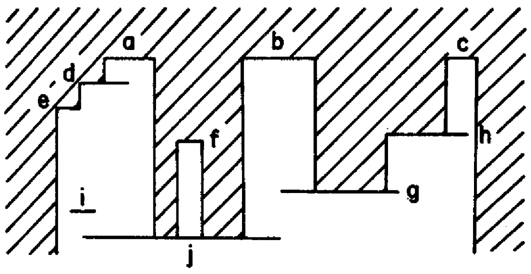

Figure 4.3

Before providing an efficient algorithm for solving *MSAP* we briefly discuss how we use such an assignment during a plane sweep. Assume each horizontal line segment carries its slot number with it. During the preprocessing step of the plane-sweep algorithm a semi-dynamic range tree with M nodes is constructed based on the integers 1..M. Insertion and deletion takes place by using the slot number as search key, while a range query refers to the y-values of the line segments actually represented at the time. Since the slot numbers of the line segments are consistent with their partial order the modified plane-sweep algorithm operates correctly.

Thus we obtain:

Theorem 4.1 *Given an instance of ILSIP of size n having M as the length of its longest maximal chain, then the SD algorithm sketched above solves it in* $O(n \log M + k)$ *time and* $O(M)$ *space.*

But how do we compute a minimal slot assignment efficiently in time and space? We present a time-optimal algorithm which requires $O(n)$ space. The problem of finding a space-efficient solution is left open, but the reader should note then the $O(n)$ space requirement is not that bad. It occurs in a very specific and simple algorithm using a simple data structure. However once we have found a minimal slot assignment, we may use it for *many* algorithms and *many* complex data structures.

Computing a Minimal Slot Assignment

Perform a top-to-bottom sweep of the horizontal line segments painting the sweep line with the assigned color of each line segment as it is met. This ensures that we only need keep the current painted intervals of the sweep line, since earlier covered paint is never seen again. The sweep line is painted in color 0 initially, and the corresponding interval of a new line segment is painted in color (1 + the maximum color it covers). For example the line segments of Figure 4.3 give the colored snapshots of the sweep line displayed in Figure 4.4.

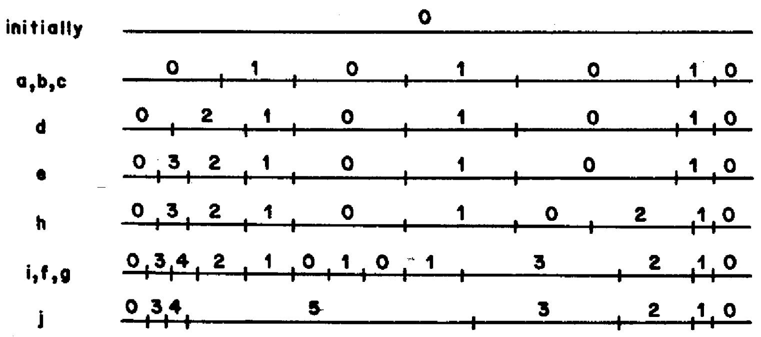

Figure 4.4

An AVL tree can be used to efficiently maintain the colored intervals. A colored interval is associated with a single node; its endpoints serve as search keys branching left depends on the left endpoint and branching right on the right endpoint. An incoming interval is treated as a range query with deletion. The range query determines all intervals it completely covers together with the, at most, two intervals it partially covers. All completely covered intervals are deleted, the maximum of their colors being noted. The, at most, two intervals partially covered are shrunk, that is either a left endpoint moves further right or a right endpoint further left. The maximum of all effected intervals is now known and this value plus one is the color of the new interval which is now inserted.

Since there are n horizontal line segments the time taken over all is $O(n \log n)$ since an interval is only completely covered at most once in its lifetime and therefore is only deleted at most once. So although a particular incoming interval may cause $O(n)$ deletions, these never occur again, thus the amortized cost is $O(n \log n)$. Unfortunately as we are carrying out a top-to-bottom sweep rather than a left-to-right sweep the space requirement is $O(n)$.

Summarizing we have:

Theorem 4.2 *Given n nonoverlapping horizontal line segments in the plane the MSAP can be solved in $O(n \log n)$ time and $O(n)$ space. Moreover this is a time-optimal solution.*

Proof: That $O(n \log n)$ time and $O(n)$ space is sufficient is demonstrated by the above algorithm. Under the usual assumptions for a comparison-based model of computation $O(n \log n)$ time is necessary, since we can use the minimal slot assignment algorithm to sort n distinct integers in the following way. Given n integers $y_1, \ldots, y_n$ represent them by unit horizontal intervals with endpoints $(0,y_i)$ and $(1,y_i)$, $1 \leq i \leq n$. These have $M = n$ and hence are assigned slot numbers $s_1, \ldots, s_n$ such that the inverse permutation $(s_1, \ldots, s_n)$ provides the sorted order of $(y_1, \ldots, y_n)$. Since sorting requires $O(n \log n)$ time in the given model so does the minimal slot assignment problem.

If we also assume the usual on-line model of computation the algorithm is also space optimal, but given our off-line model $O(n)$ space is not necessarily optimal. □

5. THE LINE SEGMENT INTERSECTION PROBLEM

The *Line-Segment-Intersection Problem* (*LSIP*) of size $n \geq 1$ can be stated as follows:

> *Given n arbitrarily-oriented line segments in the plane with no two segments being co-linear and no three having a common intersection point, determine all pairs of intersecting line segments.*

This problem was also first posed and solved in [BO] using plane-sweep based on the ideas in [ShH]. The space requirements of the solution given in [BO] were reduced to linear-space requirements in [Br] leading to an $O(n \log n + k \log n)$ time and $O(n)$ space solution. This was improved recently by [Ch] using divide and conquer to give an $O(n \log^2 n + k)$ time and $O(n)$ space solution. However we only discuss the plane-sweep solution. We begin by sketching, very briefly, the solution developed by [BO, Br]. Observe that two arbitrarily-oriented line segments may change their relative ordering, in the total order at a sweep-line position, at subsequent sweep-line positions. This occurs at their intersection point if they intersect. Thus the sweep-line structure must allow such changes to be carried out efficiently. Furthermore intersections are no longer detected by range queries unless an incoming segment happens to be vertical. They are detected by observing that two intersecting segments become adjacent in the sweep-line structure to the left of their intersection point. New adjacencies occur as a result of insertions, deletions, and interchanging the relative order of two intersecting line segments. This implies we need to be able to insert and delete segments, interchange adjacent segments, and find the successor or predecessor of a given segment. Whenever new adjacencies are created they are tested for intersection. If they intersect then, using the improvement due to [Br], only the leftmost intersection point, to the right of the sweep line, for each of the two line segments is maintained. Since each line segment can only have one such earliest intersection point this ensures an $O(n)$ space requirement. Note that future intersection points are also sweep points; they correspond to interchange operations. Modifying the *FD* algorithm for *ILSIP* to give an *FD* algorithm for *LSIP* is straightforward — an AVL tree can be used once more. To convert the *FD* algorithm into an *SD* algorithm use the semi-dynamic range tree modified so that two additional items of information appear at each node, namely the leftmost and rightmost present values in its subtree. This enables the successor and predecessor operations to be carried out in logarithmic time. Note that the interchange operation can be implemented with the successor, deletion, and insertion operations. We leave to the reader the task of deriving the two algorithms in full detail.

Let us first examine the effect of the cross-section number C on the *FD* algorithm. Immediately the sweep-line structure requires only $O(C)$ space and therefore $O(\log C)$ time for each access. The space requirement follows by having two sweep-point structures. The original sweep points are held in secondary storage as before, while the next C intersection points, at most, are held in core. The next sweep point is the minimum of the two minima — a constant time comparison is all that is needed. Hence the *FD* algorithm requires $O(n \log C + k \log C)$ time and $O(C)$ space over all — a clear and immediate improvement.

However modifying the *SD* algorithm to use less space is no longer as simple as it is for the *ILSIP*.

The First Approach

In the approach to space economy introduced in Section 3 we need to replace one *SD* structure of size $O(n)$ with two of size $O(C)$. The insertion, deletion, and interchange of line segments causes no difficulty whatsoever.

However the method of detecting intersecting pairs needs to be modified. Intersecting pairs are detected in the *SD* algorithm when they become adjacent in the sweep-line structure. When we have two structures it may happen that the two intersecting line segments are in different structures and so never become adjacent in the original sense.

The solution to this problem is as follows. Assume a line segment L, is deleted from or inserted into one of the structures and the new adjacencies are detected. Then the predecessor and successor of L in the remaining structure are found. The actual adjacencies are amongst these four and are easily determined.

It only remains to observe that as an interchange can be carried out as a sequence of deletions and insertions the first approach can indeed be implemented correctly. Furthermore it runs in $O(n \log C + k \log C)$ time, since at most C intersection points need be kept at any stage. The space requirement is once again $O(C \log C)$ if vertical line segments are allowed and $O(C)$ otherwise.

The Second Approach

The transitive closure of the relation *above*, denoted by $above^+$, is no longer a partial order for *LSIP* and this causes some difficulty. In Figure 5.1 $b\ above\ a$ and $a\ above\ b$ and hence in $above^+$ we have:

$$a\ above^+\ b \quad \text{and} \quad b\ above^+\ a$$

but $a \neq b$, that is $above^+$ is *not* anti-symmetric and is, therefore, not a partial order.

However all is not lost since the notion of (maximal) zig-zags introduced in [OW1] are partially ordered under $above^+$. We first define this notion. A *zig-zag* is a curve which consists of line segments and is monotonic with respect to the x-axis. Given a set of line segments in the plane they form many zig-zags. We say a zig-zag in such a set is *maximal* if it cannot be extended into a longer zig-zag at either end and we say two zig-zags *cross* if their order of appearance, on two vertical lines which cut them, is different. In Figure 5.2 six maximal noncrossing zig-zags are formed, namely a, b, c, d, e, and f, from nine line segments, which partition the figure. The maximal noncrossing zig-zags are intersection-free with the possible exception of endpoints and intersection points. A partition of a set of line segments into maximal noncrossing zig-zags is unique and, moreover the zig-zags are partially ordered by $above^+$. If there are n line segments then there are $O(n)$ maximal zig-zags. In [OW1] it is demonstrated how the maximal zig-zags can be computed, for a set of n (arbitrarily-oriented) line segments, in $O(n \log n + k \log n)$ time and $O(n)$ space, where k is the number of intersecting pairs of line segments. We now seem to have come full circle; to solve the *LSIP* with little space using a semi-dynamic range tree we need to solve the *LSIP*. At this point it is worthwhile recalling that we are using *LSIP* solely to illustrate the space-economization technique, not to solve the *LSIP* problem itself. In general we wish to solve some problem X using a single semi-dynamic sweep-line structure and this involves solving *LSIP* first. For this purpose we can use the first technique with a fully-dynamic sweep-line structure which uses no more space than the solution to problem X since $C \leq M$. In practice such a pre-processing step is relatively inexpensive, namely $O(n \log C + k \log C)$ time and $O(C \log C)$ space. These comments should be born in mind during the remainder of this section. In [OW2, OWo, WOW] the general applicability of zig-zags in translating plane-sweep solutions for isothetic problems into plane-sweep solutions for the corresponding nonisothetic problems is demonstrated.

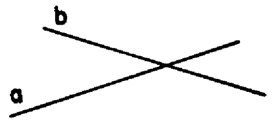

Figure 5.1

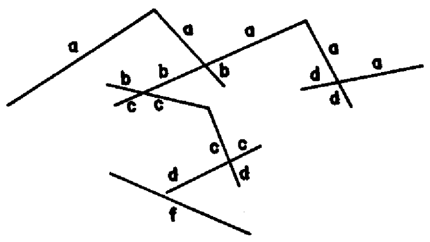

Figure 5.2

Once the zig-zags and their partial ordering are known we can treat them as we did horizontal line segments in the previous two sections. The one major difference is that a line segment may be associated with many different zig-zags and, hence, with many different slots. But at each position of the sweep-line a line segment is only associated with one slot. Its slot changes occur because it moves from one zig-zag to another as a result of an intersection, which changes its position in the partial order. Thus the semi-dynamic sweep-line structure requires $O(M \log M)$ space and the algorithm runs in time $O(n \log M + k \log M)$. As explained above a two-sweep algorithm for some problem X consisting of n line segments runs in time bounded from below by $\Omega(n \log M + k \log M + n \log C + k \log C) = \Omega(n \log M + k \log M)$ and in space bounded from below by $\Omega(C + M \log M) = \Omega(M \log M)$.

6. CONCLUDING REMARKS

Surprisingly there has been little theoretical investigation of space-economical plane-sweep algorithms even though such algorithms have significant practical applications. We have demonstrated two simple space-saving techniques, the first not only being the simpler of the two but also being the most practical. One pleasant side effect of using space-saving techniques is that they also yield time savings as well. Thus we obtain faster algorithms using less space — surely a desirable goal.

At the same time a number of new problems are raised. For example can the minimal slot assignment for line segments (or zig-zags) be computed in sublinear space? The algorithm given in the present paper depends on a top-to-bottom scan rather than a left-to-right one, therefore our space-saving techniques are inapplicable. At first glance no algorithm using a left-to-right scan exists, but . . .

References

[AHU] Aho, A.V., Hopcroft, J.E., and Ullman, J.D., *The Design and Analysis of Computer Algorithms.* Addison-Wesley Publishing Co., Reading, Mass. 1974.

[BO] Bentley, J.L., and Ottmann, Th., Algorithms for Reporting and Counting Geometric Intersections. *IEEE Transactions on Computers C-28* (1979), 643-647.

[Br] Brown, K.O., Comments on 'Algorithms for Reporting and Counting Geometric Intersections.' *IEEE Transactions on Computers C-30* (1981), 147-148.

[BW] Bentley, J.L., and Wood, D., An Optimal Worst Case Algorithm for Reporting Intersections on Rectangles. *IEEE Transactions on Computers C-29* (1980), 563-580.

[Ch] Chazelle, B., Reporting and Counting Arbitrary planar Intersections. Brown University Computer Science Technical Report CS-83-16 (1983).

[ChOSSW] Chazelle, B., Ottmann, Th., Saoisalon-Soininen, E., and Wood, D., The Complexity and Decidability of SEPARATION^{TM}, *Proceedings ICALP '84, Springer-Verlag Lecture Notes in Computer Science 172* (1984), 119-127.

[EvLOW] Edelsbrunner, H., van Leeuwen, J., Ottmann, Th., and Wood, D., Connected Components of Orthogonal Geometric Objects, *RAIRO - Informatique Théoretique 18* (1984), 171-183.

[GO] Güting, R.H., and Ottmann, Th., New Algorithms for Special Cases of the Hidden Line Elimination Problem, Universität Dortmund Technical Report 184 (1984).

[GY] Guibas, L.J., and Yao, F.F., On Translating a Set of Rectangles, *Proceedings of the 12th Annual ACM Symposium on Theory of Computing* (1980), 154-160.

[H] Hon, R.W., The Hierarchical Analysis of VLSI Designs. Doctoral Dissertation, Carnegie-Mellon University, Report CMU-CS-83-170 (1983).

[L] Lanther, U., An $O(N \log N)$ Algorithm for Boolean Mask Operations, *Proceedings of the 18th Design Automation Conference* (1981), 1-8.

[LP] Lipski, W., Jr., and Preparata, F.P., Finding the Contour of a Union of Iso-Oriented Rectangles, *Journal of Algorithms 1* (1980), 235-246.

[NP] Nievergelt, J., and Preparata, F.P., Plane-Sweep Algorithms for Intersecting Geometric Figures, *Communications of the ACM 25* (1982), 739-747.

[O] Ottmann, Th., Geometrische Probleme beim VLSI-Design, *Proceedings of the 1984 GI-Conference* (1984), Springer-Verlag, to appear.

[OW1] Ottmann, Th., and Widmayer, P., On Translating a Set of Line segments, *Computer Vision, Graphics, and Image Processing 24* (1984), 382-389.

[OW2] Ottmann, Th., and Widmayer, P., Solving Visibility Problems by Using Skeleton Structures, *Proceedings of MFCS '84 Springer-Verlag Lecture Notes in Computer Science 176* (1984), 459-470.

[OWW] Ottmann, Th., Widmayer, P., and Wood, D., A Worst-Case Efficient Algorithm for Hidden Line Elimination, University of Waterloo Technical Report CS-82-33 (1983).

[OWo] Ottmann, Th., and Wood, D., The Contour Problem for Polygons, University of Waterloo Computer Science Technical Report CS-84-33 (1984).

[ShH] Shamos, M.I., and Hoey, D., Geometric Intersection Problems. *Proceedings of the 17th Annual IEEE Symposium on Foundations of Computer Science* (1976), 208-215.

[SSW] Soisalon-Soininen, E., and Wood, D., Optimal Algorithms to Compute the Closure of a Set of Iso-Rectangles, *Journal of Algorithms 5* (1984), 199-214.

[SvW] Szymanski, T.G., and Van Wyk, C.J., Space Efficient Algorithms for VLSI Artwork Analysis, *Proceedings of the 20th IEEE Design Automation Conference* (1983), 734-739.

[vLW] van Leeuwen, J., and Wood, D., The Measure Problem for Rectangular Ranges in d-Space, *Journal of Algorithms 2* (1982), 282-300.

[VW] Vaishnavi, V.K., and Wood, D., Rectilinear Line Segment Intersection, Layered Segment Trees, and Dynamization, *Journal of Algorithms 3* (1982), 160-176.

[WOW] Widmayer, P., Ottmann, Th., and Wood, D., Isothetic Solutions for Non-Isothetic Problems. In preparation, 1984.

[Wo] Wood, D., The Contour Problem for Rectilinear Polygons, *Information Processing Letters* (1984), to appear.