Reporting Curve Segment Intersections Using Restricted Predicates

Timothy M. Chan*

March 23, 2000

Abstract

We investigate how to report all k intersecting pairs among a collection of n x-monotone curve segments in the plane, using only predicates of the following forms: is an endpoint to the left of another? is an endpoint above a segment? do two segments intersect? By studying the intersection problem in an abstract setting that assumes the availability of certain "detection oracles," we obtain a near-optimal randomized algorithm that runs in $O(n \log n + n\sqrt{k \log(n^2/k)})$ expected time. In the bichromatic case (where segments are colored red or blue with no red/red or blue/blue intersections), we find a better algorithm that runs in $O((n+k)\log_{2+k/n} n)$ worst-case time, by modifying a known segment-tree method. Two questions of Boissonnat and Snoeyink are thus answered to within logarithmic factors.

Keywords: Segment intersection; Low-degree primitives; Randomized algorithms; Segment trees; Robust computation

1 Introduction

Recently, Boissonnat, Preparata, and Snoeyink [5, 7] revisited a classic geometric problem from a new angle. The problem is *segment intersection*: given n segments in the plane, report all intersecting pairs. Traditional algorithms from the computational geometry literature (e.g., Bentley and Ottmann's standard plane sweep [3], Chazelle and Edelsbrunner's optimal algorithm [10], and randomized incremental construction [12, 16]) all rely on high-degree primitives to compare the x-coordinates of intersection points of two pairs, whereas the trivial quadratic algorithm requires simply a test of whether a pair intersects. The new "angle" is to design efficient algorithms that use only a prescribed set of lower-degree primitives.

The two papers by Boissonnat and Preparata [5] and Boissonnat and Snoeyink [7] obtained successful results for the *line-segment* intersection problem. In the latter work, we have an algorithm that makes $O(n \log^2 n + k \log n)$ evaluations of predicates of the following simplest kinds, where k denotes the number of intersecting pairs, i.e., the output size:

(a) Compare the x-coordinates of two endpoints.

(b) Decide whether three endpoints are given in counterclockwise order.

*Department of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada, tmchan@math.uwaterloo.ca

Predicates (a) and (b) are of algebraic degree 1 and 2 respectively. Boissonnat and Snoeyink's algorithm is a variant of Balaban's optimal algorithm [2], which previously required comparisons of the x-coordinates of an endpoint and an intersection point (a degree-3 operation). In contrast, comparing two arbitrary intersection points has degree 5. The motivation for minimizing the degree in geometric algorithms is well-argued by Liotta, Preparata, and Tamassia [14]: higher-degree primitives are more sensitive to numerical errors, leading to less robust algorithms; they are also more expensive to carry out in practice if exact computation is performed.

The situation can be much worse for x-monotone *curve segments* (despite the fact that most traditional line-segment intersection algorithms do generalize): for example, for semi-circles [7], comparing two intersection points is of degree 6, and comparing an endpoint and an intersection point is of degree 4, while an intersection test is just of degree 2. To determine the predicates necessary to solve the problem for curve segments, it is natural to replace (b) with the following (which is reducible to (b) for line segments):

(b′) Decide whether an endpoint is above a segment (if they are comparable).

If two segments can intersect at most once—the so-called *pseudo-segment* case—then (b′) is sufficient to decide whether two segments intersect. However, by exhibiting a simple configuration of pseudo-segments, Boissonnat and Snoeyink [7] were able to prove an $\Omega(n\sqrt{k})$ lower bound for algorithms that are restricted to use the two predicates (a) and (b′) (they even allowed (b)). Snoeyink asked at the SoCG'98 open-problem session [1] for a matching upper bound on the complexity of pseudo-segment intersection under this model. We settle the question (up to a sublogarithmic factor) by giving a simple randomized algorithm that uses only predicates (a) and (b′) and runs in $O(n \log n + n\sqrt{k \log(n^2/k)})$ expected time, as cited in Boissonnat and Snoeyink's recent paper.

Our time bound actually applies to the segment intersection problem for *general* x-monotone curve segments. If two segments may intersect more than once, it is necessary to allow the intersection test explicitly in addition to (a) and (b′):

(c) Decide whether two segments intersect.

In contrast, the primitives required by traditional algorithms (intersection/intersection and endpoint/intersection comparisons) are trickier to define because of multiple intersection points for a given pair. Besides, the number of primitives used by these algorithms is sensitive to the total number of intersection points, which can be much larger than k, the number of intersecting pairs.

An important special version of the problem—*red/blue segment intersection*—has also been studied from a similar perspective: given n disjoint red segments and n disjoint blue segments, report all k intersecting pairs. Traditional optimal algorithms (such as Mairson and Stolfi's algorithm [15], segment trees [11, 17], and the trapezoid sweep [8]) need to compare the x-coordinates of an endpoint and an intersection point.

Boissonnat and Preparata [5] described an $O((n+k)\log n)$-time algorithm for the red/blue line-segment intersection problem using only predicates (a) and (b). This was subsequently improved by Boissonnat and Snoeyink [7], with an optimal $O(n \log n + k)$-time algorithm for both line segments and pseudo-segments, using predicates (a) and (b′). Boissonnat and Snoeyink's red/blue algorithm is a variant of Chan's trapezoid sweep [8]. A remaining question, posed in their paper, is whether there is an efficient algorithm for general red/blue curve segments under predicates (a), (b′), and (c).

(They noted complications with the sweep approach in case of multiple intersection points.) We comply by giving such an algorithm that runs in $O((n+k)\log_{2+k/n} n)$ time. Note the optimality of this bound when $k = O(n)$ or $k = \Omega(n^{1+\varepsilon})$ for an arbitrarily small constant $\varepsilon > 0$. Our algorithm is a variant of the segment-tree methods of Chazelle *et al.* [11] and Palazzi and Snoeyink [17].

2 Oracle-Based Algorithms

In this section, we prove the claimed $O(n \log n + n\sqrt{k \log(n^2/k)})$ randomized time bound for the general curve-segment intersection problem under restricted predicates. The $O((n+k)\log_{2+k/n} n)$ result in the red/blue case will be left for the next section, although we will see in this section how easy it is to obtain a weaker $O(n \log n + k \log^2 n)$ bound.

All of the algorithms here are quite simple. The approach is to separate the "geometric" component from the "algorithmic" component. The geometric component is referred to as the "oracle" and can be implemented by known plane-sweep methods. The algorithmic component involves straightforward divide-and-conquer, viewing the problem abstractly rather than geometrically and using the oracle as a black box.

This oracle-based approach can be seen alternatively as a reduction of the problem to simpler decision problems (various forms of "detection") that can be solved with simpler primitives. Reducing reporting to decision problems is certainly not an original idea; for example, see [9, Section 2] and the references therein in the context of data structure design. Our applications in the bichromatic case (Sections 2.2 and 2.5), when viewed appropriately, are in fact quite similar to known strategies. However, our uses of randomization in the monochromatic case (Sections 2.3 and 2.4) are unexpected and noteworthy, as they are unlike the standard geometric randomized incremental or random-sampling paradigm [12, 16] and have a rather simple analysis.

We are not aware of similar uses of reductions in robust (degree-driven) geometric computation, although it appears to be a fruitful approach.

2.1 An abstract setting and straightforward divide-and-conquer

Consider an abstract setting involving a universe U of objects and a relation $I \subset U \times U$. Here, $(s,t) \in I$ means that s *intersects* t for two given objects $s, t \in U$, decidable in constant time. The relation I must be anti-reflexive (i.e., an object does not intersect itself). The *intersection problem* is this: given a set $S \subset U$ of n objects, report all intersecting pairs in $S \times S$ (i.e., output $(S \times S) \cap I$). We use k to denote the number of output pairs.

Our algorithms are general in the sense that no knowledge of I is assumed except through the use of certain "detection oracles" as subroutines. The algorithms all follow the intuitive recursive procedure below, which prints all intersecting pairs in $A \times B$. A call to Report(S,S) then solves the intersection problem. Here, the meaning of Test-and-Report() depends on the oracle available.

Algorithm Report(A, B), given lists $A, B \subset U$ of size r:

1. if $r \leq$ constant then trivially report all intersecting pairs in $A \times B$
2. else split A into sublists A_1, A_2 of size $r/2$
3. split B into sublists B_1, B_2 of size $r/2$
4. Test-and-Report(A_1, B_1); Test-and-Report(A_1, B_2)
5. Test-and-Report(A_2, B_1); Test-and-Report(A_2, B_2)

If Test-and-Report() is just Report(), then this recursive algorithm requires $\Theta(n^2)$ time, just like the trivial brute-force algorithm. However, by "testing" before recursion, the running time can be improved.

2.2 Using a bichromatic detection oracle

Our first setting assumes the existence of an $O(n \log n)$-time oracle Detect(A, B) to solve the *bichromatic detection problem*: given sets $A, B \subset U$ of n objects, return true iff there is an intersecting pair in $A \times B$. Here, we can simply define:

Subroutine Test-and-Report(X, Y):

1. if Detect(X, Y) then Report(X, Y)

Lemma 2.1 *Given an $O(n \log n)$-time oracle for the bichromatic detection problem, we can solve the intersection problem in $O(n\sqrt{k}\log(n^2/k))$ time.*

Proof: We examine the recursion tree T of the procedure Report(). At a node v, let (A_v, B_v) denote the argument of the corresponding call to Report(). Let k_i be the number of nodes at level i (with the root having level 0). Clearly, $k_i \leq 4^i$.

By some observation, we can also bound $k_i \leq k$ for any $i \geq 1$. The reason is that with the exception of the root, before we call Report(A_v, B_v), the test Detect(A_v, B_v) must yield true first, so $A_v \times B_v$ must contain an intersecting pair. Furthermore, the $A_v \times B_v$'s are disjoint for nodes v of the same level.

Now, the cost associated with a node at level i is $O((n/2^i)\log(n/2^i))$ for the four calls to Detect(). The total running time is then asymptotically bounded by the following expression, where we choose a positive integer p so that 2^p is near $\sqrt{k}$:

$$\sum_{i=0}^{p-1} 4^i \frac{n}{2^i} \log \frac{n}{2^i} + \sum_{i=p}^{\infty} k \frac{n}{2^i} \log \frac{n}{2^i} = O\left(n 2^p \log \frac{n}{2^p} + k \frac{n}{2^p} \log \frac{n}{2^p}\right) = O\left(n\sqrt{k}\log \frac{n}{\sqrt{k}}\right). \quad (1)$$

□

We can immediately apply this algorithm to the red/blue curve-segment intersection problem under restricted predicates. Although the result will be surpassed, the application is instructive.

Corollary 2.2 *We can solve the red/blue curve-segment intersection problem using an $O(n\sqrt{k}\log(n^2/k))$ number of predicates* (a), (b′), *and* (c) *with $O(n)$ space.*

Proof: Make the relation I assymmetric so that only red segments may intersect blue segments, but not vice versa. In Detect(A, B), we may thus assume that all segments in A are red and all segments in B are blue.

Although Bentley and Ottmann's original plane-sweep algorithm [3] uses predicates we disallow to report *all* intersections, a well-known variant of the sweep [19] *detects* an intersection in $O(n \log n)$ time using our restricted predicates (basically by stopping the sweep as soon as an intersection is found). $\square$

2.3 Using a detection oracle

Our next setting considers a less powerful one-argument oracle Detect(S) for the *detection problem*: given a set $S \subset U$ of n objects, return true iff there is an intersecting pair in $S \times S$. Our test in Test-and-Report() would thus have to be weaker:

Subroutine Test-and-Report(X, Y):

1. if Detect$(X \cup Y)$ then Report(X, Y)

We are only able to carry out the analysis in the expected sense, where it is assumed (as in standard randomized incremental algorithms) that the elements in the initial list S are given in a random permutation.

Lemma 2.3 *Given an $O(n \log n)$-time oracle for the detection problem, we can solve the intersection problem in $O(n\sqrt{k}\log(n^2/k))$ expected time.*

Proof: We proceed as in the proof of the previous lemma; however, we do not necessarily have $k_i \leq k$. Nevertheless, we will show that the expected value of k_i for a given $i \geq 1$ is $O(k)$, so that equation (1) still gives the expected running time by linearity of expectation.

Consider the "full" recursion tree T_0 generated if Test-and-Report() is replaced by Report(). Note that the actual recursion tree T is a subtree of T_0. Fix a node v at level i of T_0, and define k_v to be the number of intersecting pairs in $(A_v \cup B_v) \times (A_v \cup B_v)$. To bound $E[k_v]$, observe that A_v and B_v are random subsets of S of size $r = n/2^i$. The probability that a fixed intersecting pair lies in $(A_v \cup B_v) \times (A_v \cup B_v)$ is $O((r/n)^2) = O(1/4^i)$. Hence, $E[k_v] = O(k/4^i)$.

Now, k_i is always bounded by $\sum_v k_v$, where the sum is over all 4^i nodes v at level i of T_0, because (with the exception of the root) if Report(A_v, B_v) is called, there must be at least one intersecting pair in $(A_v \cup B_v) \times (A_v \cup B_v)$. So, $E[k_i] \leq \sum_v E[k_v] \leq 4^i \cdot O(k/4^i) = O(k)$ as claimed, and the proof is complete. $\square$

By the plane-sweep detection algorithm mentioned in the previous corollary, we immediately obtain a result for general curve-segment intersection that is near-optimal in view of Boissonnat and Snoeyink's $\Omega(n\sqrt{k})$ lower bound [7].

Corollary 2.4 *We can solve the curve-segment intersection problem using an $O(n\sqrt{k}\log(n^2/k))$ expected number of predicates* (a), (b$'$), *and* (c) *with $O(n)$ space.*

2.4 Using a cover oracle

Given $S \subset U$, we say that a subset $C \subseteq S$ is a *cover* if for every intersecting pair $(s,t) \in S \times S$, we have $s \in C$ or $t \in C$. The cover is *nonredundant* if every object in C intersects or is intersected by some object in S. In this subsection, we assume an oracle $\mathrm{Cover}(S)$ that returns a nonredundant cover C of a given set S of n objects. Note that this is stronger than the detection oracle, because an intersecting pair exists in $S \times S$ iff $C \neq \emptyset$. We show that the running time from the previous lemma can be slightly improved using this oracle, by the following change:

Subroutine Test-and-Report(X,Y), given lists $X, Y \subset U$ of size r:

1. $C \leftarrow \mathrm{Cover}(X \cup Y)$
2. if $|C| \leq \log r$
3. then trivially report all intersecting pairs in $[C \times (X \cup Y)] \cup [(X \cup Y) \times C]$
4. else Report(X,Y)

Lemma 2.5 *Given an $O(n \log n)$-time oracle for the nonredundant-cover problem, we can solve the intersection problem in $O(n \log n + n\sqrt{k \log(n^2/k)})$ expected time.*

Proof: We proceed as in the proof of the previous lemma; however, we are able to a better bound on $E[k_i]$. The reason is that if Report(A_v, B_v) is called, we know that there is a nonredundant cover of size exceeding $\log r$, implying that there are at least $\Omega(\log r)$ intersecting pairs in $(A_v \cup B_v) \times (A_v \cup B_v)$. So, $k_i = O(\sum_v k_v / \log(n/2^i))$, where the sum is over all 4^i nodes v at level i of T_0. Thus, $E[k_i] = O(k/\log(n/2^i))$.

Now, the cost of a node at level i remains $O((n/2^i)\log(n/2^i))$, because line 3 of Test-and-Report() requires $O(|C|r) = O(r \log r)$ time. The total expected running time is then asymptotically bounded by the expression below, where we choose a positive integer p so that 2^p is near $\sqrt{k/\log(n/\sqrt{k})}$:

$$\sum_{i=0}^{p-1} 4^i \frac{n}{2^i} \log \frac{n}{2^i} + \sum_{i=p}^{\infty} k \frac{n}{2^i} = O\left(n 2^p \log \frac{n}{2^p} + k \frac{n}{2^p}\right) = O\left(n \log n + n\sqrt{k \log \frac{n}{\sqrt{k}}}\right).$$

□

By observing that the standard plane-sweep algorithm solves the cover problem, we obtain a small improvement of the previous corollary, which yields our best time bound for general curve-segment intersection under restricted predicates.

Corollary 2.6 *We can solve the curve-segment intersection problem using an $O(n \log n + n\sqrt{k \log(n^2/k)})$ expected number of predicates* (a), (b′), *and* (c) *with $O(n)$ space.*

Proof: The implementation for Cover() is conceptually simple: follow the plane-sweep detection algorithm, and whenever an intersection is found, delete one of the segments involved, and continue.

To be a little more precise, the algorithm keeps track of a vertical sweep line and maintains the invariants that (i) no two segments of S intersect to the left of the sweep line, and (ii) no two consecutive segments of S along the sweep line intersect. Events occur only at x-coordinates of endpoints. During an event, (ii) can be maintained by making at most two intersection tests. As soon as an intersecting pair (s,t) is found, we delete s from S. This deletion may induce (ii) to fail again, but an intersection test suffices to see whether this is the case, and if so, we repeat this deletion process for the new intersecting pair. Clearly, there are at most n such deletions, so the total extra work is bounded by $O(n \log n)$. After the algorithm terminates and the entire plane has been swept, the set of all deleted segments forms a nonredundant cover. □

2.5 Using a strong bichromatic detection oracle

Returning to the red/blue case, we observe that a significant improvement in the running time can be obtained with a stronger oracle that detects whether there is an intersecting pair *for each* of the n given objects. Formally, an oracle Strongly-Detect(A, B) for the *strong bichromatic detection problem* returns a pair of subsets (A', B'), where $A' = \{s \in A \mid s \text{ intersects some object in } B\}$, and $B' = \{t \in B \mid t \text{ is intersected by some object in } A\}$. We can define:

Subroutine Test-and-Report(X, Y):
1. $(X', Y') \leftarrow$ Strongly-Detect(X, Y)
2. Report(X', Y')

Lemma 2.7 *Given an $O(n \log n)$-time oracle for the strong bichromatic detection problem, we can solve the intersection problem in $O(n \log n + k \log^2 n)$ time.*

Proof: Note that after line 1 of Test-and-Report(), the sizes of X' and Y' are no greater than the number of intersecting pairs in $X' \times Y'$. Thus, we can bound the cost of a non-root node v in the recursion tree T by the number of intersecting pair in $A_v \times B_v$ times a logarithmic factor. By disjointness of the $A_v \times B_v$'s, the total cost at a level of T is therefore $O(k \log n)$. The total cost, excluding the root, is therefore $O(k \log^2 n)$. □

It turns out that for curve-segment intersection, the strong detection oracle can be efficiently implemented in the red/blue case by plane sweep, giving a result that is optimal to within two logarithmic factors.

Corollary 2.8 *We can solve the red/blue curve-segment intersection problem using an $O(n \log n + k \log^2 n)$ number of predicates* (a), (b′), *and* (c) *with $O(n)$ space.*

Proof: In Strongly-Detect(A, B), we may again assume that all segments in A are red and all segments in B are blue.

To implement Strongly-Detect(), use the same algorithm as in the previous corollary, except whenver an intersecting pair is found, choose the red segment involved to delete. At the end, the set of all deleted segments is A'. We can similarly determine B' by another sweep. □

3 A Segment-Tree Algorithm for Red/Blue Curve Segments

In this section, we focus on the red/blue curve-segment intersection problem under restricted predicates. It is possible to improve the $O(n \log n + k \log^2 n)$-time algorithm from the previous section: for example, by initially "pre-sorting" the segments, we can implement the strong detection oracle in $O(n \log \log n)$ time by using van emde Boas trees. We will not pursue this direction though, since the improvement can be no more than a logarithmic factor, and the algorithm would become more complicated.

Our best result however is obtained using a different idea. The divide-and-conquer algorithm from the previous section is remarkable in that the dividing part (lines 3 and 4 of Report()) is done completely arbitrarily, not exploiting the geometry. However, this advantage of the algorithm is also its weakness and accounts for the loss of efficiency. We are thus prompted to look for more efficient divide-and-conquer strategies that incorporate geometry. There is a such a strategy for red/blue segment intersection in the literature—Chazelle *et al.*'s segment-tree algorithm [11], later simplified by Palazzi and Snoeyink [17]. The question is whether this algorithm can be made to use our restricted predicates. We show that such an adaptation is possible, at the expense of increasing the running time from $O(n \log n + k)$ originally to $O((n+k)\log_{2+k/n} n)$.

3.1 Preliminaries

Let S be the given set of red and blue segments. We say that S is *red-pre-sorted* if we have computed an ordering $g_1, g_2, \ldots$ of all red segments and all blue endpoints, such that whenever g_i is below g_j, we have $i < j$. Blue-pre-sorting can be defined similarly. Palazzi and Snoeyink [17] introduced this notion in their segment-intersection algorithm. They showed how to pre-sort in $O(n \log n)$ time by plane sweep, using only restricted predicates.

Our algorithm attempts to report all intersections inside a given *slab* σ, an open region bounded by two vertical lines, where the x-coordinate of each vertical line is assumed to be the x-coordinate of some endpoint. Without endpoint/intersection comparisons, there is no way to decide whether two arbitrary segments intersect inside σ. For this reason, we introduce a weaker version of the reporting: we say that an algorithm *safely reports* a class $\mathcal{C}$ of intersecting pairs if every pair in $\mathcal{C}$ is printed by the algorithm, and every pair printed intersects (but may or may not be in $\mathcal{C}$). We allow a pair to be printed more than once. Running time is measured in terms of n, the size of S, and k, the overall number of intersecting pairs in $S \times S$ (rather than the size of $\mathcal{C}$). This idea of safe reporting seems well-suited for robust computation, because only one-sided errors are permitted (with no false negatives).

We define some terms to classify types of segments and types of intersections. A segment s is *short* in a slab σ if it has an endpoint in σ. A segment s is *long* in σ if its left endpoint is to the left of σ and its right endpoint is to the right of σ. A pair (s,t) is an *intersecting pair in* σ if s intersects t at some point in the closure of σ. If in addition, s is red and long in σ, we call (s,t) a *red-long intersecting pair in* σ. A blue-long intersecting pair in σ is similarly defined.

3.2 Safely reporting long intersecting pairs

The starting observation of Chazelle *et al.*'s and Palazzi and Snoeyink's algorithms is that it is easy to find all red-long (or similarly blue-long) intersecting pairs in a given slab σ:

1. For blue short segments, we can report their intersections with red long segments in σ by linear scans on the list of red long segments and blue endpoints.

2. For blue long segments, we can report their intersections with red long segments in σ by linear scans if we first merge the list of red long segments and the list of blue long segments according to y-coordinates along $x = x_0$.

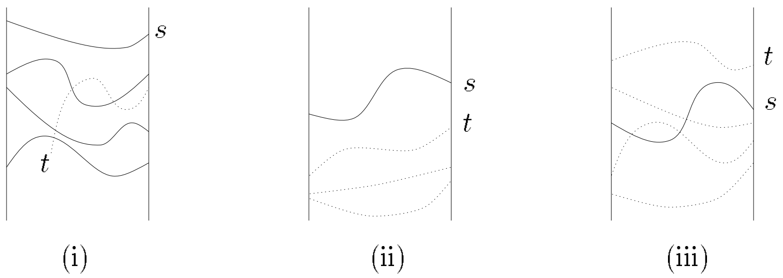

Figure 1: Proof of Lemma 3.1. (Blue segments are dotted.)

We now show that the procedure can be modified for safe reporting using our restricted predicates. Item 1 poses no new problem, but item 2 requires a little more thought, since we do not have a primitive to decide whether a segment is above another at a vertical line if they intersect. The key is to permit pairs that intersect outside the slab to be printed occasionally and charge work to the overall number of intersecting pairs.

Lemma 3.1 *Suppose that S is red- and blue-pre-sorted. Then we can safely report all red-long intersecting pairs in a given slab σ in $O(n+k)$ time, where k is the total number of intersecting pairs in $S \times S$.*

Proof:

1. For each blue short segment t with an endpoint q in σ, identify the red long segment s immediately above q. If s does not intersect t, then stop. Otherwise, print (s,t), set s to be the next red long segment above, and repeat. (See Figure 1(i).) Similarly, proceed in the downward direction.

2. For long blue segments, take the lowest red long segment s and the lowest blue long segment t. Consider the following loop. If s does not intersect t, then stop. Otherwise, print (s,t), set t to be the next blue long segment above, and repeat.

   When the loop terminates, we know that s does not intersect t, so one is above the other (we can tell which case by predicate (b$'$)). If the red segment s is above (Figure 1(ii)), then we can remove t and the blue long segments below, because they have no intersection in σ. If the blue segment t is above (Figure 1(iii)), then we can remove the red segment s, because all of its intersections in σ have been found. We can now restart the procedure with the shorter lists of long segments.

$\square$

The next lemma gives a generalization.

Lemma 3.2 *Suppose that S is red- and blue-pre-sorted. Then given b disjoint slabs $\sigma_1, \ldots, \sigma_b$, we can safely report the union of all red-long intersecting pairs in $\sigma_1, \ldots, \sigma_b$ in $O(b^2 n + k)$ time, where k is the total number of intersecting pairs in $S \times S$.*

Proof: The b slabs are defined by $O(b)$ different x-coordinates. Let Σ to the class of all $O(b^2)$ slabs formed by these x-coordinates. *Assign* a red segment s to the maximal slab of Σ in which it is long.

Our algorithm is simply this: for each $\sigma \in \Sigma$, let S_σ be the set of all red segments assigned to σ and apply the previous lemma to S_σ and the blue segments. Because of disjointness of the S_σ's, there is no overlap in the sets of intersecting pairs in these $O(b^2)$ subproblems. $\square$

3.3 Divide-and-conquer by slabs

The following divide-and-conquer algorithm safely reports all intersecting pairs of S in a given slab σ, using the preceding lemma in line 4. The structure of the recursion is a b-ary version of the familiar *segment tree* [18], although the tree is not explicitly constructed.

Algorithm Safely-Report(S, σ), given a red- and blue-pre-sorted set S of segments and a slab σ, with r endpoints in σ:

```
1. if r ≤ constant
2. then trivially report all intersecting pairs in S × S
3. else divide σ into disjoint subslabs σ_1, ..., σ_b, each with ≤ r/b endpoints
4.      safely report all red-long and blue-long intersecting pairs in σ_1, ..., σ_b
5.      for j = 1, ..., b do
6.          S_j ← all short segments of S in σ_j
7.          Safely-Report(S_j, σ_j)
```

Theorem 3.3 *We can solve the red/blue curve-segment intersection problem using an* $O((n+k)\log_{2+k/n} n)$ *number of predicates* (a), (b′), *and* (c) *with* $O(n+k)$ *space.*

Proof: The correctness of the above algorithm is easy to see: Take any red/blue intersecting pair (s,t) in σ, which must be an intersecting pair in at least one of the subslabs σ_j. If s or t is long in σ_j, then the pair is reported in line 4. If both s and t are short in σ_j, then the pair is reported recursively in line 7.

For the analysis, form the recursion tree T for Safely-Report(). At a node v at level i, let (S_v, σ_v) denote the argument of the call. Let n_v be the size of S_v and k_v be the number of intersecting pairs in $S_v \times S_v$.

Observe that except for the root, all segments of S_v are short in σ_v. Thus, n_v is bounded by the number of endpoints in σ_v. Consequently, the sum $\sum_v n_v$ over all nodes at a given level is bounded by $O(n)$, by disjointness of the σ_v's. We can also bound the sum $\sum_v k_v$ over all nodes at a level by $O(k)$, because a segment is short in at most two slabs in $\{\sigma_v\}$ and thus belongs to at most two of the S_v's, so each intersecting pair is counted at most twice in the sum.

Now, the cost at node v is $O(b^2 n_v + k_v)$ for line 4. The total cost per level is thus $O(b^2 n + k)$. Since the tree has $O(\log_b n)$ levels, the total running time of the algorithm is $O((b^2 n + k)\log_b n))$, including the initial pre-sorting step (note that the subsets S_j in line 7 are already pre-sorted).

It remains to specify the value of the parameter b. The simplest choice would be $b = 2$ (as in Chazelle *et al.*'s and Palazzi and Snoeyink's algorithms). In our analysis, the running time would be $O((n+k)\log n)$. For larger values of k, a better choice is $b \approx n^{\varepsilon/2}$, which yields a bound of $O(n^{1+\varepsilon} + k)$.

The best theoretical result, however, is obtained by the following standard but more complicated modification: use a sequence of values $b = 2, 4, 8, \ldots$ and let the algorithm run for $O(b^2 n \log_b n)$

steps. The algorithm would terminate in completion when $b^2 > k/n$. The overall running time is asympotically bounded by the following, where p is a positive integer so that 4^p is near k/n:

$$\sum_{i=1}^{p} \frac{4^i n \log n}{i} \;=\; O\left(\frac{4^p n \log n}{p}\right) \;=\; O((n+k)\log_{2+k/n} n).$$

□

By definition of safe reporting, an intersecting pair may be reported more than once—in our case, as many as $O(\log_b n)$ times. In order to avoid duplicates, one needs to maintain a dictionary of the reported pairs, which explains the extra $O(k)$ space stated in the above theorem. In contrast, the algorithms in the previous section need only $O(n)$ space.

4 Conclusions

We have given near-optimal output-sensitive algorithms for reporting intersecting pairs of arbitrary curve segments using only the simple predicates of (a) endpoint comparison, (b$'$) aboveness test, and (c) intersection test. In the general case, the complexity of the problem under this predicate restriction is indeed near the conjectured $O(n\sqrt{k})$ bound, if we allow randomization. In the red/blue case, the restriction causes almost no loss of efficiency, if we ignore a logarithmic factor. This is interesting, since with just the intersection test, we have no information to pinpoint the location of the intersection points (or determine how many times a pair intersects).

Some of our approaches, such as reduction (oracles) and safe reporting, could find uses in the design of more robust geometric algorithms.

Besides robustness, another potential advantage of our algorithms is generality. For instance, as in Corollary 2.4, we can report all k intersecting pairs among n convex m-gons in $O((n\log n + n\sqrt{k\log(n^2/k)})\log m)$ expected time; if k is small, this beats a recent result by Gupta, Janardan, and Smid [13] on this problem, which takes $O((nm)^{4/3+\varepsilon} + k)$ time.

Numerous questions are still unanswered. For example:

1. Can the extra logarithmic factors be improved in our two results (Corollary 2.4 and Theorem 3.3) on general curve-segment intersection, as well as in Boissonnat and Snoeyink's result [7] on line-segment intersection?

2. Is there a deterministic algorithm that matches the $\Omega(n\sqrt{k})$ lower bound for curve-segment intersection under restricted predicates?

3. Does the $\Omega(n\sqrt{k})$ lower bound [7] hold for line-segment intersection under predicates (a) and (b$'$)? In other words, is (b) more powerful than (b$'$) for line segments?

4. Is there a nontrivial lower bound for *counting* red/blue line-segment (or pseudo-segment) intersections under predicates (a) and (b) (or (b$'$))? The known $O(n\log n)$ algorithms [11, 17] for line segments require degree-3 primitives.

Note. We learned just recently that Boissonnat and Vignernon [6] have independently obtained an efficient algorithm for red/blue curve-segment intersections using the same set of restricted predicates. Their approach was based on plane sweep, and the running time is $O((n+k)\log n)$, which is as fast as our segment-tree algorithm for small values of k but is slower for $k = \Omega(n^{1+\varepsilon})$. Their original report also claimed to have a deterministic $O(n\sqrt{k}\log n)$ bound for the general case, but the proof there was incorrect [4].

References

[1] P. K. Agarwal and J. O'Rourke. Computational Geometry Column 34. *SIGACT News*, 29(3):27–32, 1998. *Int. J. Comput. Geom. Appl.*, 8:637–642, 1998.

[2] I. J. Balaban. An optimal algorithm for finding segment intersections. In *Proc. 11th ACM Sympos. Comput. Geom.*, pages 211–219, 1995.

[3] J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.*, C-28:643–647, 1979.

[4] J.-D. Boissonnat. Personal communication, February, 2000.

[5] J.-D. Boissonnat and F. P. Preparata. Robust plane sweep for intersecting segments. Tech. Rep. RR-3270, INRIA Sophia Antipolis, 1997. *SIAM J. Comput.*, to appear.

[6] J.-D. Boissonnat and A. Vignernon. Elementary algorithms for reporting intersections of curve segments. Tech. Rep. RR-3825, INRIA Sophia Antipolis, 1999.

[7] J.-D. Boissonnat and J. Snoeyink. Efficient algorithms for line and curve segment intersection using restricted predicates. In *Proc. 15th ACM Sympos. Comput. Geom.*, pages 370–379, 1999.

[8] T. M. Chan. A simple trapezoid sweep algorithm for reporting red/blue segment intersections. In *Proc. 6th Canad. Conf. Comput. Geom.*, pages 263–268, 1994.

[9] T. M. Chan. Fixed-dimensional linear programming queries made easy. In *Proc. 12th ACM Sympos. Comput. Geom.*, pages 284–290, 1996.

[10] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *J. ACM*, 39:1–54, 1992.

[11] B. Chazelle, H. Edelsbrunner, L. J. Guibas, and M. Sharir. Algorithms for bichromatic line segment problems and polyhedral terrains. *Algorithmica*, 11:116–132, 1994.

[12] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[13] P. Gupta, R. Janardan, and M. Smid. Efficient algorithms for counting and reporting pairwise intersections between convex polygons. *Inform. Process. Lett.*, 69:7–13, 1999.

[14] G. Liotta, F. P. Preparata, and R. Tamassia. Robust proximity queries: an illustration of degree-driven algorithm design. *SIAM J. Comput.*, 28:864–889, 1999.

[15] H. G. Mairson and J. Stolfi. Reporting and counting intersections between two sets of line segments. In R. Earnshaw (ed.), *Theoretical Foundations of Computer Graphics and CAD*, NATO ASI Series, Vol. F40, pages 307–326, Springer-Verlag, 1988.

[16] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms.* Prentice-Hall, Englewood Cliffs, N.J., 1994.

[17] L. Palazzi and J. Snoeyink. Counting and reporting red/blue segment intersections. *CVGIP: Graph. Models Image Process.*, 56:304–311, 1994.

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

[19] M. I. Shamos and D. Hoey. Geometric intersection problems. In *Proc. 16th IEEE Sympos. Found. Comput. Sci.*, pages 208–215, 1975.