INFORMATIK

New Deterministic Algorithms for Counting Pairs of Intersecting Segments and Off-Line Triangle Range Searching

Marco Pellegrini

MPI–I–95–1-022 July 1995

FORSCHUNGSBERICHT ■ RESEARCH REPORT

MAX-PLANCK-INSTITUT
FÜR
INFORMATIK

Im Stadtwald ■ 66123 Saarbrücken ■ Germany

The *Max-Planck-Institut für Informatik* in Saarbrücken is an institute of the *Max-Planck-Gesellschaft*, Germany.

ISSN: 0946 - 011X

Forschungsberichte des

Max-Planck-Instituts für Informatik

Further copies of this report are available from:

Max-Planck-Institut für Informatik
Bibliothek & Dokumentation
Im Stadtwald
66123 Saarbrücken
Germany

New Deterministic Algorithms for Counting Pairs of Intersecting Segments and Off-Line Triangle Range Searching

Marco Pellegrini

MPI–I–95–1-022 July 1995

New Deterministic Algorithms for Counting Pairs of Intersecting Segments and Off-Line Triangle Range Searching - extended abstract -

M. Pellegrini

August 22, 1995

Abstract

We describe a new method for decomposing planar sets of segments and points. Using this method we obtain new efficient *deterministic* algorithms for counting pairs of intersecting segments, and for answering off-line triangle range queries. In particular we obtain the following results:

(1) Given n segments in the plane, the number K of pairs of intersecting segments is computed in time $O(n^{1+\epsilon} + K^{1/3}n^{2/3+\epsilon})$, where $\epsilon > 0$ an arbitrarily small constant.

(2) Given n segments in the plane which are coloured with two colours, the number of pairs of *bi-chromatic* intersecting segments is computed in time $O(n^{1+\epsilon} + K_m^{1/3}n^{2/3+\epsilon})$, where K_m is the number of *monochromatic* intersections, and $\epsilon > 0$ is an arbitrarily small constant.

(3) Given n weighted points and n triangles on a plane, the sum of weights of points in each triangle is computed in time $O(n^{1+\epsilon} + \mathcal{K}^{1/3}n^{2/3+\epsilon})$, where $\mathcal{K}$ is the number of vertices in the arrangement of the triangles, and $\epsilon > 0$ an arbitrarily small constant.

The above bounds depend sub-linearly on the number of intersections among segments K (resp. K_m, $\mathcal{K}$), which is desirable since K (resp. K_m, $\mathcal{K}$) can range from zero to $O(n^2)$. All of the above algorithms use optimal $\Theta(n)$ storage. The constants of proportionality in the big-Oh notation increase as ϵ decreases. These results are based on properties of the sparse nets introduced by Chazelle [8].

1 Introduction

Intersection counting and problems in manufacturing. In geometric models of two and three dimensional objects, intersections of objects are important features. Often intersections are related to (desirable or undesirable) facts in the situation that the model should represent. A typical example is in modelling pieces cut from a flat panel, since the same material cannot belong to two pieces, an intersection of two polygons is a mistake to be avoided in the design of the cutting procedure. The importance of detecting, counting and reporting intersections has been recognized in the early days of computational geometry and a substantial research effort has produced several efficient algorithms for this class of problems in two-dimensional space. If we model the boundaries of objects in the plane using segments we have efficient algorithms for *detecting* intersections and *reporting* them [5, 10, 14, 26]. However, important computations in manufacturing would benefit from efficient algorithms for *counting* intersections of segments rather than reporting them. Consider a decomposition of a three-dimensional polyhedral object (e.g. a car engine) obtained by projecting the edges of this object onto a plane, by decomposing the two dimensional arrangement of segments, and by back-projecting the two dimensional decomposition in three-space. Since the direction of projection can be chosen arbitrarily, among several directions one resulting in fewer intersections is preferable. In order to select a good projection, counting intersections is

sufficient. In the design of VLSI and PCB boards the rules for minimum distance among wires (each wire is modelled as several overlapping thin polygons[1]) can be checked by expanding each polygon in every direction by half of the minimum distance. An increase in the number of intersections is a symptom of a possible design error. We should notice that, in this example, detection of one intersection is not sufficient since the overlap of polygons forming a single wire is not a design error, while on the other hand reporting all intersections is not necessary. Wires of two different types (e.g. polysilicon and diffusion in n-MOS technology) are used to fabricate integrated transistors. In this case we can check deign rules for transistors by counting *only* the number of intersections among wires of the two types. An interesting variation on the counting problem is finding the segments contributing the largest number of intersections or the minimum number.

In this paper we describe a new deterministic method for partitioning planar sets of segments and points. Using this method and other tools we are able to obtain improved worst case time bounds for three basic problems. The first problem is that of counting pairs of intersecting segments. The second problem is an important variation of the first one, in which we have two sets of segments (coloured, say, yellow and blue) and we are interested in counting only bi-chromatic pairs of intersecting segments. In this special case we obtain time bounds that improve on those for counting all intersections. The third problem is that of solving off-line simplex range searching queries over sets of points. We obtain a result that holds in any dimension d, but the planar case ($d = 2$) is the one where the benefit over previous methods is more evident. Consider the problem of integrating a sampled scalar function of two variables over a series of *possibly intersecting* planar polygonal domains. The supersampling technique used for antialiasing is a case where such problem arises in the area of computer graphics [17]. If we model each domain as a disjoint union of triangles and the samples as weighted points on the plane, we reduce the integration problem to a case of off-line triangle range-searching for which our efficient algorithm can be used.

Counting Intersections: Summary of results. Bentley and Ottman describe a classical algorithm for reporting intersections of segments in time $O((n+K)\log n)$ and $O(n\log^2 n)$ storage, where K is the number of reported intersections and n the number of segments [5, 27]. Work on this problem culminated with several optimal randomized and deterministic algorithms [10, 14, 26] with $O(n\log n + K)$ time bounds.

At the moment it is not known how to *count points of intersection* more efficiently than by reporting them, if we allow degenerate sets of segments [2] (see [16] for a lower bound argument applicable to quite a large family of algorithms). What we can do efficiently is *counting pairs of intersecting segments.* If the input set is in general position the two measures coincide. If the input set is degenerate then by counting pairs of intersecting segments we always estimate from above the number of intersections. The running time of our algorithms depends on the number of points of intersection. For most of the applications in manufacturing it is sufficient to count intersecting pairs and from now on we will not dwell on the distinction and we will assume that the input set is in general position.

Algorithms for reporting intersections can be used to count pairs of intersecting segments, but they are efficient only when there are few intersections (say $K < n\log n$). When the number of intersections is high, algorithms based on plane partitioning have been proposed [6],[18, 3, 8]. Currently the best deterministic method uses $O(n^{4/3}\log^{1/3} n)$ time and linear storage for a set of n segments [8] (see also [21]). Note that these methods improve on the reporting algorithms only when $K > n^{4/3}$. In this paper we show a new method for counting intersections among segments whose time bound interpolates between those of previously known deterministic algorithms. We obtain the following result:

Theorem 1 *Given a set of n segments forming an arrangement with K vertices we can count the*

[1] Although it is common in VLSI to use axis-parallel rectangles to model wires during high levels of layout design, after the action of some optimization tools such as homotopic compactation [22] the wires are unrestricted polygons.

[2] A set of relatively open segments is in degenerate if there are three segments sharing a point.

pairs of intersecting segments in time $O(n^{1+\epsilon} + n^{2/3+\epsilon}K^{1/3})$, *using linear storage, where* $\epsilon > 0$ *is an arbitrarily small constant.*

This result improves on previous deterministic methods in the range $n^{1+\epsilon} < K < n^{2-\epsilon}$. For $K < n^{1+\epsilon}$ or $K > n^{2-\epsilon}$, our algorithm is within an n^{ϵ} factor from the previously known best algorithms in those ranges. If we allow randomization, then a method in [15] attains expected time $O(n \log n \log K + K^{1/3}n^{2/3}\log^{1/3} n)$.

In the bichromatic case we are given two coloured sets of segments, and we are interested in counting only the *bichromatic* intersections [11, 1, 3]. Agarwal [3] gives an algorithm with time and storage bounds $O(n^{4/3}\log^{1.78} n)$ for any set of segments. By using the approach in [8] it is possible to reduce the running time to $O(n^{4/3}\log^{1/3} n)$ and the storage to linear. For the special case when there are no monochromatic intersections an $O(n \log n)$ algorithm is described in [11]. In this paper we obtain the following result:

Theorem 2 *Given a set of* n *blue segments forming an arrangement with* K_b *vertices, and a set of* n *yellow segments forming an arrangement with* K_y *vertices, we can count the number of bi-chromatic pairs of intersecting segments in time* $O(n^{1+\epsilon} + n^{2/3+\epsilon}(K_y + K_b)^{1/3})$ *and linear storage, where* $\epsilon > 0$ *is an arbitrarily small constant.*

Note that the time bound depends only on the number of monochromatic intersections, and not on the reported number of bi-chromatic pairs of intersecting segments. The bound of our algorithms again interpolates between the two best previous algorithms and improves on both in the range $n^{1+\epsilon} < (K_y + K_b) < n^{2-\epsilon}$.

On-line versus Off-line simplex range searching. Simplex range searching has emerged in recent years as one of the basic problems in computational geometry [30, 13, 19, 31, 12, 24]. The problem in its on-line version is the following. We are given n points in d-dimensional real space E^d (here d is a constant and the multiplicative constants in the bounds depend on d). We wish to answer efficiently the following class of queries: given a query d-simplex[3] q, count the number of points in q. Usually the problem is stated in a slightly more general setting for points with associated weights drawn from a semi-group. In this case the query asks for the cumulative weight of the points in q. For simplicity of exposition we will concentrate on the counting problem, but all the results extend to weighted sets of points. In the off-line case, both the points and the query simplices are known at pre-processing time.

A remarkable result of Chazelle [7] is that, if m units of storage are allowed, than for every data organization scheme[4], there exists a query that costs $\Omega(n/m^{1/2})$ for $d = 2$, and $\Omega(n/m^{1/d}\log n)$ for $d > 2$. The final step of the proof of the lower bound rests on an adversary argument. Given any organization of partial sums in a pre-processed data structure, an adversary is going to choose the hardest possible query. Almost matching upper bounds have been found [12, 25] which are off the lower bound by small polynomial or polylogarithmic factors. Thus, for a sequence of queries on a set of points, the total cost of the sequence can be bounded by multiplying the number of queries times the worst case cost of a query, plus the cost for preprocessing. A sequence of n queries over n points in E^d has a worst cost $\Omega(n^{4/3})$ for $d = 2$ and $\Omega(n^{2d/(d+1)}/\log^{d/(d+1)} n)$ for $d > 2$. In the off-line case, when the queries are known in advance, Chazelle has recently proved a lower bound $\Omega(n^{2-2/(d+1)}(\log n)^{-5/2})$ in the semi-group model [9]. The best algorithms presented in literature for the off-line case [3] matches, up to small factors, the lower bound. In the planar case we could also solve the off-line problem in time, roughly $O((n + \mathcal{K})\log n)$, by using a sweeping line approach and dynamic data structures, where $\mathcal{K}$ is the number of vertices in the arrangement of query triangles.

The type of argument that leads to the lower bounds does not take into account the fact that some sequences might be easier than others, and therefore in many practical applications such bounds may be unduly pessimistic (see e.g. [20]). In this paper we study the effect of sparsity on the simplex range searching problem as well as on the intersec-

[3] A d-simplex is the convex hull of $d+1$ linearly independent points in d-space. For $d = 2$ it is a triangle, for $d = 3$ a tetrahedron.

[4] Such data structure satisfies very general conditions.

tion counting problem. We consider the complexity of the arrangement of query simplices as a measure on the "sparsity"[5] of the sequence of queries. Our main contribution on this problem is that in the off-line case we obtain a bound on the total cost of a sequence that depends on the number of simplices n and on the sparsity parameter $\mathcal{K}$. In particular we prove:

Theorem 3 *Given n points and n simplices in d-dimensional space we count the number of points in each simplex* deterministically *in time $O(\mathcal{K}^{1/(d+1)}n^{d/(d+1)+\epsilon}+n^{f(d)+\epsilon})$, where $\mathcal{K}$ is the number of vertices in the arrangement of query simplices, ϵ an arbitrarily small constant, and $f(d) = (2d^2-4d+1)/(d^2-d-1)$. The storage used is $O(n)$.*

For the case $d = 2$, as a corollary we obtain thus a bound $O(n^{1+\epsilon}+\mathcal{K}^{1/3}n^{2/3+\epsilon})$, which improves on the previous fastest algorithms in the range $n^{1+\epsilon} < \mathcal{K} < n^{2-\epsilon}$. Our result leaves open the question whether in the *on-line* case we can obtain bounds on the cost of answering a sequence of queries S in time depending on the sparsity of S.

The method used. We extend a method for partitioning of a set of simplices, developed in [29] which is based on the properties of sparse nets introduced by Chazelle [8], to the case of simplices and points. The general strategy is a divide and conquer approach. We have an input set S of $(d-1)$-simplices[6] and an input set M of points. We select a subset of S that is used to partition E^d into cells of constant descriptive complexity (also called elementary cells) in such way as to balance the number of elements of S intersecting each elementary cell and the number of points in M inside each cell. We apply the method recursively in each elementary cell. The elementary cells are then organized in a *partition* tree. The gist of the method is to keep a tight control on the size of the partition tree. This tight counting is attained using the sparse nets [8] to select the subset of S. The construction of the partition tree is carried on up to a certain depth. At the leaves of the tree we use known methods that are not sensitive to the sparsity of the input (a similar approach is used in [4] to prove combinatorial bounds). In order to count pairs of intersecting segments we specialize the analysis for $d = 2$ and we add auxiliary computations specific to the intersection counting problem. In the second part of the paper we use the partition tree to solve a variation of Hopcroft's problem, which consists in detecting incidences of points and $(d-1)$-simplices[7] in E^d. Once we have a solution to this problem we add auxiliary operations on the partition tree to obtain the main result on off-line simplex range searching (Theorem 3).

The paper is organized as follows. In Section 2 we recall the main properties of sparse nets. In Section 3 we discuss a method for obtaining a balance partition tree for simplices and points, which is used as a basis by the other algorithms. In section 4 we discuss the algorithm to count the intersections in a set of segments. In Section 5 we discuss how to count efficiently bichromatic intersections of coloured segments. In Section 6 we discuss a variation of Hopcroft's problem for points and simplices. In Section 7 we we extend the solution to Hopcroft's problem to solve the off-line simplex range searching problem.

2 Sparse Nets

Let H be a set of n hyperplanes in Euclidean d-dimensional Space E^d. We assume that H is in *general position*, meaning that exactly d hyperplanes meet in a common point. Let $R \subseteq H$ be a subset of $\rho \leq n$ hyperplanes. For a segment e, let R_e (resp H_e) be the number of hyperplanes in R (resp. H) intersecting e but not containing e. For a d-simplex s, let R_s (resp H_s) be the number of vertices of the arrangement created by R (resp. H) contained in s. Let r be a positive integer number.

Definition 1 *R is a $(1/r)$-approximation for H if, for any segment e: $|R_e/\rho - H_e/n| < 1/r$.*

[5] Naturally one could prefer other ways to capture the informal notion of sparsity. An advantage of choosing the complexity of the arrangement is that this measure is invariant under projective transformations.

[6] A $(d-1)$-simplex is the convex hull of d linearly independent points in d-space. It is segment for $d = 2$, a triangle for $d = 3$.

[7] A randomized method for the planar case is given in [15].

Definition 2 *R is a $(1/r)$-net for H if for any segment e, $H_e > n/r$ implies $R_e > 0$.*

Definition 3 *R is a sparse $(1/r)$-net for (H, s) if for any segment e, $H_e > n/r$ implies $R_e > 0$; and the following inequality holds: $R_s \leq 4(\rho/n)^d H_s$.*

Definition 4 *A $(1/r)$-cutting for H is a partition of E^d into interior-disjoint simplices such that any simplex meets at most n/r of the hyperplanes in H. The number of simplices in the partition is called the size of the cutting.*

Let s be any d-simplex in E^d. We denote by $H(s)$ the subset of hyperplanes of H intersecting s. Computing a sparse net for $(H(s), s)$ directly can be time-consuming, therefore first an approximation A of $H(s)$ is generated, then a strong net for (A, s) is computed. Let r_0 be a constant and i an integer such that $n/r_0^{i-1} > |H(s)| > n/r_0^i$. Moreover define $\rho_0 = r_0^i |H(s)|/n$ and $\rho = \rho_0 \log \rho_0$. The following lemma summarizes some important properties of this construction:

Lemma 1 ([8]) *(i) Let A be a $(1/(2d\rho_0))$-approximation of H and R a sparse $(1/(2d\rho_0))$-net for (A, s), then the following inequality holds: $R_s \leq 4(\rho/|H(s)|)^d H_s + 4\rho^d/\rho_0$. (ii) A canonical triangulation of the sparse net R of Lemma 1 is a $1/r_0$-cutting for $H(s)$ in s and it has size $O(\rho^{d-1} + R_s)$. (iii) The approximation A and the sparse net R at (i) is computed in time $O(|H|)$.*

We need two additional important concepts. We fix once and for all a vertical direction in E^d. A $(d-1)$-simplex t *partially covers* a d-simplex s if t intersects s and some $(d-2)$-face of t intersects s. If $(d-1)$-simplex t partially covers a d-simplex s then the vertical projection of some $(d-2)$-face of t will intersect the vertical projection of s. A $(d-1)$-simplex t *completely covers* a d-simplex s if t intersects s and no $(d-2)$-face of t intersects s.

3 Partitioning points and $(d-1)$-simplices in E^d

In this Section we describe a general algorithm for partitioning sets of points and $(d-1)$-simplices. The objective is to maintain a tight control on the number of simplices and points incident to any cell associated with nodes of the partition tree, as well as a tight control on the size of the partition tree.

We are given a set M of m points in E^d and a set S of n $(d-1)$-simplices in general position. We denote with K the number of d-tuples of simplices having a point in common. We build a sequence of sets $C_0, .., C_l$ where $l = \log_{r_0} r$ and r_0 is a suitable constant. The set C_i is a collection of quadruples $(s, P(s), Q(s), M(s))$ where s is d-simplex in E^d or a d-cylinder (i.e. a degenerate d-simplex with a vertex at infinity), also called an elementary cell. $P(s)$ is the subset of simplices in S *partially covering* s, $Q(s)$ is the subset of simplices in S *covering* s, $M(s)$ is the set of points of M in s. In each set C_i the union of the elementary cells is E^d. The invariants maintained over the sets C_i are: $|P(s)| \leq n/r_0^i$, $|Q(s)| \leq nr_0/r_0^i$, and $|M(s)| \leq m/r_0^{i(d-1)}$. The algorithm to construct C_k from C_{k-1} works in three main phases.

(1) Take s in C_{k-1} and build a sparse net of Lemma 1 for the hyperplanes spanning simplices in $Q(s)$, restricted to s. Triangulate the net thus obtaining a set of simplices $(\sigma, \emptyset, Q(\sigma), M(\sigma))$. By induction we assume that $|Q(s)| \leq r_0 n/r_0^{k-1}$. We choose the parameter $\rho_0' = r_0^{k-1}|Q(s)|/n$ and thus obtain $Q|(\sigma)| \leq r_0 n/r_0^k$ as follows from Lemma 1. Also, $\rho_0' \leq r_0$.

(2) Take s in C_{k-1} and $P(s)$. If $P(s)$ is not empty project the simplices in $P(s)$ and s in $(d-1)$-dimensional space obtaining a set $P'(s)$ and a $(d-1)$-simplex s'. Extend each $(d-2)$-face of simplices in $P'(s)$ into a full hyperplanes and make a sparse net of Lemma 1 for this set of hyperplanes. Triangulate the net and obtain a set of $(d-1)$-dimensional elementary cells. Extend the elementary cells vertically in d-space within s obtaining cylinders $(\eta, P(\eta), Q(\eta))$. Inductively $|P(s)| \leq n/r_0^{k-1}$ and we choose $\rho_0'' = r_0^k |P(s)|/n$ so as to obtain $|P(\eta)| \leq n/r_0^k$ and $|Q(\eta)| \leq |P(s)| \leq r_0 n/r_0^k$. Also $\rho_0'' < r_0$. We further decompose the cylinder η into d-simplices. We have a constant number of them and they satisfy the two invariants.

(3) We compute, for each cell σ built in phase (1) and (2), the points in $M(\sigma)$. If $M(\sigma)$ does not satisfy the invariant we can split σ into subcells with at

most $m/r_0^{k(d-1)}$ points. For each cell $s \in C_{k-1}$ we introduce at most r_0^{d-1} new cells in this phase.

At the end of the three phases we collect all the cells in the set C_k. The set C_k satisfies the three invariants. The simplices produced in are organized in a two-level search tree. The search trees on simplices built in phase (1) (resp. (2)), with the refinement in phase (3), will be called Q-trees (resp. P-trees). The base case C_0 corresponds to the whole space E^d and trivially satisfies the invariants.

3.1 Analysis of the algorithm

Let us denote by s' the vertical projection of a simplex s onto $(d-1)$-dimensional subspace, and by R' the sparse net for $P'(s)$. We use Lemma 1 to bound the number of simplices obtained at each iteration of the algorithm. We obtain the following inequalities in which we denote with $c_1, c_2, \ldots$ absolute multiplicative constants that depend on d. Let $\rho_0 = \rho_0' + \rho_0''$. Notice that $\sum_{s \in C_{k-1}} P'(s)_{s'} = O(n^{d-1})$. We denote with K the total number of points which are the intersection of d $(d-1)$ simplices in S. From the definition we have $K \leq \binom{n}{d}$. We define $z(r_0) = \log^{O(1)} r_0$.

$$
\begin{aligned}
|C_k| \leq & \sum_{s \in C_{k-1}} c_1[\rho_0'^{d-1} \log^{d-1} \rho_0' + R_s + r_0^{d-1}] + \\
& \sum_{s \in C_{k-1}} c_2[\rho_0''^{d-2} \log^{d-2} \rho_0'' + R'_{s'} + r_0^{d-1}] \\
\leq & \sum_{s \in C_{k-1}} c_3 z(r_0)[r_0^{d-1} + (r_0^k/n)^d Q(s)_s + r_0^{d-1}] + \\
& \sum_{s \in C_{k-1}} c_4 z(r_0)[r_0^{d-2} + (r_0^k/n)^{d-1} P'(s)_{s'} + r_0^{d-1}] \\
\leq & \sum_{s \in C_{k-1}} c_5 z(r_0)[r_0^{d-1} + (r_0^k/n)^d Q(s)_s + \\
& (r_0^k/n)^{d-1} P'(s)_{s'})]
\end{aligned}
$$

Finally, we obtain the following recursive inequality in the variable k: $|C_k| \leq c_6 z(r_0)[r_0^{d-1}|C_{k-1}| + r_0^{kd}(K/n^d) + r_0^{k(d-1)}]$. A similar recursive inequality is solved in [29](journal version) with the following bound:

Lemma 2 *$|C_k| \leq D r_0^{k(d-1+\epsilon)} + F(K/n^d) r_0^{kd}$, where F and D are constants with respect to k, n and K.*

We continue the construction until k reaches the value $l = \log_{r_0} r$, where r is a value to specified later. The total number of elementary cells is $\sum_{k=0}^{r} |C_k|$, which is bounded by $\sum_{k=0}^{l} [D r_0^{k(d-1+\epsilon)} + F(K/n^d) r_0^{kd}]$. This is a summation of geometric series of ratio $r_0^{d-1+\epsilon}$ and r_0^d, which is proportional to the last term of the series. Thus we have a bound $O(D r_0^{l(d-1+\epsilon)} + F(K/n^d) r_0^{ld}) = O(r^{d-1+\epsilon} + (K/n^d) r^d)$ on the size of the search tree. The time spent on each d-simplex in the search tree is linear in the number of $(d-1)$-simplices intersecting it by Lemma 1. So the construction of the nets in phases (1) and (2) has the following cost: $\sum_{k=0}^{l} (n r_0 / r_0^k) |C_k|$, which is bounded by $\sum_{k=0}^{l} (n r_0 / r_0^k)[D r_0^{k(d-1+\epsilon)} + F(K/n^d) r_0^{kd}]$, which is equal to $\sum_{k=0}^{l} (n r_0)[D r_0^{k(d-2+\epsilon)} + F(K/n^d) r_0^{k(d-1)}]$.

Since these are sums of geometric series of ratio greater than 1 they are proportional to the last element of the summation. We obtain bound: $O(n[r^{d-2+\epsilon} + (K/n^d) r^{d-1}])$ for all the time used to construct the hierarchy. The location of the points and the splitting of phase (3) is done in time proportional to the number of points $|M(s)|$. Phase (3) over the whole algorithm costs: $\sum_{k=0}^{l} (m/r_0^{k(d-1)}) |C_k|$ which is bounded by $\sum_{k=0}^{l} (m/r_0^{k(d-1)})[D r_0^{k(d-1+\epsilon)} + F(K/n^d) r_0^{kd}]$, which is equal to $\sum_{k=0}^{l} m[D r_0^{k\epsilon} + F(K/n^d) r_0^k]$. Again we have sums of geometric series of ratio r_0^ϵ and r_0, so we obtain a bound $O(m r^\epsilon + m(K/n^d) r)$. This analysis accounts for the construction of a partition tree that is common to several algorithms in this paper.

4 Counting intersections of segments

Given a set S of n segments in E^2, let $\mathcal{A}(S)$ be the arrangement formed by the segments in S and let K be the number of vertices in the arrangement. We specialize the construction of section 3 for $d = 2$ and we use as the set M of input points the end-points of the segments in S. Thus, by using the algorithm of the previous section we obtain a partition tree with leaves associated with cells σ in C_l. The following conditions hold at the leaves:

$|P(\sigma)| \leq n/r$, $|Q(\sigma)| \leq nr_0/r$, $|M(\sigma)| \leq n/r$, and $|C_l| = O(r^{(1+\epsilon)} + (K/n^2)r^2)$.

We build the search tree up to level $l = \log_{r_0} n^2/K$, corresponding to $r = n^2/K$. We consider several types of intersections that can be accounted for using different techniques on the search tree. The main difficulty is in the fact that in the construction of the partition tree we separate covering segments from the partially covering ones. Thus we will describe a scheme for merging back again those sets of segments.

We discuss how to count intersections in a set $P(s)$, for a generic cell s. The results follows by starting with the whole space, $s = E^d$. Let $P(s)$ be the set of simplices partially covering s. We then apply phase (2) of the main algorithm obtaining a set of elementary cells Σ whose union is s. For each cell σ we obtain sets $P(\sigma)$ and $Q(\sigma)$. We then have to compute: The set $QQ(\sigma)$ of intersections between a segment in $Q(\sigma)$ and a segment in $Q(\sigma)$, clipped in σ. The set $PQ(\sigma)$ of intersections between a segment in $Q(\sigma)$ and a segment in $P(\sigma)$, clipped in σ. The set $PP(\sigma)$. of intersections between a segment in $P(\sigma)$ and one in $P(\sigma)$, clipped in σ. Clearly: $|PP(s)| = \sum_{\sigma\in\Sigma} |QQ(\sigma)| + |PQ(\sigma)| + |PP(\sigma)|$. Next we describe how to compute each type of intersection: (1) The intersections in $QQ(\sigma)$ can be computed in time $O(|Q(\sigma)| \log |Q(\sigma)|)$ by using a method in [2, Lemma 3.1], which is based on counting inversions in permutations. (2) The intersections in $PP(\sigma)$ are counted recursively using the method we are describing for $PP(s)$, unless σ is a leaf of the upper search tree. (3) If σ is a leaf of the upper search tree, we apply directly the method in [8], which uses time $O(|Q(\sigma)|^{4/3}\log^{1/3}|Q(\sigma)|)$ and $O(|Q(\sigma)|)$ storage. (4) The intersections $PQ(\sigma)$ are computed in the following way. Let us consider the subtree rooted at σ and built by repeated applications of phases (1) and (3) of the algorithm of Section 3, where the points $M(\sigma)$ that we have used in phase (3) are are the endpoints of segments in $P(\sigma)$. Each point in $M(\sigma)$ will appear in one cell for each level of the Q-tree rooted at σ. For each such cell and for each point, we find the sibling cells intersecting the segment in $P(\sigma)$ incident to that point. Since the Q-tree has degree bounded by a constant, each segment of $P(\sigma)$ is associated with at most a constant number of cells for each level of the Q-tree. Moreover, for each level the associate segments cross all but at most two of the cells to which it has been associated at that level. Now, we can count the intersections between Q-sets at a cell and crossing associated segments again by using Lemma 3.1 in [2]. Thus accounting for these intersections will cost overall at most $O((|P(\sigma)| + |Q(\sigma)|)\log^2 n)$ over the whole Q-tree rooted at σ. If an associated segment is short at a level of the Q-tree, we deal with that segment recursively on the next level of the Q-tree rooted at that cell. (5) When a segment from $P(\sigma)$ is associated to a leaf η of the Q-tree and it is short for that leaf, then we use at that leaf a non-sensitive bi-chromatic method [3, 8] on the set $Q(\eta)$ and on the set of segments in $P(\sigma)$ incident to points in $M(\eta)$. From the invariant we have that the input to the non-sensitive method at a leaf is of size at most $2n/r$.

The proof of correctness of this method derives from the fact that all possible pair of intersecting segments are accounted for. The details are routine and are left as an exercise. Next we analyze the cost of the algorithm. The total cost for setting up the search tree is: $nr^\epsilon + rn(K/n^2) + nr^\epsilon + n(K/n^2)r$. For $r = n^2/K$, we obtain a cost: $O(n^{1+\epsilon})$. The cost of the accounting on all levels of the search tree except the leaves is: $\sum_{k=0}^{l}[(nr_0/r_0^k)^{1+\epsilon} + (n/r_0^k)^{1+\epsilon}]|C_k|$, which is bounded by $\sum_{k=0}^{l}[(nr_0/r_0^k)^{1+\epsilon}][Dr_0^{k(1+\epsilon)} + F(K/n^d)r_0^{k2}]$, which is equal to $\sum_{k=0}^{l}(nr_0)[Dr_0^0 + F(K/n^d)r_0^{k(1-\epsilon)}]$. Since these are sums of geometric series of ratio grater than 1 they are proportional to the last element of the summation. We obtain bound: $O(n[\log r + (K/n^2)r^{1-\epsilon}])$ which, for $r = n^2/K$ is $O(n^{1+\epsilon})$.

The cost incurred at the leaves of the partition tree is: $\sum_{\sigma\in C_k}(n/r)^{4/3}\log^{1/3} n$, which is bounded by $(r^{1+\epsilon} + (K/n^2)r^2)(n/r)^{4/3}\log^{1/3} n$. Again, substituting our choice of r we obtain a bound $O(n^{2/3+\epsilon}K^{1/3})$.

We can reduce the working storage to linear in n by building the partition tree in a depth first manner. The working storage is used to store one path from the root to one leaf of the partition tree, plus all siblings of the nodes on the path, together with the associated sets Q, P and M. The size of these sets decreases geometrically, therefore the sum of all

the size of sets along the path is linear in n. The result claimed in the introduction is almost proved, except for the issue of how the algorithm guesses the correct value of the parameter r, which depends on the unknown value of K. We overcome this problem by using a well known trick of doubling. We start with guess $K_0 = n$ and we run the algorithm until it exceeds the time bound. At this point we double the guess using the general rule $K_i = 2K_{(i-1)}$. There are at most a logarithmic number of guesses before the algorithm terminates having counted all of the pairs of intersecting segments. The time bound is unchanged except for a slightly higher value of ϵ. This trick will be used also for the other results in this article. The above discussion constitutes the proof of Theorem 1.

5 Bi-chromatic intersections

In the bichromatic case we show an algorithm whose time bound depends only on the number of mono-chromatic intersections, and thus it may be much faster than the general algorithm to count segment intersections, if the two coloured input sets are separately sparse. We are give two sets of n segments which we colour yellow and blue. The problem is to count the number K_{yb} of yellow-blue intersections. We give a method whose time bound does not depend on the bichromatic intersections, but on the number of mono-chromatic intersections $K_y + K_b$. More in detail, for a cell s, we will consider the sets: $P_y(s)$ of yellow simplices partially covering s, $Q_y(s)$ of yellow simplices covering s, $M_b(s)$ of end-points of blue segment in s, $P_b(s)$ of blue simplices partially covering s, $Q_b(s)$ of blue simplices covering s, and $M_y(s)$ of end-points of yellow segments in s.

We have these types of intersections: (i) The set $Q_yQ_b(s)$ of intersections between a segment in $Q_b(s)$ and a segment in $Q_y(s)$, clipped in s. (ii) The set $P_yQ_b(s)$ of intersections between a segment in $Q_b(s)$ and a segment in $P_y(s)$, clipped in s. Symmetrically we have also a set $P_bQ_y(s)$ (iii) The set $P_yP_b(s)$. of intersections between a segment in $P_y(s)$ and one in $P_b(s)$, clipped in s.

Let us take a cell s and two sets of partially covering segments, $P_y(s)$ and $P_b(s)$ we describe a method for counting the set of intersections $P_yP_b(s)$. The final bound will be derived by setting $s = E^2$. We build a cell decomposition for $P_y(s)$ and one separate for $P_b(s)$ by using phases (2) and (3) of the main algorithm. We obtains sets Σ_y and Σ_b which are sets of vertical strips. We merge the two decompositions obtaining a set Σ of cells. We then have the sets $P_y(\sigma)$, $Q_y(\sigma)$, $P_b(\sigma_b)$ and $Q_b(\sigma_b)$ for all $\sigma \in \Sigma$. We have that the number of bichromatic intersections $P_bP_y(s)$ is: $|P_yP_b(s)| = \sum_{\sigma\in\Sigma} |P_yQ_b(\sigma)| + |Q_yP_b(\sigma)| + |Q_yQ_b(\sigma)| + |P_yP_b(\sigma)|$.

Since Σ_y and Σ_b are just back-projections of 1-dimensional arrangement, their intersection has size at most $2r_0$. This fact helps in keeping the branching factor of the partition tree small. We build the search tree keeping the following invariants: $P_y(s) \leq n/r_0^i$, $Q_y(s) \leq nr_0/r_0^i$, $M_b(s) \leq n/r_0^i$, $P_b(s) \leq n/r_0^i$, $Q_b(s) \leq nr_0/r_0^i$, and $M_y(s) \leq n/r_0^i$.

Those invariants are satisfied by repeatedly applying phases (1), (2) and (3) of the main algorithm, where we process separately yellow points with blue segments on one side and and blue points with the yellow segments on the other side. It is easy to see that after phases (1), (2) and (3) all the six invariants are satisfied. We denote with R_{ys} (resp. R_{bs}) the number of vertices of a sparse net of yellow (resp. blue) segments in s. The number of cells produced in the three phases satisfies this inequality:

$$|C_k| \leq \sum_{s\in C_{k-1}} c_1[\rho_0' \log \rho_0' + R_{bs} + r_0] + \sum_{s\in C_{k-1}} c_1[\rho_0' \log \rho_0' + R_{ys} + r_0] + \sum_{s\in C_{k-1}} c_2 r_0$$

We use reductions similar to those used in the previous section and we obtain the following recursive inequality in the variable k: $|C_k| \leq c_6 z(r_0)[r_0|C_{k-1}| + r_0^{2k}((K_y + K_b)/n^2) + r_0^k]$.

The solution the same as in Lemma 2 for $d = 2$, with the difference that K is now to be interpreted as $K_y + K_b$, i.e. the number of monochromatic intersections. Now we can augment the partition tree with the methods for accounting of the different intersections. The intersections in $Q_yQ_b(\sigma)$ can be computed in time $O(|Q(\sigma)| \log |Q(\sigma)|)$ by using a method in [2, Lemma 3.1]. The intersections in $P_yP_b(\sigma)$

are counted recursively using the method we are describing for $P_y P_b(s)$, unless σ is a leaf of the upper search tree. The method for counting intersections described in [8] can be easily changed in a method for counting bi-chromatic intersections within the same time and storage bounds. If σ is a leaf of the upper search tree, we apply directly the variation of the method in in [8] in order to count pairs in $P_y P_b(\sigma)$.

The intersections $P_y Q_b(\sigma)$ are computed in the following way. Let us consider the subtree rooted at σ and built by repeated applications of phase (1) and (3) of the algorithm, where we have traced also the set of points $M_y(\sigma)$ (recall that these are the endpoints of segments in $P_y(\sigma)$). Each point in $M_y(\sigma)$ will appear in one cell for each level of the Q-tree rooted at σ. For each such cell and for each point, we find the sibling cells intersecting the segment in $P_y(\sigma)$ incident to that point. Since the Q-tree is of constant degree, each segment of $P_y(\sigma)$ is associated with a constant number of cells for each level of the Q-tree. Moreover, for each level the associate segments cross all but at most two of the cells to which it has been associated at that level. As before, we can count the intersections between Q-sets at a cell and crossing associated segments. Thus this accounting will cost overall at most $O((|P_y(\sigma)| + |Q_b(\sigma)|)\log^2 n)$ over the whole Q-tree rooted at σ. If an associated segment is short at a level of the Q-tree than we deal with that segment recursively on the next level of the Q-tree rooted at that cell.

When a segment from $P_y(\sigma)$ is associated to a leaf η of the Q-tree and it is short for that leaf, then we use at that leaf a non-sensitive bi-chromatic variation of the algorithm in [8] on the $Q_b(\eta)$ and on the set of segments in $P(\sigma)$ incident to points in $M_y(\eta)$. From the invariant we have that the input to the non-sensitive method at a leaf is of size at most $2n/r$.

An analysis similar to the monochromatic case accounts for all of cost associated to tracing the four types of intersections in the tree. With the caveat the K is now only the number of monochromatic intersections, the time analysis is exactly as in the case of non coloured segments. Again, by expanding the partition tree in depth first manner and by storing at any given time only the data relative to one path from the root to a leaf, plus the siblings of nodes on the path, we can achieve linear storage. This concludes the proof of Theorem 2.

6 Incidence of points and $(d-1)$-simplices

In the second part of this paper we derive a result for off-line simplex range searching. As a preliminary step we solve the problem of detecting an incidence between $(d-1)$-simplices and points in d-space. We assume $d \geq 2$. Let us consider the construction of the partition tree in the Section 3. We have a set of leaves associated with cells in C_l, where each leaf has the cell σ has associated sets $P(\sigma)$, $Q(\sigma)$ and $M(\sigma)$, with the following invariants: $|P(\sigma)| \leq n/r$, $|Q(\sigma)| \leq nr_0/r$, and $|M(\sigma)| \leq m/r^{(d-1)}$.

Let us set $q = r^{d-1+\epsilon} + (K/n^d)r^d$, which is proportional to the number of leaves of the tree. The number of simplices cutting each cell at a leaf is $O(n/r)$. We take the points at each leaf and we divide them in groups of equal size m/q. This operation takes time[8] $O(qm/r^{(d-1)})$. For the remainder of this chapter, in order to keep the calculation simple we assume that $m = n$ and we drop constant, logarithmic and epsilon factors. We will take them into account in the statement of the final result. For each group of points and each group of simplices we extend the simplices into hyperplanes. The incidence of a $(d-1)$-simplex and points can be expressed as a conjunction of inequalities involving bilinear forms whose terms are functions of the coordinates of the point and of the linear subspaces supporting faces of the $(d-1)$-simplices (see e.g [28] for a general treatment of these classes of problems). Using results in [28] and the deterministic cuttings in [23] it is possible to find whether in a point of a set of p_i points is incident to a $(d-1)$-simplex in a set of h_i simplices deterministically in time $O(p_i^{d/(d+1)+\epsilon} h_i^{d/(d+1)} + p_i^{1+\epsilon} + h_i \log^{d+2} p_i)$. Also, by applying a depth-first strategy in the order of executions of subproblems in the algorithm in [28], it is possible to use only $O(n+m)$ storage. The total cost for building the partition tree is: $mr^\epsilon + rm(K/n^d) + nr^{d-2+\epsilon} + n(K/n^d)r^{d-1}$. At the leaves of the partition tree we spend time:

[8] A rough estimate but sufficient for our purposes

$mq/r^{d-1} + q[(m/q)^{d/(d+1)}(n/r)^{d/(d+1)} + m/q + n/r]$. Thus modulo constant, logarithmic and ϵ-factors we have a cost: $nr^{d-2} + n(K/n^d)r^{d-1} + nq/r + n^{2d/(d+1)}q^{1/(d+1)}/r^{d/(d+1)}$. We obtain the best performance by finding the value for the parameter r that balances the several costs. To simplify the analysis we consider two cases.

Case $r^{d-1} > (K/n^d)r^d$. In this case $q < 2r^{d-1}$, therefore $nq/r = nr^{d-2}$. Also, since $r < n$, we have. and $r^{d-1} < nr^{d-2}$. Therefore the cost is: $nr^{d-2} + n^{2d/(d+1)}q^{1/(d+1)}/r^{d/(d+1)}$. The optimal trade-off is for $r = n^{(d-1)/(d^2-d-1)}$, and the cost is $O(n^{(2d^2-4d+1)/(d^2-d-1)})$.

Case $r^{d-1} < (K/n^d)r^d$. In this case $q = (K/n^d)r^d$, therefore $nq/r = n(K/n^d)r^{d-1}$. Therefore the cost is: $n + n(K/n^d)r^{d-1} + n^{2d/(d+1)}q^{1/(d+1)}/r^{d/(d+1)}$. The optimum is given by: $r^{d-1} = n^{(d-1)/(d+1)}(K/n^d)^{-d/(d+1)}$. The total cost is thus: $K^{1/(d+1)}n^{d/(d+1)}$. We keep in memory at any given time only one path from root to a leaf and we explore the partition tree in depth-first manner. Thus the storage is only linear in n. Summarizing:

Theorem 4 *Given n points and n $(d-1)$-simplices in E^d, we can determine whether any $(d-1)$-simplex is incident to any point in time $O(K^{(1/(d+1)}n^{d/(d+1)+\epsilon} + n^{f(d)+\epsilon})$, where $f(d) = (2d^2-4d+1)/(d^2-d-1)$, and K is the number of vertices in the arrangement of the $(d-1)$-simplices, $\epsilon > 0$ an arbitrarily small constant. The storage is linear in n.*

The highest gain over the non-sensitive method is attained for low-dimensional spaces. For $d = 2$, we have $f(2) = 1$; for $d = 3$ we have $f(3) = 7/5$.

7 Off-line simplex range searching

We present the algorithm for solving off-line simplex range searching queries as a modification of the algorithm in the previous section. At the leaves of the partition tree we need a non-sensitive method. The inclusion of a point in a d-simplex can be expressed as a Boolean formula involving only bilinear expressions and inequalities. By applying the theory in [28] and the deterministic cuttings in [23], we can derive easily a method that uses for n simplices and m points time $O(m^{d/(d+1)+\epsilon}n^{d/(d+1)} + m^{1+\epsilon} + n\log^{d+2} m)$ and $O(n+m)$ storage.

We consider the algorithm in the previous section using the facets of the input d-simplices as the set of $(d-1)$-simplices. Also, for each cell $\sigma \in C_k$ we compute the list of d-simplices that contain the cell σ, but do not contain the parent of σ. It is easy to see that if s is the parent of σ, then such set of d-simplices must have facets in $Q(s) \cup P(s)$. Thus the total length of such lists and the total time to compute them is asymptotically $O(nr^{d-2+\epsilon} + n(K/n^d)r^{d-1})$. In the data structure so modified we can collect for each d-simplex then number of points contained in it. By using a depth first strategy we can keep the storage linear. This concludes the proof of Theorem 3.

References

[1] P. K. Agarwal and M. Sharir. Red-blue intersection detection algorithms, with applications to motion planning and collision detection. *SIAM J. Comput.*, 19:297–321, 1990.

[2] P.K. Agarwal. Partitioning arrangements of lines I: An efficient deterministic algorithm. *Discrete & Computational Geometry*, 5:449–483, 1990.

[3] P.K. Agarwal. Partitioning arrangements of lines II: Applications. *Discrete & Computational Geometry*, 5:533–573, 1990.

[4] B. Aronov, H. Edelsbrunner, L.J. Guibas, and M. Sharir. The number of edges of many facets in a line segment arrangement. *Combinatorica*, (3):261–274, 1992.

[5] J.L. Bentley and T. Ottman. Algorithms for reporting and counting geometric intersections. *IEEE Trans. on Computers*, C-28:643–647, 1979.

[6] B. Chazelle. Reporting and counting segment intersections. *J. Comput. Syst. Sci.*, 32:156–182, 1986.

[7] B. Chazelle. Lower bounds on the complexity of polytope range searching. *J. Amer. Math. Soc.*, 2:637–666, 1989.

[8] B. Chazelle. Cutting hyperplanes for divide and conquer. *Discrete & Computational Geometry*, 9:145–158, 1993.

[9] B. Chazelle. Lower Bounds for Off-Line Range Searching. Manuscript, March 1995.

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

[11] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir. Algorithms for bichromatic line segment problems and polyhedral terrains. Report UIUCDCS-R-90-1578, Dept. Comput. Sci., Univ. Illinois, Urbana, IL, 1989.

[12] B. Chazelle, M. Sharir, and E. Welzl. Quasi-Optimal Upper Bounds for Simplex Range Searching and New Zone Theorems. *Algorithmica*, (8):407–429, 1992.

[13] B. Chazelle and E. Welzl. Quasi-optimal range searching in spaces of finite VC-dimension. *Discrete Comput. Geom.*, 4:467–489, 1989.

[14] K. Clarkson and P. Shor. Applications of random sampling in computational geometry II. *Discrete & Computational Geometry*, 4:387–422, 1989.

[15] M. de Berg and O. Schwarzkopf. Cuttings and Applications. Tech. Report TR CS-92-26. Dept. of Computer Science, Utrecht University, 1992.

[16] J. Erickson and R. Seidel. Better lower bounds on detecting affine and spherical degeneracies. In *Proceedings of the 34th Symposium on Foundations of Computer Science*, 1993. 528–536.

[17] J. D. Foley, A. Van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice.* Addison-Wesley, Reading, MA, 1990.

[18] L. Guibas, M. Overmars, and M. Sharir. Intersecting line segments, ray shooting, and other applications of geometric partitioning techniques. In *Proc. 1st Scand. Workshop Algorithm Theory*, volume 318 of *Lecture Notes in Computer Science*, pages 64–73. Springer-Verlag, 1988.

[19] D. Haussler and E. Welzel. ϵ nets and simplex range queries. *Discrete Comput. Geom.*, (2):127–151, 1987.

[20] R. Karp. On-line algorithms versus off-line algorithms: how much is it worth to know the future? Technical Report TR-92-044, International Computer Science Institute, 1992.

[21] H. G. Mairson and J. Stolfi. Reporting and counting intersections between two sets of line segments. In R. A. Earnshaw, editor, *Theoretical Foundations of Computer Graphics and CAD*, volume F40 of *NATO ASI*, pages 307–325. Springer-Verlag, Berlin, West Germany, 1988.

[22] F. M. Maley. A generic algorithm for one-dimensional homotopic compactation. *Algorithmica*, 6:103–128, 1991.

[23] J. Matoušek. Cutting hyperplane arrangements. *Discrete Comput. Geom.*, 6:385–406, 1991.

[24] J. Matoušek. Efficient partition trees. *Discrete & Computational Geometry*, 8:315–334, 1992.

[25] J. Matoušek. Range searching with efficient hierarchical cuttings. In *Proceedings of the 8th ACM Symposium on Computational Geometry*, pages 276–285, 1992.

[26] K. Mulmuley. A fast planar partition algorithm, I. In *Proc. 29th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 580–589, 1988.

[27] J. Pach and M. Sharir. On vertical visibility in arrangements of segments and the queue size in the Bentley-Ottman line sweeping algorithm. *SIAM J. Comput.*, 20:460–470, 1991.

[28] M. Pellegrini. On collision-free placements of simplices and the closest pair of lines in 3-space. *SIAM J. on Computing*, 23(1):133–153, 1994.

[29] M. Pellegrini. On Point Location and Motion Planning in Arrangements of Simplices. In *Proceedings of the 26th ACM Symposium on Theory of Computing*, pages 95–104, 1994.

[30] D.E. Willard. Polygon retrieval. *SIAM Journal of Computing*, pages 149–165, 1982.

[31] A. C. Yao and F. F. Yao. A general approach to D-dimensional geometric queries. In *Proc. 17th Annu. ACM Sympos. Theory Comput.*, pages 163–168, 1985.