INFORMATIQUE THÉORIQUE ET APPLICATIONS

JIŘÍ MATOUŠEK

Spanning trees with low crossing number

Informatique théorique et applications, tome 25, n° 2 (1991), p. 103-123.

<http://www.numdam.org/item?id=ITA_1991__25_2_103_0>

Informatique théorique et Applications/Theoretical Informatics and Applications
(vol. 25, n° 2, 1991, p. 103 à 123)

SPANNING TREES WITH LOW CROSSING NUMBER (*)

by Jiří MATOUŠEK (1)

Abstract. – *Let P be a point set in the plane and T a spanning tree on P, whose edges are realized by segments. We define the* crossing number *of T as the maximum number of edges of T intersected by a single line. We give a $\Lambda(n^{2.5})$ deterministic algorithm finding a spanning tree with crossing number $O(\sqrt{n})$ on a given n point set (this crossing number is asymptotically optimal), and a $\Lambda(n^{4/3})$ randomized (Las Vegas) algorithm finding a spanning tree with crossing number $O(\sqrt{n}\log n)$ (here $f(n)=\Lambda(g(n))$ means $f(n)=O(g(n)\log^c n)$ for a constant c). This improves results of Welzl and Edelsbrunner* et al.

We also consider the construction of a family of $O(\log n)$ spanning trees, such that for every line λ there is a tree in this family such that λ crosses only $O(\sqrt{n}.\log^2 n)$ of its edges. We obtain a $\Lambda(n)$ Monte Carlo algorithm for this problem, improving a result of Edelsbrunner et al. *This result has numerous consequences for the construction of further randomized algorithms, using the above problems as a subroutine.*

Résumé. – *Soit P un ensemble de points du plan et soit T un arbre recouvrant de P, dont les arêtes sont des segments. Le nombre de croisements de T est le nombre maximal d'arêtes de T intersectées par une même droite. Si f et g sont deux fonctions, on pose $f(n)=\Lambda(g(n))$ s'il existe une constante c telle que $f(n)=O(g(n)\log^c(n))$. Nous donnons un algorithme déterministe en $\Lambda(n^{2.5})$ pour construire un arbre recouvrant dont le nombre de croisement est $O(\sqrt{n})$, où n est le nombre de points (ce nombre de croisement est asymptotiquement optimal); on donne également un algorithme probabiliste en $\Lambda(n^{4/3})$ pour construire un arbre recouvrant dont le nombre de croisement est $O(\sqrt{n}\log n)$. Ceci améliore des résultats de Welzl, Edelsbrunner* et al.

On considère également la construction d'une famille de $O(\log n)$ arbres recouvrants tels que, pour chaque droite λ il existe un arbre de la famille tel que λ intersecte seulement $O(\sqrt{n}\log^2 n)$ arêtes de l'arbre. On obtient un algorithme probabiliste en $\Lambda(n)$ pour ce problème, ce qui améliore un résultat de Edelsbrunner et al. *Ce résultat a de nombreuses conséquences pour la construction d'autres algorithmes probabilistes, qui utilisent alors les solutions des problèmes ci-dessus comme sous-programmes.*

1. INTRODUCTION

It will be convenient to adopt the "Λ-notation" of [8], which is similar to the usual "big-O" notation: we write $f(n)=\Lambda(g(n))$, which means that

(*) Received January 1989, revised April 1990.
(1) Department of Computer Science, Charles University Malostranské nám. 25, 11800 Praha 1, Czechoslovakia.

Informatique théorique et Applications/Theoretical Informatics and Applications
0988-3754/91/02 103 21/$4.10/

$f(n) = O(g(n)\log^c n)$ for a constant c. For the sake of simplicity, we shall not pay the price of more complicated methods for an optimization of the polylogarithmic factors.

Let P be a set of n points in the plane, let T be a spanning tree with vertex set P and let its edges be realized by straight segments (we also call this realization a spanning tree on P). For a line λ, we define the *crossing number of T relative to λ* as the number of edges of T intersected by λ, and we define the *crossing number* (some authors use *stabling number*) of T as the maximum of crossing numbers of T relative to λ over all lines λ.

We consider the following problem: Given a set P of n points in the plane, construct a spanning tree T on P with crossing number as small as possible.

In general one cannot have a better crossing number than $\Omega(\sqrt{n})$ [14]. The example is simple: Consider a set L of $\sqrt{(2n)}$ lines in general position, and choose one point in every region of the arrangement of L, yielding a (roughly) n point set P. Then every edge of the $n-1$ edges of any spanning tree on P must cross at least one of the lines of L, and thus the average (and also maximum) number of intersections per a line of L is of order $\Omega(\sqrt{n})$.

Welzl [14] proved that for every n-point set in the plane there exists a spanning tree with crossing number $O(\sqrt{(n\log n)})$, and gave a polynomial algorithm (not very efficient) finding such a spanning tree (in fact, his result applies to a more general situation).

The primary aim of this paper was to improve this algorithm. During the preparation of the paper the author obtained a version of the paper of Edelsbrunner *et al.* [8] (by the kindness of Emo Welzl), which contains some quite similar ideas, and their results were used to simplify some of the proofs. The paper [8] gave an $O(n^3 \log n)$ deterministic algorithm constructing a single spanning tree with crossing number $O(\sqrt{n}.\log n)$. Since the construction of a *single* "good" spanning tree seems to be difficult, they suggested to construct a small *family* of spanning trees on a given point set, such that for every line there is a "good" spanning tree in this family. Such a family can replace a single tree in many applications.

If F is a family of spanning trees, let the *crossing number of F relative to a line* λ be the minimum of crossing numbers of $T \in F$ relative to λ, and let the *crossing number* of F be defined similarly as for a single tree. [8] gave a $\Lambda(n^{3/2})$ Monte Carlo algorithm (*see* section 2 for the explanation of various notions of randomized algorithms), which finds a family of $O(\log n)$ spanning trees with crossing number $O(\sqrt{n}.\log^2 n)$. Our techniques used for the single spanning tree can also be applied to an improvement of their results.

The version of this paper originally submitted for publication contained the following results: A single spanning path with crossing number $O(\sqrt{n}\log n)$ can be found by a deterministic algorithm in time $\Lambda(n^{2.5})$, or by a randomized (Las Vegas) algorithm in time $O(n^{1.4+\delta})$ for every $\delta > 0$. A family of $O(\log n)$ spanning paths on P with crossing number $O(\sqrt{n}.\log^2 n)$ can be computed by a deterministic algorithm in time $\Lambda(n^{1.75})$, or by a randomized (Las Vegas) algorithm in time $\Lambda(n^{4/3+\delta})$ for every $\delta > 0$, or by a Monte-Carlo randomized algorithm in time $\Lambda(n)$ with high probability.

Later on, significant new developments have occurred in the work on the problems discussed here. Chazelle and Welzl [5] have shown that even the crossing number $O(\sqrt{n})$ can be always attained. Applying their results, our deterministic algorithm for the construction of a single spanning tree could easily be modified to yield the following:

1.1. THEOREM: *Given a set P of n points in the plane, a spanning path on P with crossing number $O(\sqrt{n})$ can be found by a deterministic algorithm in time $\Lambda(n^{2.5})$.*

Applying results of Agarwal [1, 2], our randomized algorithm for the construction of a single spanning tree could be simplified (or, at least, its description) and made slightly more effective:

1.2. THEOREM: *Given a set P of n points in the plane, a spanning path on P with crossing number $O(\sqrt{n}\log n)$ can be found by a randomized (Las Vegas) algorithm in expected time $\Lambda(n^{4/3})$.*

Agarwal [1] proved (applying his deterministic plane partitioning algorithm and some of the techniques of the present paper) that a family of $O(\log n)$ spanning trees with crossing number $O(\sqrt{n})$ can be found deterministically in time $\Lambda(n^{3/2})$, and this is the best known result in this direction so far.

As for the Monte Carlo algorithm for a family of spanning trees, we show the following:

1.3. THEOREM: *Given a set P of n points in the plane, a family of $O(\log n)$ spanning paths on P can be computed by a Monte-Carlo randomized algorithm in expected time $\Lambda(n)$, so that the crossing number of this family is $O(\sqrt{n}.\log^2 n)$ with high probability.*

If we use this algorithm as a subroutine in applications, the crossing number of the family usually will not affect the correctness of the output of the application, but only its running time. This means that as a whole, we get a Las Vegas algorithm – the usual type of a randomized algorithm in computational geometry.

Spanning trees with low crossing number have by now a wide range of applications in computational geometry. The first ones were given in [14], and a very impressive list is in [2]; *see* also [5, 8]. We shall not list all the applications here, but we shall try to give the reader an idea why such spanning trees are useful.

The basic application is the construction of efficient algorithms for answering geometric range queries. Data structures with fast query answering are one of the main tools in computational geometry, as many problems can be reduced to them (as the present paper also illustrates). Let us consider e. g. a *halfplanar range counting problem* as the basic case. This is the following algorithmic problem: Given a set P of n points in the plane, preprocess it so that given a query halfplane h, the number of points of P lying inside h can be determined quickly.

Suppose now that we have a spanning *path* on P with crossing number $c = O(\sqrt{n})$. Given a query halfplane h, its boundary line cuts the path into at most $c+1$ pieces, each of them lying entirely inside h or entirely outside h. With a suitable (and not quite simple) data structure (*see* [8] or [5] for such structures) the points of intersection can be determined quickly. Now if the vertices of the path are numbered along the path, it is easy to count the number of vertices in every piece in time $O(c)$ in total. This yields a data structure for the halfplanar range counting problem, which uses only $\Lambda(n)$ storage and answers a query in time $\Lambda(\sqrt{n})$. As the results of Chazelle [4] on lower bounds for query answering algorithms indicate, this is probably nearly optimal (up to logarithmic factors) solution to this problem. This method can be also generalized for triangular ranges, etc. A small familly of spanning paths can be used similarly in most of the applications. E. g., combining Theorem 1.3 with the algorithm described in [1], we get the following:

1.4. COROLLARY: *Given a set P of n points in the plane and a parameter m ($n \log^2 n \leqq m \leqq n^2$), we can compute a data structure of size $O(m)$ for counting the number of points of P in a query triangle, by a Monte Carlo algorithm in expected time $\Lambda(m)$. With high probability, the worst-case query time for this data structure is $\Lambda(n/\sqrt{m})$.*

The plan of the paper is the following: section 2 gives some background material we shall use. Section 3 presents some results (mainly from the literature) on data structures for answering geometric queries. Section 4 describes how to select a small test set of lines, which suffice for verification that a spanning tree on given point set has a small crossing number. Section 5

describes a modification of the algorithm of Welzl [14] (formulated for a general range space), and section 6 gives the proofs of Theorems 1.1-1.3.

2. PRELIMINARIES

In addition to usual deterministic algorithms (for which we measure the worst-case complexity) we shall consider two types of randomized algorithms. These algorithms use a random number generator in their computation. A *Las Vegas algorithm* of complexity $f(n)$ computes a correct answer for each input, and the expected time for obtaining an answer for input of size n is at most $f(n)$. A *Monte Carlo algorithm* of complexity $f(n)$ and failure probability p ($p < 1$) computes an answer for every input of size n in expected time $\leqq f(n)$. The answer be wrong, but with probability at most p.

When we say that something happens *with high probability* during a computation of some randomized algorithm, we mean that this probability can be made n^{-c} (n the size of input) for every fixed $c > 0$, by an appropriate choice of constant factors appearing in the algorithm. In this sense, all our Monte Carlo algorithms give correct results with high probability. With some additional effort we can implement them so that the execution time does not exceed the claimed bound with high probability.

Note that if we have a Las Vegas algorithm of complexity $f(n)$ for verifying the correctness of an answer obtained by the Monte Carlo algorithm of the same complexity, we can solve the original problem by a Las Vegas algorithm of complexity $f(n)$.

Let us introduce some geometric terminology and notation.

A finite set L of lines in the plane determines a cell complex in the plane, called the *arrangement* of L. The 2-dimensional faces of the arrangement are called *regions;* their sides – the 1-dimensional faces – are called *edges* of the arrangement.

We shall use the line-point *duality transform.* This is a transform D, which maps points to lines and nonvertical lines to points, and its main property is that it preserves the relation "lying above" for pairs point-line or line-point (*see* e. g. [6] for more information).

For the sake of simplicity we shall assume that all the arrangements and point configurations we deal with are non-degenerate. The results hold also for the general case, as one can show by a perturbation argument (*simulation of simplicity*, see [6]).

If λ is a nonvertical line, then λ^+ will denote the upper closed halfplane determined by λ.

A probabilistic approach to computational geometry problems turned out to be very fruitful (let us quote the pioneering works of Clarkson – e. g. [3] – and Haussler and Welzl [10]). Important notions here are range spaces and ε-nets.

A *range space* S is a pair (X, R), where X is a set and R is a set of subsets of X. Members of R are called *ranges* of S. S is *finite* if X is finite. The basic combinatorial characteristics of a range space is its dimension, a concept introduced in [13]. The *Vapnik-Chervonenkis dimension* (or simply *dimension*, [10]) of S is the largest integer d for which there exists a d-element set $A \subset X$ such that the set $\{A \cap r;\ r \in R\}$ consists of all subsets of A. If no such maximal d exists, we say that the dimension of S is infinite.

Let $S = (X, R)$ be a finite range space, ε a nonnegative real number and N a subset of X. We say that N is an ε-net *for* S, if N intersects each range $r \in R$ such that $|r| > \varepsilon |X|$.

A basic result about ε-nets (obtained by a counting argument) is the following:

2.1. THEOREM [10]: *Let S be a finite range space of dimension d and let δ, $\varepsilon \in (0, 1)$ be real numbers. Then a sample N of at least* $\max[4\varepsilon^{-1}\log(2/\delta), 8d\varepsilon^{-1}\log(8d/\varepsilon)]$ *points, drawn independently from X (with uniform probability distribution), is an ε-net for S with probability at least* $1-\delta$. ■

This theorem holds also if the points of S are considered as a multiset, *i. e.* a point may have a multiple occurrence (in this case we count the cardinality with the multiplicities).

3. QUERIES ON SETS OF POINTS, LINES AND SEGMENTS

Our algorithms will use some data structures, which allow to answer range counting queries efficiently; we shall need various tradeoffs between preprocessing and query time. By now, most of the data structures we need have appeared in the literature, but we shall briefly mention the underlying principles here (at least for the simpler results).

First we mention a data structure (by now classical) for halfplanar range counting queries with fast query answering and rather long preprocessing. The problem "count how many points of P lie above given line λ" is (by

duality) equivalent to the problem "count how many lines of $L = D(P)$ lies above the point $D(\lambda)$". The answer remains constant within the regions of the arrangement of L, hence we may associate these counts with the regions. Then, given a query point, it suffices to determine in which region it lies.

A suitable representation of the arrangement of a set of n lines can be constructed in time $O(n^2)$ [9]. Applying an optimal algorithm for point location in planar subdivisions (e. g. that of Kirkpatrick [11]), the arrangement can be further preprocessed [in time $O(n^2)$], so that given a query point, the region of the arrangement containing it can be located in time $O(\log n)$.

A slightly more complex type of queries, which we shall need in the proof of 1.3, is described in the following lemma (which appears e. g. in [8]):

3.1. LEMMA: *Given a set S of n segments, we may preprocess it in time $O(n^2)$, so that then given a query line, we can compute the number of segment of S intersecting it in time $O(\log n)$.*

In dual form, the problem reads as follows: given a set W of n double wedges, count for a given point the number of double wedges of W in which it lies. We may consider the arrangement of the boundary lines of the elements of W, and associate with every region the answer for points in that region; then again the point location is used. ■

The halfplanar range counting is a basic problem, but sometimes we need to handle more complicated ranges (wedges, triangles). A standard tool which allows us to pass to these ranges are the *range trees* (*see* e. g. [12] for more information and examples of application of this idea). Let $\{x_1, x_2, \ldots, x_n\}$ be a linearly ordered set ($x_1 \leqq x_2 \leqq \ldots \leqq x_n$). Let us imagine a rooted binary tree T of height $O(\log n)$ with leaves $x_1, \ldots, x_n$, such that the order of leaves from left to right is just the order of the x_i's. *A canonical range* is a set of all leaves of the subtree of T rooted at some vertex of T (thus canonical ranges form intervals in our ordered set). The sum of sizes of all canonical ranges is $O(n \log n)$ and every contiguous interval in the set $\{x_1, \ldots, x_n\}$ can be partitioned into $O(\log n)$ canonical ranges.

Given a point set P, we may sort the points by the x-coordinate and make a range tree on it. Then we preprocess every canonical range for halfplanar range counting, and this data structure allows us to answer a counting query for a range given as the intersection of a halfplane with a vertical strip. Both the preprocessing and the query time increase at most by a logarithmic factor compared to the original algorithm for halfplanar range counting. Now the complement of every triangle can be partitioned in at most five ranges of the above form, which allows to handle range counting queries for triangles.

The following useful result of Agarwal [1] speaks about a fast deterministic processing of a batch of many queries:

3.2. THEOREM [1] (Red-blue intersection counting): *Given a set of n blue segments and a set of m red segments, we can count for every red segment the number of its intersections with the blue segments, deterministically and in time $\Lambda((n+m)^{4/3})$.* ■

Let us remark that the time can be made $\Lambda(m+n+m^{2/3}n^{2/3})$ by a slight modification of the methods of [1].

3.3. COROLLARY: *Given a spanning tree T on a n point set and a set L of m lines, we can compute the crossing number of T relative to every line of L in total time $\Lambda((m+n)^{4/3})$.* ■

A very sophisticated data structures for arrangements of lines are given in [8]. Among many applications of their technique mentioned there, we select one tailored to our purpose. We define a *canonical triangulation* for an arrangement of lines. This triangulation is obtained as follows: we pick the leftmost vertex of every region and we connect it to all other vertices of the region (except for its neighbors). The results of [8] immediately give the following:

3.4. LEMMA [8]: *Given a set L of n lines, one can preprocess it by a Monte Carlo algorithm in expected time $\Lambda(n^{3/2})$, yielding with high probability a data structure, which allows to answer in time $\Lambda(n^{1/2})$ queries of the following form: Given a query point p, return the triangle Δ of the canonical triangulation of the arrangement L in which p lies.* ■

In fact, Theorem 1.3 allows to reduce the preprocessing time to $\Lambda(n)$.

A powerful range counting structure is that of Corollary 1.4. This is proved from Theorem 1.3. Actually if one takes the algorithm of Agarwal [1] for triangle range counting and replaces the deterministic construction of a family of spanning trees with low crossing number [in time $\Lambda(n^{3/2})$] by the Monte Carlo construction from Theorem 1.3 [running in expected time $\Lambda(n)$], one immediately obtains the claim of Corollary 1.4. We shall not repeat the proof here.

We shall apply 1.4 in the proof of Theorem 1.2 (but we avoid its use in the proof of 1.3). We apply it via two lemmas, which are already tailored to our application:

3.5. LEMMA: *A set P of n points in the plane can be preprocessed by a $\Lambda(n^{4/3})$ Monte Carlo algorithm, yielding with high probability a data structure*

which allows to draw a random point from the set $P \cap \Delta$, *where* Δ *is a query triangle, in time* $\Lambda(n^{1/3})$ (*provided that this set is nonempty*).

Proof: We order the points of P arbitrarily and build a range tree on it. For a node p of the range tree, we denote by $set(p)$ the canonical range associated with p.

We preprocess every canonical range for triangle range counting queries. We choose $m = n^{4/3}$; then 1.4 says that we can do the preprocessing in total time $\Lambda(n^{4/3})$ and a triangle range counting query on each canonical range can be answered in time $\Lambda(n^{1/3})$. Now given a query triangle Δ, a random point of $P \cap \Delta$ is generated by the following procedure (we suppose that $\Delta \cap P$ is nonempty):

1. Set p to the root of the range tree.

2. If p is a leaf, return it as the answer.

3. Otherwise let q, q' be the sons of p. Count the numbers $n = |\Delta \cap set(q)|$, $n' = |\Delta \cap set(q')|$. Randomly choose q or q' and set p to the chosen node, the probability of choice of q being $n/(n+n')$ and the probability of choice of q' being $n'/(n+n')$. Continue by step 2.

Since the tree has height $O(\log n)$, we obtain the desired point after $O(\log n)$ repetitions of steps 2-3. ■

3.6. LEMMA: *Given a set* S *of* n *segments, we may preprocess it by a Monte Carlo algorithm in expected time* $\Lambda(n^{4/3})$, *yielding with high probability a data structure, which allows to count for a given line* λ *the number of segments of* S *intersected by it in time* $\Lambda(n^{1/3})$.

Proof: It is sufficient to reduce one query of the above form to $\Lambda(1)$ halfplanar range counting queries; then we apply 1.4 with $m = n^{4/3}$.

When we express every segment of our set as the difference of two semilines, a counting query on the set of segments stabbed by a query line can be transformed to line stabbing queries on two sets of semilines.

Let us consider a query on a set of semilines by a line λ. We may divide the semilines into two groups according to their directions relative to λ: In one group, a semiline meets λ iff its endpoint lies below λ, while in the other group the opposite is true. This partitioning of the semilines depends on λ, but the directions of semilines within each group form contiguous intervals. We build a range tree on the set of semilines ordered by their directions and we preprocess the set of endpoints for semilines in each canonical range for halfplanar range counting. This reduces the problem to the halfplanar range counting. ■

Finally we shall need also another form of queries – reporting of lines intersecting a query segment. The dual form of this problem is to report points of a given set, lying inside a given query double wedge. This phase will not be critical in our application, so we can afford to use a suboptimal (and simple) solution, e. g. that of Edelsbrunner and Welzl [7]:

3.7. LEMMA [7]: *Given a set P of n points, we may preprocess it in time $O(n \log n)$, so that given a query double wedge w, all the points of P inside w can be reported in time $O(n^{0.695} + |w \cap P|)$.* ■

4. SELECTING TEST LINES

In this section we observe that if the spanning tree T has bounded degrees, only $O(n)$ lines suffice for testing its crossing number. We shall show this via a plane partitioning lemma (which has many other important applications in computational geometry).

4.1. LEMMA (partitioning lemma): *Given a set L of n lines in the plane and a number $r \leqq n$,*

(a) [1] *we can partition the plane into at most r^2 triangles in such a way that the interior of every triangle is intersected by at most $O(n/r)$ lines. This partitioning can be performed by a deterministic algorithm in time $\Lambda(nr)$.*

(b) we can partition the plane into at most r^2 triangles in such a way that the interior of every triangle is intersected by at most $O(n . \log r/r)$ lines. This partitioning can be performed by a Monte Carlo algorithm in expected time $\Lambda(n + r^2)$ [obtaining the above property of the partitioning with high probability, the probability of success depending on the constant in the $O(n . \log r/r)$ bound], or by a Las Vegas algorithm in expected time $\Lambda((n + r^2)^{4/3})$.

Proof of (b): As it was observed independently by several authors, if we pick a random sample R of r lines of L and triangulate the regions of the arrangement of R, then this triangulation will have the desired property with high probability. This can be easily proved using ε-net theory (in general form this appears in [14], Lemma 4.1):

Consider the range space U, where the role of points is played by the lines of L and each range is the collection of lines intersected by a given segment. This range space has a bounded dimension, as demonstrated in [14]. Thus by 2.1, R is an ε-net for U with high probability, where $\varepsilon = O(\log r/r)$, which implies that any segment which does not intersect any line of R has is intersected by at most εn lines of L. Now if the interior of a triangle from a

triangulation of the arrangement of R were intersected by more than $3\varepsilon n$ lines of L, we could find a segment in its interior intersected by more than εn lines.

Thus the Monte Carlo algorithm consists of the selection of random sample R of size r from L, constructing its arrangement and triangulating its regions. To get a Las Vegas algorithm from this, it suffices to use the red-blue intersection algorithm (Lemma 3.2) for verification that none of the $O(r^2)$ sides of the triangles is intersected by too many lines. ■

Now we can state the main result of this section:

4.2. LEMMA:

(a) Given a set S of n points, one can find a set L of $O(n)$ lines, such that if a spanning path T on S has crossing number $\leqq s$ relative to every line of L, then it has crossing number $\leqq s+O(\sqrt{n})$ relative to all lines. Such a set can be found by a deterministic algorithm in time $\Lambda(n^{3/2})$.

(b) Given a set S of n points, one can find a set L of $O(n)$ lines, such that if a spanning path T on S has crossing number $\leqq s$ relative to every line of L, then it has crossing number $\leqq s+O(\sqrt{n}\log n)$ relative to all lines. Such a set can be found by a Las Vegas algorithm in time $\Lambda(n^{4/3})$ or by a Monte Carlo algorithm in expected time $O(n)$ with high probability.

Proof: Consider the dual line arrangement $D(S)$. Use the partitioning lemma 4.1 with $r=\sqrt{n}$ to partition the plane into $O(n)$ triangles, intersected by $O(\sqrt{n})$ lines each [resp. by $O(\sqrt{n}.\log n)$ in the randomized version], and choose one interior point in each of the triangles, obtaining a point set P. Now for every point in the plane, there exists a point in P lying in the same triangle, and thus separated from the given point only by $O(\sqrt{n})$ lines of $D(S)$ [resp. by $O(\sqrt{n}.\log n)$ lines]. In the dual plane, this means that for every line λ there exists a line ν in $D(P)$, such that one of the double wedges determined by λ and ν contains only $O(\sqrt{n})$ points of S [resp. $O(\sqrt{n}\log n)$ points]. Every edge of T crossed by λ and not crossed by ν must have one endpoint in the "small" double wedge. Since T is a spanning path, it has bounded degrees of vertices (by 2), and so the crossing number of λ exceeds the crossing number of ν by at most $O(\sqrt{n})$ [resp. $O(\sqrt{n}\log n)$]. Hence we may take $L=D(P)$.

The time bounds follow from 4.1. ■

5. GENERAL ALGORITHM FOR A SINGLE TREE

We shall give an algorithm finding a spanning tree on given point set with a low crossing number relative to a given set of lines. We shall formulate it for a general range space of finite dimension, slightly refining the algorithm of Welzl [14]. For a general range space, the crossing number of a spanning tree is defined analogously to the special planar case; an edge $\{x, y\}$ crosses a range r if exactly one of x, y belongs to r.

The points where our algorithm differs from [14] are that we construct a spanning path (which is easy) and that we group the selection of edges into stages, and we update the weights only between the stages.

Let $S=(X, R)$ be a range space, and m a number. Let $\pi^*(m)$ denote the maximum number of equivalence classes, into which the ranges are divided by any set of m points of S (two ranges r_1 and r_2 are equivalent relative to a set $Y \subset X$ if $r_1 \cap Y = r_2 \cap Y$. The function $\pi^*(m)$ is called the *dual shatter function* of S. It is not difficult to show (*see* [14]) that a range space of finite dimension has a polynomially bounded dual shatter function.

Let us assume that S is a range space with dual shatter function of order $O(m^d)$ for some $d > 1$. In such a range space, the following holds:

5.1. LEMMA [14]: *Let P be a set of n points in S and let Q be a multiset of ranges in S. Then there exists a pair of points x, y in P, such that $\{x, y\}$ is not crossed by more than $O(|Q| \log n/n^{1/d})$ ranges of Q.* ■

This lemma is easily proved by an ε-nets argument. It is a basis for the algorithm finding a spanning tree with low crossing number in [14]. Our (modified) algorithm will be the following:

5.2. Algorithm:

Input: A set $P \subset X$ of n points of the range space S and a set $Q \subset R$ of ranges (we may assume that they are pairwise nonequivalent and thus the size of Q is polynomial in n).

Output: A spanning path T on the set P.

Method: Let us put $t = \lfloor n^{1-1/d} \rfloor$. We shall construct T in about $n^{1/d}$ stages, adding t edges at every stage (except for the last few ones). Let us denote by $e_1, e_2, \ldots, e_{n-1}$ the edges of T in the order as they are added. Let T_j denote the graph with vertex set P and edges $e_1, \ldots, e_j$; it will be a disjoint union of paths. We set $Q_1 = Q$.

At the beginning of stage i we have a multiset Q_i of ranges of cardinality k_i (the ranges are just those of Q, but with multiple occurrences) and the edges $e_1, \ldots, e_{(i-1)t}$ have been selected. We choose one endpoint in each component of $T_{(i-1)t}$, forming a set P_i, $|P_i| = n_i = n-(i-1)t$. We choose t vertex disjoint edges (partial matching) on P_i, such that each edge crosses at most $w_i = C.k_i.\log n_i/n_i^{1/d}$ ranges of Q_i. Here C is a suitable positive constant (whose value affects the multiplicative factor in the bound on the crossing number, stated in Lemma 5.3 below). The existence of such edges can be deduced from Lemma 5.1 (provided that C is larger than some number C_0, depending only on the range space in question). In this general algorithm we do not specify the method how these edges are found.

It remains to form Q_{i+1}, *i.e.* specify the multiplicity of every range of Q in Q_{i+1}. If a range r crosses p edges among $e_1, e_2, \ldots, e_{it}$, we may choose the multiplicity for it as any number between $2^{\lfloor p/t \rfloor}$ and $2^{\lfloor p/t \rfloor + 1}$ (this gives us the freedom to compute p only with accuracy t).

It will be convenient to handle last few ($O(1)$) stages separately (the reason is e.g. that when $|P_i|$ decreases below $2t$, we cannot find a matching of sufficient size on it). Namely instead of last stages we finish the algorithm by joining the pieces of the path constructed so far into a spanning path in an arbitrary way. This increases the number of crossings for an arbitrary range with the edges of the path by at most $O(t)$, which does not affect the desired result:

5.3. LEMMA: *The spanning path T constructed by algorithm 5.2 crosses every range of Q at most $O(n^{1-1/d}\log n)$ times.*

Proof: The proof closely follows the one of Theorem 4.2 in [14]. Let us define the *weight* for a range $r \in Q$ relative to edges $e_1, \ldots, e_j$ as $2^{p/t}$, where p is the number of edges among $e_1, \ldots, e_j$ crossing r. The multiplicity of a range r in Q_i (at the beginning of stage i) as used in the algorithm approximates the weight of this range relative to the edges selected before stage i. The weights will only be needed for the proof of correctness, while in the computation we shall suffice with the multiplicities.

Let f_i denote the sum of weights of all ranges of Q at the beginning of stage i; it is easily seen that $k_i/2 \leq f_i \leq 2k_i$.

If there is a range with crossing number s in the end of the algorithm, its weight relative to $e_1, \ldots, e_{n-1}$ is equal to $2^{s/t}$. As we will show later, the sum of weights of all ranges relative to $e_1, \ldots, e_{n-1}$ will be polynomial in n. This implies that s must be of order $O(t\log n) = O(n^{1-1/d}\log n)$, which is what we want.

Let us consider the increase of the sum of weights for all ranges, caused by adding the edge e_j at stage i ($\lfloor j/t \rfloor = i$). First we observe that during stage i the weight of every individual range might have increased at most twice. The choice of edge e_j in the algorithm guarantees that it crosses ranges with sum of weights at most $4w_i$. For each range crossed by e_j, its weight is multiplied by the factor $2^{1/t}$, hence the sum of weights increased by at most $(2^{1/t}-1).4w_i$. From this we get that $f_{i+1} \leqq f_i(1+(2^{1/t}-1).4C.\log n_i/n_i^{1/d})^t$. We also have $f_1 = |Q| = n^{0(1)}$. By suitable asymptotic estimations, one can compute from the above recurrence that the sum of weights at the end of the last stage is polynomial in n, which proves the lemma. ■

Let us remark that while in the original algorithm of Welzl [14] the weights can be exponentially large numbers and some problems with the bit complexity of the algorithm might appear (although these can be overcome by representing the weights with a limited precision only), in our algorithm the multiplicities are polynomially large integers.

6. PROOFS OF THEOREMS 1.1-1.3

Let the set P of n points be given. All the algorithms start by selecting the set L of $O(n)$ "test" lines as in lemma 4.2, using the appropriate algorithm version. We shall assume that the lines of L contain no point of P; this can be assured by a slight perturbation.

A single spanning path in Theorem 1.2 will be constructed by algorithm 5.2. The range space S in question will be $(E^2, \{\lambda^+, \lambda \text{ a nonvertical line}\})$ [thus $d=2$ in algorithm 5.2, since a set of m points partitions the ranges into $O(m^2)$ equivalence classes]. For Theorem 1.1, we shall need a few modifications of algorithm 5.2 to get the "tight" crossing number $O(\sqrt{n})$.

The set Q of ranges in the input of 5.2 will be in our case the set $\{\lambda^+; \lambda \in L\}$. Often we shall not distinguish between a set of ranges (which are the upper closed halfplanes) and the underlying set of boundary lines.

Two things must be specified to make algorithm 5.2 work:

(A) How the $t = \lfloor n^{1/2} \rfloor$ new short (relative to Q_i) edges at each stage are chosen.

(B) How the multiplicities of ranges in Q_{i+1} are computed, which amounts to specifying how the number of intersections of each line of L with the already selected edges is counted (with accuracy $\sqrt{n}$ only).

6.1. Monte Carlo construction of a single tree

In order to prove Theorem 1.2 (Las Vegas construction of a single spanning path in time $\Lambda(n^{4/3})$), it suffices to give a Monte Carlo algorithm; then we may use 3.3 for verification of the crossing number.

We start by the implementation of (A) (the selection of new edges at each stage). We draw a random sample N of size $\sqrt{(n_i/2)}$ from the multiset Q_i and we triangulate the arrangement of N, yielding at most $n_i/2$ triangles. This induces an equivalence relation on P_i with $\leq n_i/2$ classes (the equivalent points lie in the same triangle). We may then choose the desired $\sqrt{n}$ edges of a partial matching inside the equivalence classes (we have $\sqrt{n} < n_i/4$, since the last stages are treated separately in algorithm 5.2). This guarantees that the edges of the matching intersect at most $|Q_i| \log n/\sqrt{n}$ lines of Q_i (with high probability) – we may use the same ε-net argument as in the proof of 4.1 *b*).

If we know the multiplicities of the lines in Q_i, we can select the random sample N in time $O(\sqrt{n} \log n)$ (e. g., storing the multiplicities suitably in a balanced tree).

We could now handle (A) straightforwardly as follows: We construct the arrangement of N, triangulate the faces, locate all points of P_i in this planar subdivision, construct the equivalence classes on P_i and select the edges of the matching. This takes time $O(n \log n)$ per stage, and hence $\Lambda(n^{1.5})$ in the whole algorithm.

To do better, we must avoid both the construction of the arrangement of N, and the construction of the whole equivalence on P_i.

The following procedure brings a slight (theoretical) improvement. We take the canonical triangulation of the arrangement of N, mentioned in 3.4. The procedure finding the new edges in (A) then goes as follows:

1. Set M to an empty matching.
2. Choose a random point X of P_i.
3. Construct the triangle Δ (of the canonical triangulation) in which X lies.
4. Choose a random point Y among the points of P_i lying in Δ.
5. If the edge XY is vertex disjoint with the matching M, add it to M.
6. If M contains sufficiently many ($\sqrt{n}$) edges, finish, else continue by step 2.

If the points X and Y are chosen with uniform distribution, the expected number of repetitions of steps 2-6 is $O(\sqrt{n})$ (since a sufficiently large

matching exists). Let us discuss the implementation of the above steps; nontrivial are the steps 3 and 4.

Step 3 is covered by 3.4; both the expected preprocessing time and the time for $O(\sqrt{n})$ queries at one stage will be $\Lambda(n^{3/4})$, and the expected total time spent on this in all stages $\Lambda(n^{1.25})$ (this could be further improved by the results of this paper, but this step is not the bottleneck of our algorithm).

To facilitate the execution of step 4, we preprocess the set P_1 for the generation of random points in query triangles as in Lemma 3.5. We shall not repeat the preprocessing at each stage (for each P_i), but we only preprocess the set of all endpoints of the components of the (partial) spanning path whenever their number decreases twice. In this way, P_i will be only a part of the preprocessed set, but at least 1/4, so in about 1/4 cases the point Y generated at step 4 will belong to P_i. This increases the expected running time only by a constant factor. During the whole algorithm, there will be $O(n)$ queries on the random point generating structure, and so by Lemma 3.5 the total expected time for both the preprocessing and the queries is $\Lambda(n^{4/3})$, which dominates the execution time.

Let us turn to the implementation of (B) – recomputing the multiplicities of ranges in Q_{i+1}. Here, again, a straightforward strategy is at hand, namely to use some variant of a red-blue intersection counting algorithm. This yields total $\Lambda(n^{1.5})$ expected time with the best known red-blue intersections counting.

To improve this bound, we will not test all lines at each stage, but select those for which the number of intersections is likely to have grown considerably. These lines should reveal themselves by intersecting a small sample of the newly added edges.

We shall dynamize the range counting structure from Lemma 3.6 (line stabbing on segments), so that it can accommodate insertions of segments (not deletions). This is done in a standard way: We have at most one static structure of size 2^i for each $i=0, 1, \ldots, \lfloor \log n \rfloor$, an insertion is handled by re-building some of the static structures according to the pattern of "counting in binary". A query on the whole current set is decomposed into at most $\log n$ queries on the current static structures. The query time remains of the same order as for the static structure, and the amortized time for n insertions is of the same order as for building the whole structure afresh. Into this dynamic structure we shall insert the newly added edges after each stage.

Now at each stage, the new multiplicities are computed as follows: we choose a random sample R of $c.\log n$ of the newly added edges of the spanning path (c a sufficiently large constant). We detect all lines of L

intersecting a segment from R, and for these lines we recompute the number of their intersections with all the edges selected so far (and stored in the dynamic structure). For the remaining lines we use the old number of intersections unchanged (since the last counting query for this line).

The lines of L intersecting edges of R are detected using a suitable data structure for reporting lines intersecting a query segment. Here lemma 3.7 is more than sufficient; we have $\Lambda(\sqrt{n})$ queries in total and $O(n^{0.695})$ time per query (plus the time proportional to the number of intersections, whose number will be discussed later); this gives less than $O(n^{4/3})$ in total.

Let us estimate the running time for the recomputing of the number of intersections for "suspicious" lines (those intersecting the random sample R of edges at some stage). Let t_i be the number of intersections of lines of L with the edges added at i-th stage. We know that the sum of t_i's is $O(n^{3/2}.\log n)$ (because of the crossing number of the resulting spanning path). At the same time, the expected number of intersections of the random sample R with L at stage i is $t_i|R|/\sqrt{n}$, thus the expected total number of intersections of random samples with lines of L is $\Lambda(n)$. Each such intersection is responsible for one line stabbing query on the data structure storing the segments, and from this we get that also the expected total running time for these line stabbing queries is $\Lambda(n^{4/3})$.

It remains to show that the above mechanism of counting multiplicities gives (with high probability) the correct multiplicities. Suppose that there is a line $\lambda \in L$ and stages i, $i+m$, such that the multiplicity of λ was recomputed the last time at stage i, and the number of intersections of λ with the edges added at stages $i+1$, $i+2$, ..., $i+m$ exceeds $\sqrt{n}$. For a fixed λ, i, m, the probability of this event is equal to the probability that each of $cm.\log n$ elements randomly drawn from $m\sqrt{n}$ elements misses a subset of $\sqrt{n}$ elements. This probability is $(1-1/m)^{cm.\log n} \leqq n^{-c}$. The overall probability that such a situation occurs for some λ, i, m is thus at most $n^3.n^{-c}$, and hence such a situation does not occur with high probability (if c is large enough). This completes the Monte Carlo implementation of (B) and the proof of Theorem 1.2. ■

6.2. Deterministic construction of a single tree

For the special range space for which we want to construct a spanning tree with low crossing number, the following strengthening of 5.1 holds:

6.1. LEMMA [5]: *Let P be a set of n points in the plane and let Q be a multiset of lines. Then there exists a pair of points x, y in P, such that the segment $\{x, y\}$ is crossed by no more than $O(|Q|/\sqrt{n})$ lines.* ■

This allows us to achieve the crossing number $O(\sqrt{n})$ by algorithm 5.2, but we must change some of its parameters. Namely instead of $t=\sqrt{n}$, we put $t=\sqrt{n}/\log n$ (t was the number of edges selected per stage of the algorithm; hence the number of stages increases to $\sqrt{n}\log n$). In each stage, we will select edges that intersect $O(|Q_i|/\sqrt{|P_i|})$ lines of Q_i (whose existence is guaranteed by iterated application of the above lemma). With these modifications, we can prove in the same way as in 5.3 that the resulting spanning tree will have crossing number $O(\sqrt{n})$.

Now let us turn to the details of implementation of steps denoted as (A) and (B) in the beginning of section 6. We allow time $\Lambda(n^2)$ for each stage, yielding total time $\Lambda(n^{2.5})$. Then (B) is straightforward: we may explicitly test each new edge for intersections with all lines of L.

As for (A), at each stage we preprocess the arrangement of Q_i for the queries of the following type (distance computing): given two points X, Y, compute the number of lines of Q_i intersecting the segment XY (counting the lines with their multiplicities in Q_i). Using a range tree, we may reduce one such query to $O(\log n)$ queries of the form "what is the number of lines lying above a query point X", which can be handled by point location in time $O(\log n)$ per query.

With such a mechanism, we compute the number of intersections for every edge on P_i, and then we select the desired partial matching edge by edge, always taking the edge intersecting the minimum number of lines of Q_i (and not incident to the previously selected edges).

In this way, each stage takes time $\Lambda(n^2)$. This proves Theorem 1.1. ■

6.3. Monte Carlo construction of family of trees

Here we may use the algorithm of [8] almost without change, only that we have a smaller set of "test" lines [$|L|=O(n)$ instead of $|L|=O(n^2)$], and because of this we must build a spanning tree with bounded degrees, namely a spanning path (this is easy). The first change allows us to improve the only part of the algorithm of [8], which does not run in time $\Lambda(n)$.

We shall briefly describe the algorithm, referring to [8] for the details and proofs. We build spanning paths $T_1, T_2, \ldots, T_k$ ($k=O(\log n)$), such that for every $\lambda\in L$ there exists i such that λ crosses only $O(\sqrt{n}\log^2 n)$ edges of T_i.

Suppose that we can construct (with high probability) the paths $T_1, T_2, \ldots, T_{i-1}$ and a set $L_i\subset L$ of size $\leqq |L|.2^{-i}$, containing all the lines of L intersecting too many edges of each T_j, $j<i$—the "bad lines". We

describe how to construct T_i and L_{i+1} which have similar properties with high probability, in expected time $\Lambda(n)$.

We build T_i in $O(\log n)$ phases from matchings. On the beginning of phase j ($j=1, 2, \ldots$) we have a current point set $P_j (P_1 = P)$, $|P_j| \leqq n.2^{-j+1}$, and a current collection of paths on P (initially a discrete graph on P). We choose a random sample N of size $\sqrt{(|P_j|/2)}$ from L_i, construct its arrangement and triangulate it. As in the case of a weighted arrangement in section 6.2, we compute the equivalence induced on P_j by this triangulation, which has $\leqq |P_j|/2$ classes. Inside each equivalence class, we choose an inclusion-maximal matching on the points of P_j, and add these edges to the current collection of paths (joining some path together). In this way, we add at least $|P_j|/4$ edges. Finally we choose one endpoint of each component of the current path collection into P_{j+1}. The number of components was halved by this, so $|P_{j+1}| \leqq |P_j|/2$. Hence in $O(\log n)$ phases we get a connected path T_i (similarly as in algorithm 5.2, we handle the "singular case" on the end separately).

Having finished T_i, we must detect the new "bad lines" to get the set L_{i+1}. We select a random sample R of $r = c_2 \sqrt{n}/\log n$ edges of T_i. This size is chosen so that a "bad line" is very likely to intersect some segment of R, while at least half of all lines of L_i is likely to miss R completely (to this end, the constant c_2 must be chosen suitably – *see* [8]). We preprocess the edges of R for line stabbing queries (as in Lemma 3.1) and form L_{i+1} as the set of lines intersecting at least one segment of R (here, for simplicity, our strategy differs from [8]). Both the preprocessing of R and the stabbing queries take time $\Lambda(n)$. We repeat the selection of R until we have $|L_{i+1}| \leqq |L_i|/2$.

In [8] it is shown that this strategy really works, namely that c_2 can be chosen so that all the bad lines are included in L_{i+1} with high probability – here the additional $\log n$ factor in the crossing number is needed, to have "really bad" lines – while $|L_{i+1}| \leqq |L_i|/2$ with probability at least 1/2. Thus the selection of R will be repeated only $O(\log n)$ times with high probability, while the bad lines are still caught with high probability.

The expected total time for the construction of $T_1, \ldots, T_k$ is thus $\Lambda(n)$. This proves Theorem 1.3. ■

7. DISCUSSION AND OPEN PROBLEMS

We use quite a lot of tools from computational geometry (ε-nets, range query structures, plane partitioning algorithm, etc.), whose combination gives

the results. It would be interesting to see whether such or slightly worse results could be achieved without such machinery. In particular, the results seem to be far from being practically applicable, because of rather large constants of proportionality and many logarithmic factors; may be simpler techniques could yield more practical algorithms.

It might be the case that some of our algorithms are optimal up to logarithmic factors. However, we suspect that at least the $\Lambda(n^{3/2})$ bound for a deterministic construction of a family of spanning trees could possibly be lowered. Also improving the $\Lambda(n^{4/3})$ bound for a (randomized) verification that the crossing number of a given tree is $O(\sqrt{n})$ (or proving its optimality) seems to be a difficult and challenging problem.

The execution time for the deterministic algorithm finding a single spanning tree is quite long mainly because we do not have a deterministic procedure for cutting a weighted arrangement into few triangles intersected only by a small fraction (in the sense of weight) of the lines, similar to the plane partitioning Lemma 4.1 *a*). This is another interesting open problem.

Added in proof: Recently, a new progress in these problems was made by the author [J. Matoušek: More on cutting arrangements and spanning trees with low crossing number, Tech. Report, FU Berlin, FB Mathematik, B-90-2]. E. g., the time complexity in Theorem 1.1 was improved to $\Lambda(n^{3/2})$.

REFERENCES

1. P. K. Agarwal, A deterministic algorithm for partitioning arrangements of lines and its applications, *Proc. 5. Ann. ACM Symposium on Comput. Geometry* (1989), pp. 11-21; *full version* to appear in *Discrete & Comput. Geometry*, 1990.
2. P. K. Agarwal, Ray shooting and other applications of spanning trees with low stabbing number, *Proc. 5. Ann. ACM Symposium on Comput. Geometry* (1989), pp. 315-325; *full version* to appear in *Discrete & Comput. Geometry*, 1990.
3. K. Clarkson, New applications of random sampling in computational geometry, *Discr. & Comput. Geometry*, 1987, 2, pp. 195-222.
4. B. Chazelle, Lower bounds on the complexity of polytope range searching, *J. Amer. Math. Soc.*, 1989, *2*, No. 4, pp. 637-666.
5. B. Chazelle and E. Welzl, Quasi-optimal range searching in spaces of finite VC-dimension, *Discr. & Comput. Geometry*, 1989, *4*, pp. 467-490.
6. H. Edelsbrunner, Algorithms in combinatorial geometry, Springer-Verlag, 1987.
7. H. Edelsbrunner and E. Welzl, Halfplanar range search in linear space and $O(n^{0.695})$ query time, *Inf. Proc. Letters*, 1986, 23, No. 6, pp. 289-293.
8. H. Edelsbrunner, L. Guibas, J. Herschberger, R. Seidel, M. Sharir, J. Snoeyink and E. Welzl, Implicitly representing arrangements of lines or segments, *Discr. & Comput. Geometry*, 1989, *4*, pp. 433-466.

9. H. EDELSBRUNNER, J.O'ROURKE and R. SEIDEL, Constructing arrangements of hyperplanes with applications, *SIAM J. on Computing*, 1984, *15*, pp. 341-363.
10. D. HAUSSLER and E. WELZL, ε-nets and simplex range queries, *Discr. & Comput. Geometry*, 1987, *2*, pp. 127-151.
11. D. G. KIRKPATRICK, Optimal search in planar subdivisions, *S.I.A.M. J. on Computing*, 1983, *12*, pp. 28-35.
12. F. P. PREPARATA and I. M. SHAMOS, Computational geometry – An introduction, Springer-Verlag, 1985.
13. V. N. VAPNIK and A. YA. CHERVONENKIS, On the uniform convergence of relative frequencies of events to their probabilities, *Theory Probab. Appl.*, 1971, *16*, pp. 264-280.
14. E. WELZL, Partition trees for triangle counting and other range searching problems, 4. *A.C.M. Symposium on Comput. Geometry*, 1988, pp. 23-33.