Geometric range searching*

Jiří Matoušek
Department of Applied Mathematics
Charles University
Malostranské nám. 25, 118 00 Praha 1, Czech Republic

Abstract

In geometric range searching, algorithmic problems of the following type are considered: Given an n-point set P in the plane, build a data structure so that, given a query triangle R, the number of points of P lying in R can be determined quickly. Problems of this type are of crucial importance in computational geometry, as they can be used as subroutines in many seemingly unrelated algorithms. We present a survey of results and main techniques in this area.

Contents

*Part of the work on this paper was done while the author was visiting the Computer Science Institute, Free University Berlin. The work has been supported by the German-Israeli Foundation of Scientific Research and Development (G.I.F.).

1 Introduction

This paper considers an algorithmic problem called *range searching*. We describe the problem and outline current theoretical knowledge about it, including the main ideas of several proofs and constructions.

> Together with the main material we also include various extensions, historical notes etc. Many of them are somewhat subjective. Several notions and results of computational geometry appearing in our discussion but not directly related to range searching are explained in separate boxes. Table 1 on page 43 summarizes the current best known complexity bounds for simplex and halfspace range searching.

Computational geometry — general remarks. The considered problems belong into the area of *computational geometry*. In the rest of this section we briefly introduce this field and mention some features and conventions which seem particular to it; reader somewhat familiar with computational geometry may probably skip the rest of section 1 safely. Older computational geometry monographs of a wide scope are [PS85],[Ede87],[Meh84], a recent one is [Mul93]. Useful survey papers can be found in [Pac93], [GPS91].

The subject of computational geometry is the design and analysis of efficient algorithms for computing various properties and parameters of finite configurations of geometric objects. The last sentence may sound rather incomprehensible, and the reader can get a better picture from few examples of simple problems studied in computational geometry:

P1. Given an n-point set P in the plane (or in a Euclidean space of dimension d)[1], find a pair of points of P with a smallest distance.

P2. Given n segments in the plane, compute the number of their intersections.

P3. Given a convex polytope P in d-dimensional space specified as an intersection of n halfspaces, determine a point of P minimizing a given linear function.

P4. Given a polygon P in the plane, not necessarily convex, and two points a, b in its interior, find a shortest path from a to b inside P, or (another version) a path from a to b consisting of a minimum possible number of segments.

> And so on. In computational geometry it is deceptively easy to formulate problems. Many problems have practical or theoretical motivations, and for almost any problem one can make up an acceptable looking application *ex post.* In the pioneer era of the field, say around the year 1980 and few years later, it was not too difficult to find an unsolved elementary and interesting problem. Today more complicated and more powerful methods are known, and finding a really remarkable new problem became quite hard.

[1] This is probably the most frequent preamble of a problem in computational geometry.

The subject of computational geometry is similar to the areas of interest of other fields, such as geometric aspects of combinatorial optimization etc. For instance, a careful reader might have recognized Problem P3 as a linear programming problem. The approach of computational geometry is usually distinguished by two typical features: The problems are considered in a fixed dimension, and an infinite precision model of computation is used.

Dimension and hidden constants. The original computational geometry problems were formulated in the plane or in 3-dimensional space. Generalizations to an arbitrary dimension are also studied, but almost always we imagine that the dimension is quite small, say at most 10, while the number of objects (points, segments etc.) appearing in the problem is large.

Formally the dimension is usually considered as a constant. The efficiency of typical computational geometry algorithms decreases with the dimension quite rapidly; for example, the multiplicative constants in the asymptotic bounds for the complexity of the algorithms are usually exponential in the dimension (this is also the case for range searching). This fixed dimension assumption contrasts e.g., with the theory of linear programming algorithms, where the dimension is comparable with the number of constraints.

In theoretical research, the efficiency of algorithms is compared, almost exclusively, by the asymptotic order of growth of their complexity (as a function of the input size). One can raise various objections against this criterion, mainly from the practical point of view, but no better theoretical criterion seems to be available.

> For simpler algorithms, mainly older ones, the effectivity expressed in this way agrees with the intuitive notion 'how fast the algorithm computes', and even with the speed of actual implementations for real life problems. For more complicated algorithms, this touch with reality is sometimes lost. Sometimes it is even apparent that an asymptotic improvement has nothing to do with the speed of computation for any physically feasible inputs. Attempts to improve an algorithm with $O(n \log \log n)$ complexity to an $O(n \log^* n)$ algorithm might look like some peculiar kind of sport.
>
> In defense of such activities one can say that the actual goal is, or at least should be, a better understanding of the studied problem. A discovery of a complicated and quite impractical algorithm with a good asymptotic complexity indicates the possibility of an improvement, and often a simpler and easily implementable algorithm of a similar complexity is found soon afterwards.

Model of computation, rounding problems. Algorithms in computational geometry are designed for an ideal computer (model of computation) called *Real RAM*. This is an analog of the usual RAM, i.e. of an abstraction of an actual computer programmed in a machine code. However, a Real RAM can store an arbitrary real number in one register and perform arithmetic operations with real numbers in unit time, while the usual RAM works with integers whose size is bounded by a polynomial in the input size. In the sequel we will also use the Real RAM model of computation.

The infinite precision computation in the Real RAM model is very convenient theoretically, but it often behaves quite differently than an actual computation with limited precision. Not only that the result of a calculation will be imprecise — a carelessly programmed algorithm may give completely wrong results with a limited precision, as the combinatorial information derived from comparisons of imprecise numbers may be erroneous or inconsistent. Difficulties with rounding errors are a very serious potential obstacle (perhaps the most serious one) to successful implementation of geometric algorithms.

Several promising attempts have appeared at developing program packages for a sufficiently precise and relatively quick arithmetic (e.g., [FvW93]). However, probably we have to expect that if a more complicated geometric algorithm is implemented reliably (i.e. in such a way that it cannot give an erroneous result because of rounding errors), it causes a significant slowdown compared to a straightforward implementation ignoring these aspects.

General position assumption. In many computational geometry algorithms, one has to deal separately with configurations of input objects that are degenerate in some sense, for instance when three input points lie on a common line. Such exceptions complicate both the description and implementation of algorithms. Luckily for theoreticians, it is known how to avoid such degenerate cases systematically, although at the cost of a slowdown by a constant factor (at least in versions published until now). Conceptually, one perturbs the input objects by infinitesimal amounts which brings them into a general positions. Such methods are called *simulation of simplicity.* The idea of infinitesimal perturbations can be realized in various ways, see e.g., [EM90], [EC91], [EC92], and the best method is still being sought.

Simulation of simplicity has also theoretical drawbacks: Since we are in effect replacing a given input problem by a different problem (although an infinitesimally close one), an algorithm run on this different problem may sometimes yield a different answer than the correct one for the original problem, and it may actually be quite complicated to recover a correct answer to the original problem. Of course, we may say that the input numbers for a “real world” problem are inaccurate anyway, and so the answer for the perturbed problem is equally appropriate as the one for the original problem. However, this is not suitable for many applications, where degeneracies in the input are not mere “coincidences”, but rather are important for the problem structure.

Deterministic and randomized algorithms. The complexity of computational geometry algorithms is most often estimated in the worst case, which means that the estimate must hold for each admissible input of a given size (one may imagine that the input is given by an adversary who knows the algorithm and tries to make it as slow as possible).

Much fewer results concern the average case, where the input is considered as a random variable, and the expected complexity of the algorithm for such an input is estimated (e.g., for input points selected independently from the uniform distribution in the unit square). The most problematic point of the average case analysis is the choice of the probability distribution on the inputs. Often various natural possibilities exist which give significantly different expected behavior of the algorithm. For geometric problems it is usually much easier to give an algorithm with a good average case complexity than an algorithm with a comparable worst case complexity. This is the case also for geometric range searching, which is almost trivial for points uniformly distributed in the unit square.

Older algorithms are, with few exceptions, *deterministic*, which means that their computation is uniquely determined by the input. Only recently so-called *randomized algorithms* appeared in computational geometry. These algorithms randomly choose one from several possible continuations at some stages of their computation[2]. For a fixed input, the complexity of such an algorithm is a random variable. One estimates its expectation, or also further parameters (such as the probability of large deviations from the expectation, *tail estimates*), and then considers the worst case over all inputs (thus the input is chosen by the adversary, but he cannot influence the random decisions of the algorithm).

Sometimes also so-called Monte-Carlo algorithms are studied. These may sometimes give a wrong result, but with a very small probability only (such a bound again holds for every input).

In a seminal paper [Rab76] on randomized algorithms M. O. Rabin gives a typical computational geometry problem as one of two examples; at that time computational geometry as a field did not exist yet. Also a paper of Chew [Che85] with a very elegant application of randomness as well as earlier works of Clarkson went almost unnoticed. In last few years, however, the randomized algorithms became dominating in computational geometry, promoted by the works of Clarkson ([Cla87], [Cla88a], [Cla88b] and others), Haussler and Welzl [HW87], Sharir, Chazelle, Edelsbrunner, Guibas (e.g. [CEG+90]), Mulmuley ([Mul88], [Mul91a], [Mul91b] etc.), Seidel ([Sei91a], [Sei91b]) and others. In most cases randomized algorithms are simpler, more effective and easier to implement than deterministic ones. For most problem deterministic algorithms with a similar or only slightly worse asymptotic complexity were found (see e.g., [CF90], [Mat90], [Mat91b], [Cha93], [Cha91], [CM93a]), but for practical purposes the randomized algorithms are still winning by their simplicity and smaller hidden constants.

2 Geometric range searching

Let $\mathcal{R}$ be a system of subsets of the d-dimensional Euclidean space $\mathbb{R}^d$. The sets of $\mathcal{R}$ are called *ranges*. The following are typically considered cases

$$\begin{aligned} \mathcal{R}_{orthog} &= \textit{axis-parallel boxes}\text{, i.e. all sets of the form } \prod_{i=1}^{d}[a_i,b_i],\ a_1,b_1,\ldots,a_d,b_d \in \mathbb{R} \\ \mathcal{R}_{halfsp} &= \text{the set of all (closed) halfspaces in } \mathbb{R}^d \\ \mathcal{R}_{simplex} &= \text{the set of all (closed) simplices in } \mathbb{R}^d \end{aligned}$$

[2]In actual implementations, we don't have a true randomness at our disposal, and random decisions are simulated using pseudorandom numbers. Then the randomness in the algorithm is restricted to the initial setting of the random number generator, which represents only few random bits. Empirical evidence indicates that even this weak randomness is sufficient for most of the algorithms, and recent works have already confirmed this for some classes algorithms, see [Mul92].

$$\mathcal{R}_{ball} \quad = \quad \text{the set of all (closed) balls in } \mathbb{R}^d \; .$$

Further let P be a given n-point set in $\mathbb{R}^d$. One of the geometric range searching problems is the following:

> Design an efficient algorithm, which, for a given range $R \in \mathcal{R}$, finds the number of points of P lying in R.

If the set P and the range R were given together and the problem ended by determining the number $|P \cap R|$, it would be simplest to go through all points of P one by one and count those lying in R. In fact, in such a situation we can hardly do anything better. In our problem, however, the point set P is given in advance, and we can prepare some auxiliary information about it and store it in a suitable data structure. Then we will be repeatedly given various ranges of $\mathcal{R}$ as queries. Each such query is to be answered as soon as it appears (on-line), and as efficiently as possible. Assuming that the number of queries will be large, it will be advantageous to invest some computing time into building the data structure (this phase is called the *preprocessing*), if this makes the query answering faster.

> **Example 2.1** Let us look at the situation in dimension $d = 1$, with intervals as ranges. Without any preprocessing, a query answering (counting the number of points in a given interval) requires time proportional to n. However, having stored our points in a linear increasingly sorted array, we can answer queries in $O(\log n)$ time: We locate the position of the endpoints of the query interval among the points of P by binary search, and then we find the required number by a subtraction of indices. Preprocessing, in our case sorting the points of P, can be performed in $O(n \log n)$ time, and $O(n)$ memory is sufficient for storing the data structure.

Counting points in a given range (a *range counting query*) is only one of possible range searching problems. Another natural problem is to compute a list of all points of P lying in a query range (we speak of a *range reporting query*), or we can only ask if the query range contains any point of P at all (*range emptiness query*). Also, each points of P can be assigned some weight (e.g., a real number), and we can be interested in the sum of weights of points in a given range, or in the maximum weight.

> All these problems are quite convincingly motivated by direct practical applications, most often in various database systems. A rather banal example: The points of P might correspond to employees of some company, with the coordinates representing age, salary, time of employment etc. Then, in order to find an employee with age between 30 and 35, with at least 8 years of experience and salary below 4000 per month[3], one might apply a query with an axis-parallel box from $\mathcal{R}_{orthog}$. A circular range query might serve for locating airports, where a defective airplane still succeeds to land, etc. Perhaps more important than such direct applications are applications of geometric range searching as subroutines in the design of algorithms for more complicated geometric problems.

In more recent papers, one usually investigates a unifying generalization of various range searching problems. We assume that every point $p \in P$ is assigned a weight $w(p) \in S$, where $(S, +)$ is some semigroup (common to all

[3]The reader may supply a currency unit according to his own preference.

the points). The objective of a query is to find the sum of weights of all points of P lying in a given range, $\sum_{p \in R \cap P} w(p)$. For example, for range counting queries, $(S, +)$ will be the natural numbers with addition, and all weights will be equal to 1. For queries on maximum weight, the appropriate semigroup will be the real numbers with the operation of taking a maximum of two numbers. For emptiness queries, we may choose the boolean values $\{\text{false}, \text{true}\}$ for S, with the operation of logical disjunction (OR); all the point weights will be 'true'.

Concerning the computational aspects, we usually assume that the weights can be stored in a single computer word, and that the semigroup operation can be executed in constant time.

The range reporting queries have a somewhat special position. We could also include them into the above discussed semigroup model — the semigroup $(S, +)$ would be the set of all subsets of P, the operation would be the union and the weight of each point $p \in P$ would be the one-point set $\{p\}$. However, in reasonable models of computation a subset cannot be represented in a constant amount of memory, and also the union operation requires a nonconstant time. From the algorithmic point of view, a further specialty of range reporting is the following: If the answer consists of k points, then we need time of order k only to output the answer. Thus, we may spend further $O(k)$ time for auxiliary computations, without decreasing the overall asymptotic complexity. If we know that the answer is going to be large, we can afford to compute it slowly. This was observed and cleverly applied by Chazelle [Cha86]; he calls this the *filtering search*. The query complexity of range reporting algorithms is usually expressed in the form $O(f(n) + k)$, where k is the number of points in the answer, and $f(n)$ is some function of the total number of points in P, called the *overhead*.

Let us return to the data structure from Example 2.1, which was a simple sorted array. We can use it to answer interval reporting queries in $O(\log n + k)$ time, and it is easy to generalize it for queries seeking the sum of real weights in a given interval. In this case, as well as for the interval counting queries, we make use of the possibility to subtract the weights (that is, the semigroup $(S, +)$ can be embedded into a group). If this is not possible (in cases such as the finding of the maximum weight), we need a different (and more complicated) data structure.

Here is a simple and well-known example of such a data structure:

Example 2.2 Consider the points of $P \subset \mathbb{R}^1$ in sorted order, and build a balanced binary tree T with the points of P as leaves. With every node v of T, store the total weight of points in the leaves of the subtree rooted at v (note that these points form an interval in P; such intervals are called the *canonical intervals*). Given a query interval (a, b), we search the position of a and b among the leaves of T, and the search paths give us a decomposition of the set of points of P lying between a and b into a disjoint union of $O(\log n)$ canonical intervals. Indeed, let π_a and π_b denote the search paths for a and b, respectively, and let x be the node where they branch. Then for every node $v \in \pi_a$ lying below x and such that the next node of π_a is the left son of v, we take the right subtree of v, and it will define one of the canonical intervals. We proceed symmetrically for the portion of π_b below x, obtaining the desired decomposition. Thus, to answer the query, it remains to sum the precomputed weights of the canonical intervals in the decomposition. Note that this does not use the weight subtraction, so that we can find e.g., the point of maximum weight in a given interval.

An analogous situation appears also for more complicated geometric range searching problems: Reporting queries and queries with weights which can be subtracted often allow a simpler and/or more efficient solution than the general case, whose prototype are the queries asking for maximum weight. For the case of subtraction, we can usually express the answers for more complicated ranges using the answers for several simpler ranges. For instance, in our one dimensional example, we have implicitly expressed a query interval as the difference of two semiinfinite intervals.

For a full specification of a range searching problem we need, in addition to the already mentioned objects (the set P, the semigroup $(S,+)$, the weights of points and the set of admissible ranges $\mathcal{R}$), also limits on the maximal permissible storage and preprocessing time for the data structure.

Such a limitation does not show up in Example 2.1, where the query time $O(\log n)$ is optimal, and at the same time if we should be able to answer queries at all, we need at least a linear storage. However, for halfspace and simplex range searching one cannot achieve both a linear storage and a logarithmic query time, and so we have to choose the most efficient query answering algorithm depending on the amount of memory and preprocessing time we have at disposal. The memory requirements are usually considered more important; this has roots in the database applications, where the preprocessing needs to be done only once and we can spend a relatively long time for it, while the memory or disk space is allocated permanently and in each copy of the database. On the other hand, in many applications the preprocessing time is as critical as the storage.

In this paper we will mainly consider the simplex and halfspace range searching problems. These problems turned out to be crucial in computational geometry, they are even universal in some sense, since many other problems with more general ranges can be reduced to them, see e.g., [YY85], [AM92] and also section 6.4.

The algorithms for simplex and halfspace range searching problems have been applied also in problems which look much more complicated. A nice example is in the paper of de Berg *et al.* [dBHO$^+$], which was motivated by a computer graphics problem. The range searching problem, which the authors need to solve as part of their hidden surface removal algorithm, is as follows.

Problem 2.3 Given a set $s_1, \ldots, s_n$ of segments in $\mathbb{R}^3$, we imagine that there is a semiinfinite curtain $c(s_i)$ hanging from each segment s_i downwards; we have

$$c(s_i) = \{(x,y,z) \in \mathbb{R}^3;\ (x,y,z_0) \in s_i \text{ for some } z_0 \geq z\}$$

(the curtains may intersect). We want to design a data structure for answering queries of the following type: Given a point o and a direction θ, find the first curtain hit by the ray sent from o in direction θ.

Both the original solution of this problem in [dBHO$^+$], and a more efficient method described in [AM93] reduce this problem to a combination of simplex range searching and halfspace range searching.

The orthogonal range searching problems (the ones with axis-parallel boxes as ranges) are no less important than the simplex range searching, perhaps even more important for direct applications. In this paper we will not consider them in detail, but for reader's convenience we briefly summarize the main results concerning orthogonal range searching. The basic idea of data structures for this problem, the so-called *range tree*, was described by Bentley [Ben80]. Its straightforward use gives query time $O(\log^d n)$ with $O(n \log^{d-1} n)$ storage and preprocessing time in dimension d. Various improvements in some parameters, mostly by logarithmic factors, were achieved in the works of Willard [Wil85], Willard and Lueker [WL85], Gabow, Bentley and Tarjan [GBT84] and Chazelle [Cha86], [Cha88]. The exact complexities differ depending on the type of the problem (range counting queries, range reporting queries, general semigroup queries etc.) and on the model of computation (pointer machine, RAM etc.). Lower bounds for the computational complexity of orthogonal range searching were given by Chazelle [Cha90a],[Cha90b], and they match the known upper bound almost exactly. A variant where the points lie on a grid of a bounded size was studied by Overmars [Ove88]; in this case somewhat better results are obtained.

3 Intuition and lower bounds

In this section we start considering the simplex and halfspace range searching problems in more detail. Results about their computational complexity can be summarized as follows:

Let us consider a simplex range searching problem for an n-point set $P \subset \mathbb{R}^d$, with weights from a semigroup $(S, +)$, and with storage and preprocessing time at most m, where m lies in the range from n to approximately n^d. Then the query time is approximately

$$\frac{n}{m^{1/d}}\,. \tag{1}$$

The word "approximately" in the previous sentence means "up to a multiplicative factor bounded by $O(\log^c n)$, c a constant."

In particular, for an approximately linear storage the query time is approximately $n^{1-1/d}$, and in order to achieve a polylogarithmic query time, one needs space and preprocessing approximately n^d. The complexity of halfspace range queries is believed to be very similar, except for few special cases, as e.g., halfspace emptiness queries, which can be handled more efficiently.

In this section we formulate known lower bounds, which show that under certain restrictions on the algorithm and the model of computation the query time (1) is approximately optimal. It is possible (although currently it does not seem very likely), that one could circumvent these lower bounds somewhat using algorithms that do not satisfy the corresponding assumptions. It would not be the first such case in theoretical computer science.

First we will try to give the reader some intuitive explanation where the formula (1) comes from. The explanation is quite far from a proof, and in reality the lower and upper bounds work in a more complicated manner; it is meant just for a basic orientation. We will consider the two extreme cases, polylogarithmic query time and roughly linear space.

Logarithmic query time. First we consider halfspace queries. It is not difficult to see that for an n-point set P in a general position there are $\Theta(n^d)$ different subsets of the form $P \cap R$, where R is a halfspace (this is best seen in the dual setting, where distinct subsets correspond to distinct cells in an arrangement of hyperplanes, see the boxes on pages 24, 25). Storage of the order n^d thus means that we can store the answers for all essentially different halfspaces that can ever appear. Actual algorithms are indeed based on this principle. A naive attempt on extending this idea to simplex range searching results in a much larger space than n^d. A suitable method preserving storage close to n^d is more complicated and was discovered only recently [CSW92].

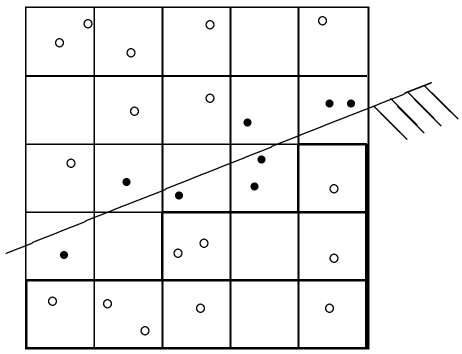


Figure 1: A simple halfspace range searching method, for uniformly distributed point sets

Approximately linear storage. Here we will assume that the set P is chosen randomly, by n independent random draws from the uniform distribution in the unit square (we consider the planar case first). We put $t = \lfloor \sqrt{n} \rfloor$ and we cover the unit square by a $t \times t$ square grid, each grid square having side $1/t$, see Fig. 1. With high probability, almost every grid square then contains only a small number of points of P (bounded by a constant).

Let R be a given halfplane. We note that the boundary line h of R intersects at most $2t$ squares of the grid (if its slope is at most 1, the it intersects at most 2 squares in every column, and for slope > 1 we apply a similar argument with rows).

For the squares intersected by h we go through all the points of P lying in them, and for each such point we test its membership in R. The uniform distribution implies that the number of points processed in this phase is $O(t) = O(\sqrt{n})$ (such points are marked as full circles in Fig. 1) .

In remains to account for the weight of points in grid squares which are completely contained in R. This can be done row by row, using the fact that such squares form a contiguous interval in every row. The total weights of points in each such segment of each row are computed in advance, thus we only need a constant time per row for the query answering. The total memory requirement is $O(n)$.

For a higher dimension d we can proceed quite similarly, dividing the unit cube into a grid of cubes with sides $n^{-1/d}$. The bounding hyperplane of a given halfspace R always intersects only $O(n^{1-1/d})$ grid cubes. The cubes completely contained in the query halfspace can be processed by columns parallel to one (arbitrarily chosen) coordinate axis. In this way we get a data structure with $O(n)$ storage and $O(n^{1-1/d})$ query time for uniformly distributed point sets in the unit cube, as required by formula (1).

This time also the generalization to simplex range searching is straightforward. We associate a suitable one-dimensional data structure for range searching in intervals with every column of the grid; we leave the details to the reader.

It is quite conceivable that this simple data structure is the most practical option for simplex range searching with linear space, at least for roughly uniformly distributed sets (this method somewhat resembles the *quadtrees*, see [PS85]). For point sets which are not uniformly distributed this simple approach fails, and all known methods with query time close to $n^{1-1/d}$ are considerably more complicated.

Lower bounds. Lower bounds are due to Chazelle and his students; previous weaker results were obtained by Fredman [Fre81]. The basic work [Cha89] bounds from below the computational complexity of simplex range searching with weights from a suitable semigroup. In order to formulate this result, we have to define an appropriate model of computation, the so-called *arithmetic model* originally introduced by Yao and Fredman as means for lower bound proofs.

Roughly speaking, the arithmetic model only considers the number of semigroup operations needed to compute the answer to a query. Certain partial sums of the point weights are precomputed and stored in memory; such partial sums are called *generators*. Their number cannot exceed the prescribed storage, m. The arithmetic model does not at all consider various auxiliary operations of the algorithm needed in the query answering, such as finding out which generators are to be added to yield the result. This is a strength of lower bounds in the arithmetic model. On the other hand, this model puts various restrictions on the query answering algorithm. One of the most significant restrictions is the impossibility of subtractions of the weight, even in the weights happen to belong a group. Let us now pass to exact definitions.

A semigroup $(S,+)$ is called *faithful* if the following holds: For any $n>0$, any two distinct subsets $A,B \subseteq \{1,2,\ldots,n\}$ and natural numbers $\alpha_i > 0$ $(i \in A)$, $\beta_j > 0$ $(j \in B)$, there exist elements $s_1,\ldots,s_n \in S$ so that

$$\sum_{i\in A} \alpha_i s_i \neq \sum_{j\in B} \beta_j s_j \, .$$

In other words, two linear forms over variables ranging over S are never identically equal unless they have the same set of variables. Faithfulness is a quite weak condition. An example of a semigroup which is not faithful is $\mathbb{Z}/2\mathbb{Z}$, i.e. $\{0,1\}$ with addition modulo 2, where the identity $2x = 2y$ holds. On the other hand, real numbers both with the operation of addition and with the operation of pairwise maximum do form faithful semigroups.

In the sequel let $(S,+)$ be a fixed faithful semigroup. Any linear form $g(s_1,\ldots,s_n) = \sum_{i=1}^{n} \alpha_i s_i$, where s_i are variables ranging over S and α_i are nonnegative integers, is called a *generator*.

It turns out to be advantageous to consider geometric range searching problems in a more abstract form. To each range searching problem with point set $P = \{p_1, p_2, \ldots, p_n\} \subset \mathbb{R}^d$ with a set $\mathcal{R}$ of admissible ranges we assign a set system $(P, \mathcal{P})$, where $\mathcal{P}$ is the system of all subsets of P definable by ranges from $\mathcal{R}$, that is,

$$\mathcal{P} = \{P \cap R;\ R \in \mathcal{R}\}\,. \tag{2}$$

Then we work with this set system only, thus restricting the geometric contents of the problem to a minimum. This more abstract approach turned out to be fruitful not only for geometric range searching; see e.g., [HW87], [Mat91b].

A set $\Gamma = \{g_1, \ldots, g_m\}$ of generators is called a *scheme* for the set system $(P, \mathcal{P})$, if for every set $Q \in \mathcal{P}$ there exist nonnegative integer coefficients $\beta_1, \ldots, \beta_m$ such that for *any* choice of weights $w(p_1), \ldots, w(p_n) \in S$ we have

$$\sum_{p \in Q} w(p) = \sum_{i=1}^{m} \beta_i g_i(w(p_1), \ldots, w(p_n))\,. \tag{3}$$

It is thus required that using the generators of the scheme one can express the weight of any set $Q \in \mathcal{P}$, and this expression must be of the same form for any choice of weights of the points of P.

A scheme Γ for $(P, \mathcal{P})$ is called a (t,m)-scheme if $m = |\Gamma|$ and for every $Q \in \mathcal{P}$ the coefficients in (3) can be chosen in such a way that at most t of the numbers $\beta_1, \ldots, \beta_m$ are nonzero. Then it is natural to define that the geometric range searching problem with the point set P, set of ranges $\mathcal{R}$ and with weights from $(S,+)$ has query complexity at least t in the arithmetic model for storage m if there is no $(t-1, m)$ scheme for $(P, \mathcal{P})$, where $\mathcal{P}$ is as in (2).

> Let us pause for a moment to give a simple example of what the generators might look like in an actual algorithm. Considering Example 2.2, we have one generator for every canonical interval, namely the sum of point weights in that interval. Hence each generator is just the sum of weights over some subset of P, and the answer to a query is computed by expressing the point set in the query range as a disjoint union of canonical subsets, therefore all coefficients in (3) are either 0 or 1. Other range searching algorithms for semigroup weights also use this more special form of generators and answer computation, in fact we are not aware of any single instance where the more general form allowed by the arithmetic model would be used.

Lower bounds in the arithmetic model hold only for algorithms computing the answer (the total weight of points in a query range) using a scheme in the just defined sense. There might thus exist, in principle, a better algorithm using the specific weights in a given range searching problem (while an algorithm covered by the arithmetic model must work "uniformly" for any choice of weights). Lower bounds also do not apply to algorithms using weight subtraction (if $(S,+)$ is a group). Proving lower bounds valid also for this group case is one of the main challenges in this area, another one being a proof of lower bounds for emptiness queries with halfspaces or simplices. Let us remark that an orthogonal range searching type problem is known where the complexity in the group model is provably better than in the semigroup model, see [CR91].

Now that we have overcome all these definitions, we can quote the main result of [Cha89].

Theorem 3.1 (Chazelle) *Let $(S,+)$ be a faithful semigroup. Then for any fixed dimension d and parameters n, m there exists an n-point set $P \subset \mathbb{R}^d$ such that the simplex range searching problem with point set P, weights from S and storage at most m has query complexity at least*

$$\Omega\left(\frac{n}{\log n \; m^{1/d}}\right) \tag{4}$$

(for $d \geq 3$), resp. at least

$$\Omega\left(\frac{n}{\sqrt{m}}\right) \tag{5}$$

for $d = 2$ in the arithmetic model.

The proof of this theorem in fact gives somewhat stronger results. First of all, P is no artificially constructed pathological set, rather it suffices to choose it randomly from the uniform distribution in the unit cube — we obtain a "hard" set with high probability. Also the hard query simplex need not be chosen in any too special way. The proof uses, instead of simplices, so-called *slabs*, which are the parts of space bounded by two parallel hyperplanes. The proof shows that for a randomly chosen slab of a suitable width (among the slabs intersecting the unit cube) the query complexity is at least as shown by the lower bound formulas. In this sense (4), (5) bound not only the worst case, but also the average case.

It is quite likely that the bound (4) holds without the logarithmic factor in the denominator as well (as is the case in dimension 2). Such an improvement has an interesting relation to a generalization of a famous problem of combinatorial geometry, the so-called Heilbronn problem. The Heilbronn problem itself can be formulated as follows:

Problem 3.2 *For a set $P \subset \mathbb{R}^2$, let $a(P)$ denote the area of a smallest triangle with vertices in P. What is the asymptotic behavior of the function $a(n) = \sup\{a(P);\ P \subset [0,1]^2,\ |P| = n\}$?*

This is a very nice problem, it has been worked on by many excellent mathematicians and its complete solution still seems to be quite far of, see e.g., [KSP82], [Rot76] for a survey of results and references. The following is a generalization related to Chazelle's lower bound proof method:

Problem 3.3 *For a set $P \subset \mathbb{R}^d$, denote by $a_d(P,k)$ the smallest volume of the convex hull of a k-tuple of points of P. What is the asymptotic behavior of the function $a_d(n,k) = \sup\{a_d(P,k);\ P \subset [0,1]^d,\ |P| = n\}$?*

Chazelle has shown that for a suitably chosen set P in the unit cube the volume of the convex hull of each k-tuple is at least proportional to k/n for any $k \geq c \log n$, with a sufficiently large constant c. In other words,

$$a_d(n, k) = \Omega(k/n) \tag{6}$$

for $k \geq c \log n$. This is essentially a result about uniform distribution of the set P, saying that no k-tuple is too clustered (in the sense of volume). If (6) could also be proved for smaller k, an improvement of (4) would follow immediately.

Such an improvement may not be easy. Known results for the Heilbronn problem imply that (6) is false for $d = 2$, $k = 3$ (which may contradict intuition somewhat). However, it is not known that (6) could not hold for larger but constant k. On the other hand, this connection shows that by finding an algorithm for simplex range searching with $o(n/m^{1/d})$ query complexity in the arithmetic model one would also get a nontrivial result for the above mentioned difficult combinatorial problem.

At this point the reader might ask, where the better bound (5) in the plane comes from, if the generalized Heilbronn's problem is open for dimension 2 as well as for higher dimensions. The main reason is that in dimension 1, it is easy to construct a point set satisfying (6) for every $k \geq 2$. One particular construction, although not the most straightforward one, is to take a random uniformly distributed set, and throw out a suitably selected half of the points. In Chazelle's proof, subsets of P defined by slabs are considered, and what matters is the one-dimensional distribution of the orthogonal projection of each such subset onto the axis of the corresponding slab. To obtain (5) it suffices to show that for most of the slabs, one can select half of the points from the projection with a good one-dimensional distribution in the sense of (6).

In spite of Theorem 3.1 one might hope for better algorithms in some particular cases, such as for halfspace range searching, or for simplex emptiness queries etc. A better algorithm is known essentially for one special case only, namely for halfspace emptiness queries, or for halfspace range reporting queries (see section 6.1). On the other hand, it seems that both simplex emptiness queries and halfspace range counting should be approximately equally difficult as the general simplex range searching problem. This was partially substantiated in [BCP93] a [CR92]. Brönnimann *et al.* [BCP93] showed that under similar assumptions as in Theorem 3.1 the *halfspace* range searching problem for an n-point set with storage at most m has query complexity at least

$$\Omega\left(\left(\frac{n}{\log n}\right)^{1-\frac{d-1}{d(d+1)}} / m^{1/d}\right)$$

in the arithmetic model. This bound might still be lower than the actual complexity, but at least it shows that the halfspace range searching problem with arbitrary weights is more difficult than the halfspace emptiness problem, where a query time of approximately $n/m^{1/\lfloor d/2 \rfloor}$ can be achieved.

Chazelle and Rosenberg [CR92] consider the simplex range reporting problem. Their model of computation is different than in the previous results; they use the so-called *pointer machine*. They prove that for any data structure occupying space at most m there exists a halfspace for which the reporting query requires time at least $\Omega(n^{1-\varepsilon}/m^{1/d}+k)$, where k denotes the number of points in that halfspace.

> The papers of Chazelle *et al.* on lower bounds may please the reader as a nice piece of mathematics, but the conclusions for the simplex range searching problem are quite pessimistic. For instance, if we want to improve the query complexity K-times compared to the trivial algorithm (consisting of inspection of every point of P) in dimension 10, it costs storage of the order K^{10}. Since we pass from a trivial algorithm to a more complicated one, we may expect that the hidden constants of proportionality will work against us. Nontrivial algorithms can thus be practically useful for a really small dimension only. This is, however, quite a frequent feature of computational geometry algorithms, as we have already mentioned in the introduction.

4 Algorithms with approximately linear space

We divide our discussion of simplex range searching algorithms into two parts. First we consider algorithms using linear or nearly linear storage, then algorithms attaining a short — polylogarithmic — query time. As was explained above, the latter algorithms require quite large space, of the order n^d. These two groups of algorithms have been investigated separately in the literature, and more attention has been paid to linear space algorithms, which are usually more complicated and more interesting. Algorithms with memory requirements in between these two extremes can often be obtained by a more or less straightforward combination of algorithms of the two mentioned types.

Most of nontrivial algorithms with linear space (nontrivial meaning with substantially sublinear query time) are based on the idea of *partition trees* due to Willard [Wil82]. We will explain it in a simple form for halfplane range searching.

Let P be an n-point set in the plane. For simplicity we assume that n is of the form 4^k and that P is in general position. Let ℓ_0 be an arbitrarily chosen line halving P ($n/2$ points lie below ℓ_0 and $n/2$ points above ℓ_0). The classical *ham-sandwich cut theorem*, see e.g., [Ede87] guarantees the existence of another line ℓ_1 such that each of the 4 regions A_1, A_2, A_3, A_4 into which ℓ_0, ℓ_1 partition the plane contains exactly $n/4$ points of P, see fig. 2(a). Consider an arbitrary halfplane R with the bounding line h. It is easy to see that h cannot intersect all 4 regions A_1, A_2, A_3, A_4, and hence one of them, A_i, lies completely outside R or completely inside R.

If we precompute the total weight of points in each A_j, we can process all points of the region A_i missed by h in a single step, thus saving 25% of the work on the query answering compared to the trivial method. This is not significant for the asymptotic complexity, but a similar saving can be repeated recursively in each of the remaining 3 regions. Continuing in a similar manner also in a larger depth, the complexity is decreased significantly.

A data structure based on this idea, a partition tree, has a root containing the coordinates of the partitioning lines ℓ_0, ℓ_1 as well as the total weights of points in the 4 regions. The root has 4 sons, corresponding to the 4 regions

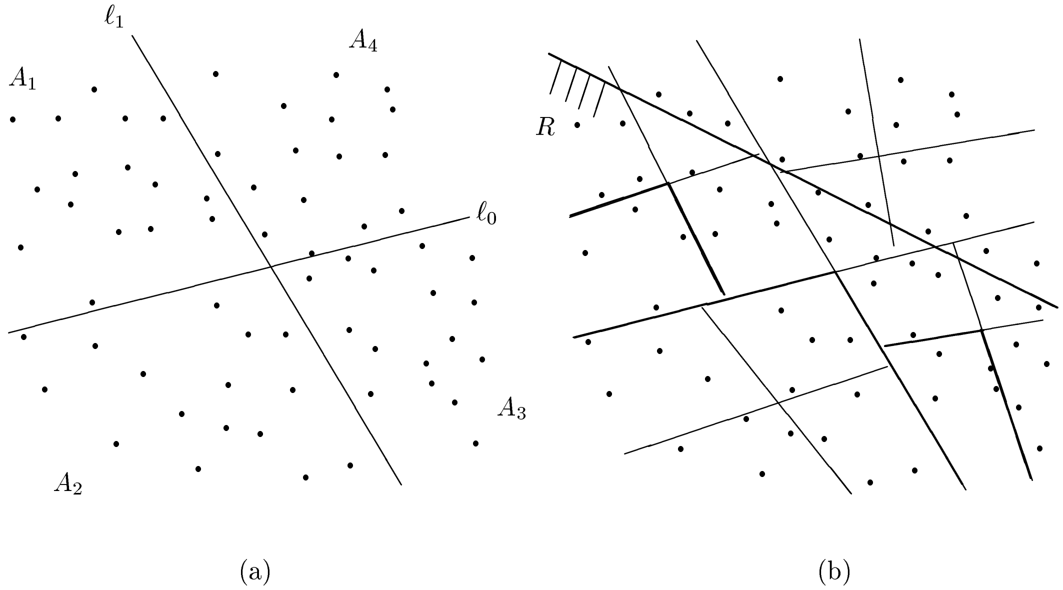


Figure 2: (a) Ham-sandwich cut theorem. (b) Two levels of the recursive construction. For the halfplane R, the regions bounded by thick lines are processed as wholes.

$A_1, \ldots, A_4$, and the ith son is the root of a partition tree constructed similarly for the set $P \cap A_i$. This recursive construction terminates after reaching small enough sets, say one-point ones.

To answer a query with a halfplane R, we start in the root of the partition tree. We find the region missed by the boundary of R and we process it immediately (ignoring it if it lies outside R, otherwise storing its weight to a global variable for accumulating the total weight). For the remaining 3 regions we proceed similarly in the corresponding 3 sons of the root. When we reach a leaf of the partition tree with a trivially small set we process the point stored there directly.

If $T(n)$ denotes the query time for an n-point set, we get the recurrence

$$T(4^k) \leq C + 3T(4^{k-1}),$$

with an initial condition $T(1) \leq C$, where C denotes a suitable constant. This gives a bound $T(4^k) = O(3^k)$. Both the construction and the analysis can be extended to an arbitrary value of n, and we obtain $T(n) = O(n^{\log_4 3}) \approx O(n^{0.792})$, thus a significantly sublinear function indeed. It is easy to verify that the described data structure occupies $O(n)$ space only.

The above defined partition tree can also be used for answering triangular range queries. The query answering algorithm is very similar: In a current node of the partition tree, we process the regions missed by the boundary of the query triangle σ directly (since they lie completely inside σ or completely outside σ), and for regions intersected by the boundary of σ we proceed recursively in the appropriate sons of the current node.

Here we cannot argue that in every node the recursion visits at most 3 of its sons. Rather we note that σ is an intersection of 3 halfplanes, and it is not difficult to check that any node of the partition tree visited by the query answering algorithm for the triangle σ will also be visited when answering the query with at least one of the 3 halfplanes. Thus the query time for triangles is of the same order as the query time for halfplanes.

Early improvements and generalizations of partition trees. The idea of a partition tree is simple but by no means obvious. Improving the above described construction and generalizing it to higher dimensions was not easy either, and the history resembles athletic records somewhat. Willard himself found a somewhat better construction than the one we have described, with query time of the form $O(n^{\alpha})$ with $\alpha \approx 0.774$. Edelsbrunner and Welzl [EW86] improved the above presented method by a better application of the ham sandwich cut theorem: Instead of dividing each of the regions $A_1, \ldots, A_4$ independently, they bisect both A_1 and A_4 simultaneously by a single line, and similarly for A_2 and A_3. Proceeding recursively in this way, they obtain the recurrence $T(n) \leq C + T(n/2) + T(n/4)$, leading to the bound $\alpha \approx 0.695$. The preprocessing time for the partition tree construction is $O(n \log n)$ and it relies on a sophisticated linear-time algorithm for the ham-sandwich cut theorem for linearly separated point sets due to Megiddo.

A first generalization for dimension 3, with $\alpha \approx 0.98$, was obtained by F. Yao [Yao83], who showed the possibility of partitioning a point set in $\mathbb{R}^3$ into 8 parts of relative size at most $1/24$ by 3 planes. This was improved by Dobkin and Edelsbrunner [DE84] ($\alpha \approx 0.916$), Edelsbrunner and Huber [EH84] ($\alpha \approx 0.909$) and Yao *et al.* [YDEP89] ($\alpha \approx 0.8988$). The latter authors re-discovered a result of Hadwiger, namely that any point set in $\mathbb{R}^3$ can be partitioned into 8 equal parts by 3 planes. They organize the recursive construction in a similar way as [EW86]. Avis [Avi84] noted that a partitioning of a point set in $\mathbb{R}^d$ into 2^d equal parts by d hyperplanes is not possible in general for $d \geq 5$. For $d = 4$, the existence of such a partition was recently established by Ramos (private communication by H. Edelsbrunner).

Cole [Col85] found a construction in dimension 4 giving $\alpha \approx 0.977$, and shortly after that Yao and Yao [YY85] showed the existence of a nontrivial simplex range searching algorithm with linear space in any fixed dimension, although their exponent $\alpha = [\log_2(2^d - 1)]/d$ only differs from 1 by a very small amount. Let us describe the beautiful construction of Yao and Yao's partition scheme.

First, the discrete point set P is replaced by a continuous, everywhere positive mass distribution μ: each point $p \in P$ is replaced by a small dense ball of radius ε and weight $1 - \varepsilon$ plus a light nebula of total weight ε spreading out to infinity. It is easy to show that is $\varepsilon > 0$ is small enough, a good partition scheme for μ will also be good for the original point set P. This passage to a continuous setting enables to apply topological arguments (and it is used e.g., in proofs of the ham-sandwich cut theorem).

The construction of the partition scheme proceeds by induction on the space dimension d. Given μ, one constructs a point $C(\mu)$ (the *mass center*) and a set $\Pi(\mu)$, which consists of a finite number of pieces of hyperplanes and whose removal partitions $\mathbb{R}^d$ into the desired regions. For $d = 1$, $C(\mu)$ is the (unique) point bisecting μ and $\Pi(\mu) = \{C(\mu)\}$. Now let $d > 1$

and let S be the hyperplane perpendicular to the x_d-axis and bisecting μ. For any vector v with a positive x_d-coordinate, we define two $(d-1)$-dimensional mass distributions in S, denoted by μ_v^+ and μ_v^-. The mass distribution μ_v^+ arises by projecting the mass from the halfspace above S into S in direction of $-v$, and similarly μ_v^- arises by projecting the mass from the lower halfspace along $+v$. The hyperplane S can be identified with $\mathbb{R}^{d-1}$, and thus the points $C(\mu_v^+), C(\mu_v^-) \in S$ are defined. A key claim Yao and Yao prove by topological means is that there exists v such that $C(\mu_v^+)$ and $C(\mu_v^-)$ coincide. For such a v, we set $C(\mu) = C(\mu_v^+)$ and

$$\Pi(\mu) = S \cup \{x + tv;\ t > 0,\ x \in \Pi(\mu_v^+)\} \cup \{x - tv;\ t > 0,\ x \in \Pi(\mu_v^-)\}$$

This finishes the definition of the partition; it turns out that $C(\mu)$ and $\Pi(\mu)$ are unique. Moreover, the partition defines 2^d regions, each containing 2^{-d} fraction of the total mass, and that each hyperplane avoids at least one of the regions.

All the above mentioned algorithms are based on theorems of a topological flavor, such as the ham-sandwich cut theorem or the Borsuk-Ulam theorem. They always use some *partition scheme*, which is a way of partitioning the space into several regions (usually bounded by hyperplanes), depending on the given point set. The crucial property of such a partition is that for any hyperplane, the regions intersected by it contain significantly fewer points than the whole set. The quality of a partition scheme is determined by the maximum number of regions which can be intersected by a hyperplane and by the maximum number of points contained in such regions. From a given partition scheme one then builds a partition tree in a more or less standard way. These algorithms usually have a polynomial but slow preprocessing (especially in higher dimensions).

Many researchers tried to find more efficient partition schemes, but it seemed that simple constructions could not be improved, and the analysis of more complicated constructions was too involved.

A randomized partition scheme. A significant breakthrough was made by Haussler and Welzl [HW87]. They were among the first (together with Clarkson and a few others) to bring probabilistic methods into computational geometry in a larger extent. They introduced an abstractly formulated range searching problem (specified as a set system (2)), and started building a theory of such abstract problems. The partition scheme used in their algorithm is also of a new type, with large and adjustable number of regions (the asymptotic query complexity improves with increasing of the number of regions). The partitioning scheme can be described quite simply (we do it in the plane): For a suitable large constant r, pick a random r-tuple of points of P, and draw all the $\binom{r}{2}$ lines determined by these points. These lines partition the plane into the desired regions. Haussler a Welzl proved that with high probability, the partition tree arising in this way guarantees query time $O(n^\alpha)$ with $\alpha = 2/3 + \delta$, where δ is a positive constant tending to 0 with $r \to \infty$. In dimension d one gets $\alpha = 1 - 1/[d(d-1)+1] + \delta$. The partition tree can be constructed in $O(n \log n)$ expected time by a randomized algorithm, later on also a deterministic construction of the same asymptotic complexity was found [Mat90].

Spanning trees with low crossing numbers. Query time close to $\sqrt{n}$ in the plane was first achieved by Welzl [Wel88]. He abandoned the principle of a partition tree constructed recursively using a partition scheme with a constant-bounded number of regions, and he replaced it by the idea of a *spanning tree with low crossing number*. This notion is worth explaining.

Consider a point set P in the plane and some spanning tree T on P, i.e. a graph-theoretic tree having P as its vertex set. We say that a halfplane R *crosses* an edge $\{u,v\}$ of T, if $|R \cap \{u,v\}| = 1$. The *crossing number of* T *with respect to the halfplane* R is the number of edges of T crossed by R, and the *crossing number of* T is the maximum of crossing numbers of T with respect to all halfplanes.

Let T be a spanning tree on P with a (possibly small) crossing number κ. For simplicity, let us moreover assume that T is a path (from known algorithms for constructing a spanning tree with low crossing number we can always obtain a path if we want one). For any given halfplane R, there are at most $\kappa + 1$ components of the induced subgraph of T on the set $P \cap R$. Each such component is an interval along the path T. Thus, having a suitable one-dimensional interval range searching data structure for intervals along T, we can answer the halfplane query by performing $\kappa + 1$ interval queries; of course, provided that we know the edges of T crossed by the halfplane R. This explains the significance of the spanning trees with low crossing number for halfplane range searching.

The notion of crossing number of a spanning tree can immediately be generalized to an arbitrary dimension d (with halfspaces in place of halfplanes), and it even makes sense for an arbitrary set system $(P, \mathcal{P})$ (a set $Q \in \mathcal{P}$ crosses an edge $\{u,v\}$ if $|\{u,v\} \cap Q| = 1$, and one continues as in the previous definition). Welzl proves a general existence result for spanning trees with low crossing number in this abstract formulation. The resulting crossing number is a function of a certain parameter of the considered set system, the so-called *dual shatter function*[4].

We cannot resist giving the definition of this important notion. Let $\mathcal{A} \subseteq \mathcal{P}$ be a subsystem of sets from $\mathcal{P}$. Let us call two points $x, y \in P$ $\mathcal{A}$-equivalent if $x \in Q \Leftrightarrow y \in Q$ for every $Q \in \mathcal{A}$. Then for an integer m we define the value of the dual shatter function $\pi^*_{\mathcal{P}}(m)$ as the maximum number of $\mathcal{A}$-equivalence classes on P, over all m-element $\mathcal{A} \subseteq \mathcal{P}$. For example, for the set system defined by triangles on an n-point set in the plane, the dual shatter function satisfies $\pi^*_{\mathcal{P}}(m) = O(m^2)$. This is because by drawing the contours of m triangles is the plane partitioned into $O(m^2)$ regions. In general, if $\pi_{\mathcal{P}}(m) \leq Cm^d$ for constants C, d and $1 \leq m \leq n$, Welzl's theorem guarantees the existence of a spanning tree with crossing number $O(n^{1-1/d} \log n)$ for $(P, \mathcal{P})$.

Returning to the case of halfspaces in $\mathbb{R}^d$, we get that Welzl's theorem implies the existence of a spanning tree with crossing number $O(n^{1-1/d} \log n)$ for any n-point set in $\mathbb{R}^d$. Later on Chazelle and Welzl [CW89] improved the bound on the crossing number in this geometric setting to $O(n^{1-1/d})$, which is asymptotically optimal[5] [6]. In this way they proved an almost optimal upper bound for the query complexity for the simplex range searching

[4]A *primal shatter function* also exists, but it is not significant for spanning trees with low crossing number.

[5]A recent result of Haussler [Hau91] implies that the bound without $\log n$ also holds in Welzl's abstract theorem.

[6]The problem of spanning tree with low crossing number in the plane belongs to the relatively few in computational geometry where someone worked on a more exact determining of the constant of proportionality. Welzl [Wel92] showed that the optimal crossing number lies between $\sqrt{n}$ and $(2.31 + o(1))\sqrt{n}$ in the worst case.

problem in the arithmetic model (which we have discussed in connection with the lower bounds). They even get a certain characterization of set systems admitting a sublinear query complexity in the arithmetic model. However, their method does not automatically give an efficient algorithm in the usual sense. We have already mentioned where the difficulty lies: We know that the boundary of the query simplex or halfspace only crosses few edges of the spanning tree, but it is not clear how to find these edges efficiently. In general, this problem seems almost as difficult as the original simplex range searching problem. Chazelle and Welzl managed to solve this additional complication in dimensions 2 and 3, thus obtaining the first almost optimal algorithms. Before leaving the spanning trees with low crossing number, let us mention that they found many other algorithmic applications (e.g., [Aga90], [EGH+89]) and even purely combinatorial applications, see [MWW93] or the contribution of J. Pach in [GPS91].

Further developments. The paper [MW92] stands somewhat aside from this "mainstream". It describes an essentially different method for answering halfplane range queries, based on a combinatorial lemma of Erdös and Szekeres, and it yields $O(\sqrt{n}\,\log n)$ query time with $O(n \log n)$ space. This method can also be used for triangles, but only if the point weights can be subtracted. There does not seem to be much hope for generalizing this method into higher dimensions. On the other hand, it is an easily implementable algorithm with small constants of proportionality, which cannot be said about most of the other algorithms.

Chazelle *et al.* [CSW92] discovered a simplex range searching algorithm for an arbitrary dimension, with $n^{1+\varepsilon}$ space and $O(n^{1-1/d+\varepsilon})$ query time, for an arbitrarily small positive constant ε. They thus approached the lower bound up to a factor of n^{ε} (for this reason they call the algorithm quasi-optimal). Their method returns to recursively constructed partition trees. For a single point set, however, they construct not one but several partition schemes at the same time, in such a way that for any halfspace R at least one of these schemes is sufficiently efficient. The construction then recurses for the point set in each of the regions of each partition scheme, which leads to a larger memory requirement, $n^{1+\varepsilon}$. As indicated above, if one partition scheme is good for all halfspaces, then it is also good for simplices, but if we must use several schemes it may happen that none of them is good for simplices. For this reason Chazelle *et al.* apply so-called multilevel data structures to handle the simplex queries (see section 6.3).

Simplicial partitions. The paper [Mat92b] returns to a single partition scheme, whose parameters are asymptotically optimal for all halfspaces, and thus also for simplices. The partition is constructed by a suitable generalization of Welzl's construction of spanning trees with a low crossing number [Wel88]. Let us describe these partition schemes. For simplicity, we only formulate the following definition for a point set in general position.

Let P be an n-point set in $\mathbb{R}^d$ in general position. A *simplicial partition for* P is a collection

$$\Pi = \{(P_1, \Delta_1), \ldots, (P_m, \Delta_m)\},$$

where the P_i are disjoint subsets of P (called the *classes*) forming a partition of P, and each Δ_i is a d-dimensional simplex containing the set P_i.

Let us remark that the simplices need not be disjoint and the simplex Δ_i

may also contain other points of P than those of P_i, see fig. 3. Although it is not clear why this should make the construction of a partition scheme any easier, no partition schemes of a comparable efficiency as in Theorem 4.1 below but with disjoint regions are known.

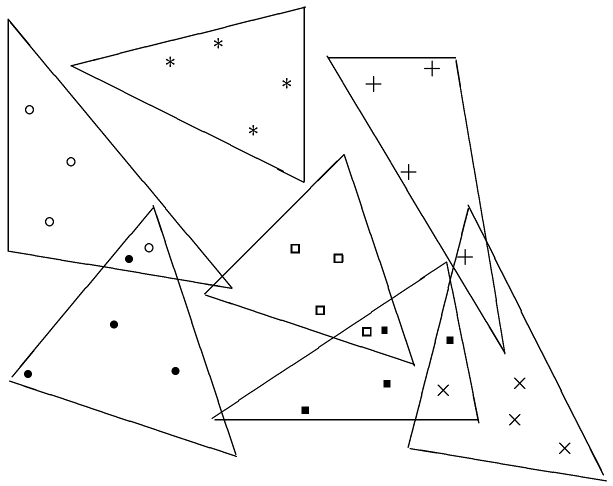


Figure 3: A simplicial partition (points of different classes are marked by different symbols).

If h is a hyperplane and Δ a simplex, we say that h *crosses* Δ if it intersects it (for simplices of lower dimension than d appearing when we deal sets in degenerate positions the definition of crossing is somewhat more complicated). Further we define the *crossing number of the simplicial partition* Π *with respect to* h as the number of simplices of Π crossed by h, and the *crossing number of* Π is the maximum of the crossing numbers with respect to all hyperplanes.

The main theorem of [Mat92b] is as follows:

Theorem 4.1 *Let P be an n-point set in $\mathbb{R}^d$ ($d \geq 2$), r a parameter, $1 < r \leq n/2$. Then there exists a simplicial partition for P satisfying $n/r \leq |P_i| \leq 2n/r$ for every class P_i (thus with $O(r)$ classes) and with crossing number $\kappa = O(r^{1-1/d})$.*

The crossing number in this result is asymptotically optimal. The paper [Mat92b] also gives an algorithm for constructing such a simplicial partition, with $O(n \log r)$ running time for sufficiently small r ($r \leq n^\beta$ for a certain small constant $\beta = \beta(d) > 0$).

Construction of simplicial partitions. The construction is based on an application of *cuttings* (see the box on page 23)[7]. The first step

[7]The use cuttings replaces another argument from the otherwise similar Welzl's construction of spanning trees with low crossing number in [Wel88]. The application of cuttings makes use of more of the geometric properties. This is not accidental, as Alon *et al.* [AHW87] showed

is the choice of a suitable finite "test set" H of hyperplanes, such that whenever the crossing number of a simplicial partition Π with respect to every $h \in H$ is bounded by some number κ, then the crossing number of Π with respect to *any* hyperplane is $O(\kappa + r^{1-1/d})$. The test set H has size $O(r)$ and it is constructed using cuttings in the dual space.

With such a test set H at our disposal, the relation of cuttings to a small crossing number of a simplicial partition is the following: If Ξ is a $(1/t)$-cutting for H, then the total number of incidences among the simplices of Ξ and the hyperplanes of H is at most $|\Xi||H|/t$ (since any simplex of Ξ has at most $|H|/t$ incidences), and hence the *average* number of simplices of Ξ crossed by a hyperplane of H is $|\Xi|/t$. The total number of simplices in Ξ is $r = O(t^d)$, and hence the crossing number with respect to an average hyperplane of H is $O(t^{d-1}) = O(r^{1-1/d})$. The cutting itself thus gives a simplicial partition whose crossing number with respect to an average hyperplane of H is asymptotically optimal. There are two shortcomings, however: First, the points of P need not be equally distributed among the simplices of Ξ, and second, although the average crossing number is optimal, there might be some hyperplanes with bad crossing numbers.

These are fixed by an incremental construction of Π using the so-called *reweighting strategy*, imitating Welzl's construction[8]. The simplicial partition Π is constructed step by step, one pair (P_i, Δ_i) at a time. Suppose that (P_1, Δ_1) thru (P_i, Δ_i) have already been constructed, $|P_j| = \lceil n/r \rceil$, $j = 1, \ldots, i$. The set $\tilde{P}_i = P \setminus (P_1 \cup \ldots \cup P_i)$ of points not yet covered has $n_i = n - i\lceil n/r \rceil$ points. For a hyperplane $h \in H$, let $\kappa_i(h)$ be the number of simplices among $\Delta_1, \ldots, \Delta_i$ crossed by h, and assign a weight $w_i(h) = 2^{\kappa_i(h)}$ to every $h \in H$. We let Ξ_i be a $(1/t)$-cutting for H, w_i, that is, the sum of the w_i weights of the hyperplanes intersecting any simplex Δ of Ξ_i is at most $(1/t)$ of the total w_i weight of all hyperplanes. We choose t as large as possible but in such a way that the total number of simplices of Ξ_i does not exceed $r(n_i/n)$, and therefore we can find a simplex of Ξ_i containing at least n/r points from $\tilde{P}_i$. Such a simplex then becomes Δ_{i+1}, and the set P_{i+1} is chosen as some $\lceil n/r \rceil$ points of $\tilde{P}_i$ among those contained in Δ_{i+1}.

This finishes the description of the construction, and one now has to prove that the resulting simplicial partition indeed has the required crossing number. One bounds the increment of the total weight of all hyperplanes of H caused by adding one new simplex Δ_{i+1} to the simplicial partition, this yields an estimate on the total weight of all hyperplanes after the last stage, and from this bound it follows that no hyperplane can give too big a crossing number; we refer e.g., to [Mat92b] for the detailed calculation.

By a recursive application of the simplicial partition from Theorem 4.1, a partition tree is created in a standard way. For a given set P we find a simplicial partition Π (with a suitable choice of the parameter r) and we store its simplices as well as the total weights of points in the classes of Π in the root of the partition tree. Each class of the simplicial partition corresponds

that for certain set systems there exists no efficient partition scheme with a constant-bounded number of regions, although the dual shatter function is small and a spanning tree with low crossing number thus exists.

[8]Probably it will be no surprise to the reader that a similar strategy was also used by Erdös much earlier.

Cuttings

A *cutting* is a finite set of closed simplices (here a simplex means an intersection of $d+1$ halfspaces, hence also unbounded "simplices" are allowed) with disjoint interiors covering $\mathbb{R}^d$. Let H be a set of n hyperplanes in $\mathbb{R}^d$ and r a parameter, $1 < r \leq n$. A cutting Ξ is called a *$(1/r)$-cutting for H* provided that the interior of no simplex of Ξ is intersected by more than n/r hyperplanes of H. Sometimes it is useful to consider also a weighted version, where a nonnegative weight function $w : H \to \mathbb{R}$ is given, and the sum of weights of hyperplanes intersecting any simplex of a $(1/r)$-cutting should not exceed $1/r$ of the total weight of all hyperplanes of H.

The notion of a cutting is a basis of many geometric algorithms of the "divide and conquer" type dealing with hyperplanes. A computational problem concerning a set H of hyperplanes is partitioned into subproblems defined by the simplices of a $(1/r)$-cutting, each involving $r\times$ fewer hyperplanes than the original problem. This dividing strategy was developed in the works of Clarkson, Haussler and Welzl and others. The cuttings are considered explicitly in the papers of Chazelle and Friedman [CF90] and of the author [Mat90], the name and the current definition were given in [Mat91c].

For applications it is important that the number of simplices in a cutting is as small as possible. Chazelle and Friedman [CF90] showed that for every H and r there exists a $(1/r)$-cutting consisting of $O(r^d)$ simplices. This result is asymptotically optimal, and it applies also to the weighted version. Efficient computation of cuttings is considered in [CF90], [Mat91c], [Mat91b], [Cha93].

to one subtree of the root, where the construction is used recursively for the points in the corresponding classes. When answering a query with a halfspace R, we process the simplices lying completely inside R or completely outside R directly in the current node, and for the simplices crossing the boundary of R we recursively descend into the appropriate subtrees. For the query time, we thus obtain the recurrence

$$T(n) \leq f(r) + \kappa T(2n/r)\,, \tag{7}$$

where $\kappa = O(r^{1-1/d})$ is the crossing number of the simplicial partitions and $f(r)$ is the cost of the auxiliary computations in one node of the partition tree, i.e. finding the crossing simplices and computing the total weight of points in simplices lying completely inside the query halfspace R. If we simply inspect all the simplices of the simplicial partition and test their position with respect to R, we have $f(r) = O(r)$.

We must now choose a suitable value of the parameter r. If we take a large constant for r (similarly as it was done in previous partition tree constructions), (7) yields $T(n) = O(n^{1-1/d+\varepsilon})$, where ε tends to 0 as $r \to \infty$. We can, however, also choose r as some function of n (more exactly, of the number of points in the current node of the partition tree), e.g., a small power of n. Our interest is even to take r as large as possible: A larger r means a more branching, thus more shallow partition tree, and since we lose a constant factor in the query answering efficiency at every level of the partition tree, the more shallow tree the

Geometric duality

In the plane, duality is a mapping assigning points to lines and lines to points. There are various versions of duality considered in the literature; for most applications, the differences among them are unessential. For definiteness, we work with the duality $\mathcal{D}$ defined as follows: A point $p = (a, b)$ has a dual line $\mathcal{D}(p) = \ell = \{(x, y);\ y = 2ax - b\}$, and conversely the image $\mathcal{D}(\ell)$ of such a line ℓ is the point $p = (a, b)$. This defines the image of every point and of every line except for vertical ones (vertical lines correspond to points in the infinity of the projective plane).

It is easy to check that for any point q and a non-vertical line h, q lies on h iff the point $\mathcal{D}(h)$ lies on the line $\mathcal{D}(q)$. Moreover, q lies above h iff $\mathcal{D}(h)$ lies above $\mathcal{D}(q)$. These properties of geometric duality often allow a much more intuitive formulation of geometric problems by passing to dual objects.

Similarly is duality defined between the points and hyperplanes in the d-dimensional space. The duality $\mathcal{D}$ maps the point $p = (a_1, a_2, \ldots, a_d)$ to the hyperplane $\mathcal{D}(p) = \{(x_1, x_2, \ldots, x_d);\ x_d = 2a_1x_1 + 2a_2x_2 + \cdots + 2a_{d-1}x_{d-1} - a_d\}$ and conversely. An analog of the above mentioned properties for the planar case holds (duality preserves incidences and above/below relation). For example, if we want to count points lying below a hyperplane, the dual problem is counting the number of hyperplanes above a given point.

Further information about geometric duality can be found in the book [Ede87].

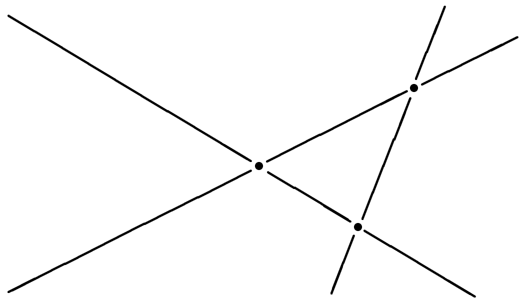
better. On the other hand, we must not overdo it, since otherwise the term $f(r)$ in the recurrence (7) becomes an obstacle. This means that if the number of simplices of the simplicial partition were, say, comparable to n, then already the auxiliary computations (finding the crossing simplices, summing the weights) would take more time than we can afford. By a suitable compromise, where r is a sufficiently small power of n, we obtain query time $O(n^{1-1/d} \log^c n)$, where c is some constant (which we better don't try to estimate). In this situation, the preprocessing time is $O(n \log n)$.

In order to use a more shallow partition tree without $f(r)$ getting too large, we can start building various auxiliary data structures for the above mentioned computations in each node of the partition tree. In [Mat92b] this method is pursued, and with much more complicated data structures one gets an asymptotically better query time $O(n^{1-1/d}(\log \log n)^c)$.

In [Mat92c] the author succeeded in reducing the query complexity for simplex range searching with linear space to $O(n^{1-1/d})$. The method is technically somewhat complicated and it is based on similar ideas as mentioned above plus an application of so-called hierarchical cuttings from Chazelle's work [Cha93]. This result is optimal in dimension 2 and most likely also in higher dimensions, see section 3. However, the preprocessing time is $O(n^{1+\varepsilon})$, while an ideal one would be $O(n \log n)$. Thus, there is still room for improvement, and there is also the challenge of finding a simpler optimal algorithm.

Hyperplane arrangements

Let us consider a finite set H of lines in the plane. These lines divide the plane into convex sets (called *cells*; sometimes also the word *faces* is used) of various dimension, see the figure: The cells of dimensions 0, or 0-cells for short, are the intersections of the lines of H, and we call them *vertices*. If we remove all vertices lying on a line $h \in H$, the line h is split into two open semiinfinite rays and a finite number of open segments. These segments and rays form the 1-cells or *edges*. Finally, by removing all the lines of H is the plane partitioned into open convex polygons (also unbounded ones), which are the 2-cells. Similarly a finite set H of hyperplanes in $\mathbb{R}^d$ defines a decomposition of $\mathbb{R}^d$ into cells of dimensions $0, 1, \ldots, d$.



This decomposition is a cell complex in the sense of algebraic topology, and it is called the *arrangement of* H. For a fixed d, the total number of cells of all dimensions is bounded by $O(|H|^d)$. This bound is best possible for H in general position; in this case it is not too difficult to obtain an exact formula for the number of cells of each dimension. Hyperplane arrangements are one of basic objects of study in computational geometry, see e.g. [Ede87] for more information.

5 Algorithms with logarithmic query time

Halfspace range searching. A logarithmic query time is essentially achieved by precomputing all possible answers and storing them in a suitable data structure. It is more natural to consider an equivalent dual version of the problem (see the box on page 24 for the definition of duality):

Problem 5.1 *Let H be a set of n hyperplanes in $\mathbb{R}^d$ equipped with weights. For a query point q, find the sum of weights of the hyperplanes lying above*[9] *q.*

The answer obviously remains constant for points q lying in the same cell of the arrangement of H (see the box on page 25). We can thus construct the arrangement, compute and store the weight for each cell, and then it suffices to detect the cell containing a query point and return the precomputed answer.

[9]This re-formulation follows by our special choice of the duality transform. In general, one can consider the problem of finding the total weight of the hyperplanes separating a query point from a fixed point o. Such problems can be converted to each other by projective transforms.

Point location

The *point location* problem is the following: Given a subdivision of $\mathbb{R}^d$ into convex cells, construct a data structure which can quickly determine the cell of the subdivision containing a query point.

Let m denote the combinatorial complexity of the subdivision (the total number of cells of all dimensions). In the plane, several optimal data structures are known, with $O(m)$ space and preprocessing time and with $O(\log n)$ query time, see e.g., [Sei91a] for a modern randomized method or the references to older works in [Ede87]. For dimension 3, a method with a polylogarithmic query time and nearly linear space was described in [PT92], but for dimension 4 and higher no data structure with a comparable efficiency has been found for general subdivisions.

A hyperplane arrangement is a very special subdivision of space, and here efficient point location methods are known in an arbitrary dimension. The first such algorithm, due to Clarkson [Cla87], requires $O(n^{d+\varepsilon})$ space and preprocessing time and it performs the location in $O(\log n)$ time. By various technical improvements of the same basic idea, Chazelle and Friedman [CF] obtained an algorithm with the same query time and with optimal $O(n^d)$ space. A conceptually much simpler method was discovered by Chazelle [Cha93], with the same space and query time and also with $O(n^d)$ preprocessing. The algorithms from these three papers are based on hierarchies of progressively refined cuttings, similarly as the algorithm described in section 5.

The preprocessing can be done in $O(n^d)$ time, see e.g. [Ede87], [CSW92]. For dimensions 2 and 3, efficient methods are known for fast point location in an arbitrary subdivision of the space into convex cells. For higher dimensions, special methods have been developed for point location in hyperplane arrangements (see the box on page 26). This gives a halfspace range searching method with $O(n^d)$ space and preprocessing time and $O(\log n)$ query time. The space and preprocessing time can even be reduced somewhat (approximately by a factor of $\log^d n$) with the same query time, as was shown in [Mat92c].

Here we explain a simple algorithm formulated directly for the problem 5.1; point location algorithms for hyperplane arrangements are similar. It is a nice application of the notion of cuttings, see the box on page 23.

The data structure is, as usual, a tree, and each node corresponds to some subset of hyperplanes of H. The construction starts in the root, which corresponds to the whole H. We choose a large constant r and we construct a $(1/r)$-cutting $\Xi = \{\Delta_1, \ldots, \Delta_s\}$ for H. By theorems on the existence of cuttings, we can assume $s \leq Cr^d$, where C is an absolute constant independent of r. For every simplex $\Delta_i \in \Xi$ let H_i be the set of hyperplanes of H intersecting its interior, and let w_i be the total weight of hyperplanes lying completely above the interior of Δ_i. The cutting Ξ and the weights w_i are stored in the root, and for every $i = 1, 2, \ldots, s$ we build a subtree corresponding to H_i by a recursive application of the same procedure. The recursion terminates as soon as we reach sets of hyperplanes smaller than a suitably chosen constant. Then, instead of a further recursion, we store the corresponding set of hyperplanes in a leaf of the constructed tree.

The total weight of hyperplanes above a query point q is determined as follows. For the cutting stored in the root, we find the simplex Δ_i containing q (in constant time, since r was a constant). This gives us the total weight w_i of the hyperplanes from $H \setminus H_i$ lying above q, and it remains to determine the total weight of hyperplanes of H_i above q. This is computed by a recursive application of the same method on the subtree corresponding to H_i. In this way we reach a leaf of the tree after $O(\log n)$ steps and we process the several hyperplanes stored there directly.

Let us look at the space requirements of this data structure. The space $S(n)$ needed for n hyperplanes can be bounded using the recurrence relation

$$S(n) \leq Cr^d + Cr^d S(n/r) .$$

The resulting bound is $S(n) = O(n^{d+\varepsilon})$, where $\varepsilon > 0$ can be made arbitrarily small by choosing r large enough.

> The above described hierarchy of progressively refining cuttings is somewhat wasteful. On each level we lose one additional constant factor, which is reflected in the total space requirement. This effect can be restricted by making r larger, but then the searching in the cutting in a single node becomes expensive. Chazelle [Cha93] succeeded in overcoming this: his method produces a hierarchy of progressively refining cuttings, where the finest cutting on the last level has a total size $O(n^d)$ only, instead of $n^{d+\varepsilon}$. Chazelle's method is also explained in [Mat93b] in a somewhat simplified form.

Simplex range searching. Since about n^6 subsets of an n point set in the plane can be defined by a triangle, storing all possible answers for the triangle range searching problem requires at least about n^6 space. Such an approach was used by Edelsbrunner *et al.* [EKM82], and their solution requires space of the order n^7. A significant reduction of the space requirement, to $O(n^{2+\varepsilon})$, was achieved by Cole and Yap [CY85]; their method is based on an idea resembling multilevel data structures (see section 6.3). They also give an $O(n^10)$ space data structure for 3-dimensional simplex range searching. Another paper considering the reporting in the planar case is [PY86]. A general method for simplex range searching with storage approximately n^d for arbitrary point weights and arbitrary dimension was found relatively recently by Chazelle *et al.*, more about it in section 6.3. Space and preprocessing time are $O(n^{d+\varepsilon})$ and query time is $O(\log^{d+1} n)$. The space and preprocessing time were improved in [Mat92c], using Chazelle's hierarchy of cuttings [Cha93].

6 Extensions and examples of applications

6.1 Halfspace range reporting and other special situations

The halfspace range searching problem with only emptiness queries or reporting queries turned out to have more efficient solutions than the general simplex range searching problem. It seems that the actual query complexity for halfspace range reporting with space at most m might be approximately

$$\frac{n}{m^{1/\lfloor d/2\rfloor}} + k\,, \tag{8}$$

where k denotes the number of points in the query halfspace. In particular, in dimensions 2 and 3 algorithms exist with almost linear storage and $O(\log n + k)$ query time, see [CGL85], [CP86], [AHL90]. For a higher dimension, Clarkson [Cla88a] found an algorithm with space and preprocessing $O(n^{\lfloor d/2\rfloor+\varepsilon})$ and query time also $O(\log n + k)$. This result was complemented in [Mat92a] by an algorithm with $O(n \log\log n)$ space, $O(n\log n)$ preprocessing time and $O(n^{1-1/\lfloor d/2\rfloor}\log^c n + k)$, query time (the method is quite similar to the one for simplex range searching from [Mat92b], discussed in the end of section 4). By combining these two methods, the complexity given by (8) can almost be achieved in the whole spectrum of memory requirements. Further small improvements were described by Schwarzkopf [Sch92]. No lower bounds are known.

> Very roughly speaking, we can say that the exponent d in the formula (1) expressing the complexity of simplex range searching originates in the fact that the combinatorial complexity of an arrangement of n hyperplanes in $\mathbb{R}^d$ is of the order n^d. Similarly the exponent $\lfloor d/2\rfloor$ in (8) is closely related to the worst-case complexity of a polytope defined as an intersection of n halfspaces in $\mathbb{R}^d$, which is of the order $n^{\lfloor d/2\rfloor}$, see e.g., [Ede87]. When we should quickly decide the emptiness of a halfspace or — in the dual form — quickly test whether a given point lies above all hyperplanes of a given set H, we essentially deal with a problem of point location in a convex polytope, namely in the upper unbounded cell of the arrangement of H.

In the dual version of halfspace range reporting, we essentially restrict our attention to a single cell of the arrangement of the given hyperplanes, and as we saw, this yields a considerable improvement. Another such special situation is when our point set lies on a fixed lower dimensional algebraic variety of bounded degree, or if all hyperplanes bounding the query ranges are tangent to such a variety; then the range searching results can sometimes be also improved (the restricted point set case helps for the linear-space case, the restricted ranges for the large space case, see e.g., [AM92] for a discussion). These improvements use a combinatorial result, a zone theorem of Aronov *et al.* [APS93b]. The situations with points on a surface is by no means rare — it arises e.g., when dealing with lines in 3-dimensional space, see e.g., [CEGS89].

6.2 Dynamization

Until now we have considered static range searching problems, where the point set is given once and for all. In many applications we need to insert new points or delete old ones from time to time, and it is thus advantageous if we can only modify the data structure suitably, instead of rebuilding it from scratch after every change. This problem is called the *dynamization* of a data structure. For a dynamic data structure storing n points we probably cannot expect a better time for one modification than a $1/n$ fraction of the time needed for building the whole (static) structure anew. For the simplex and halfspace range searching problems dynamic structures with this efficiency (up to small factors, typically of the order n^ε) are known, see [SO90], [AS93], [Mat92b], [AM91]; see also [BS80], [Ove83], [Meh84] for dynamization in general. It would still be interesting to get rid of the n^ε extra factors for, say, the dynamic halfspace emptiness data structures.

Dynamizing the simplex range searching data structures from [CSW92] or [Mat92b] is quite straightforward, and it is equally easy to dynamize the linear space halfspace reporting data structure from [Mat92a], see [AM91]. A dynamic version of Clarkson's halfspace reporting data structure is more complicated. The source of difficulties here is the fact that the maximal asymptotic complexity of the convex hull of n points in an odd dimension d and the preceding even dimension $d-1$ are the same. The change of the convex hull caused by adding or deleting a single point is proportional to the complexity of a certain convex hull in dimension $d-1$. For instance, consider the convex hull of n points in $\mathbb{R}^3$. This is a convex polyhedron with $O(n)$ vertices, edges and facets, and some vertices can have degrees close to n. If we keep inserting and deleting such vertices, it seems impossible to maintain an explicit representation of the convex hull (e.g., as a planar graph) in time substantially smaller than n for one operation.

For these reasons, among others, several authors investigated dynamic data structures under the assumption that the update sequence is random in a suitably defined sense and obtained very good update times for this case, see [Mul91d], [Mul91c], [Sch91]. In [AM91] a dynamic algorithm was found, which is efficient also in the worst case, for an arbitrary update sequence[10]. This algorithm does not maintain an explicit representation of the convex hull, rather it works with certain implicit representation. Nevertheless, this data structure in connection with other techniques (see section 6.5) supports fast answering of various queries concerning the convex hull, such as deciding if a query point lies inside or outside the current convex hull, computing a tangent hyperplane to the convex hull and containing a given line etc.

6.3 Multilevel data structures

We first explain the idea of multilevel data structures on an example, and then we introduce some abstract notions for their description.

Let $S = \{s_1, \ldots, s_n\}$ be a set of segments in the plane. We want to construct a data structure which quickly computes the number of segments of S intersected by a query line h. We permit roughly linear space for the data structure. Let a_i, b_i be the endpoints of the segment s_i. A denotes the set of all a_i and B the set of all b_i. We consider computing the number of s_i such that a_i is above h and b_i is below h. The opposite case is solved symmetrically, and the case of vertical query lines can be treated separately.

Let us consider some partition tree for the set A; for definiteness, let it be the partition tree based on simplicial partitions from Theorem 4.1, with r being a large constant. Using such a partition tree we can determine, in roughly $\sqrt{n}$

[10]In the amortized sense, i.e. an individual update time can sometimes be large but the average over a sequence of n updates is small.

time, the number of points of A in the halfplane R above h. This does not solve our problem yet, but we look more closely how the answer is obtained from the partition tree. The weight of every point from $A \cap R$ is accounted for in some of the visited nodes of the tree. In each such node, we find the simplices of the corresponding simplicial partition lying completely inside R, and the weights of their respective classes are accounted for as wholes.

For each node of the partition tree, let us call the classes of the simplicial partition stored there the *canonical sets*. We see that the partition tree provides a partition of the set $A \cap R$ into roughly $\sqrt{n}$ canonical sets of various sizes. The total number of canonical sets in the partition tree is $O(n)$, and the sum of their sizes is easily estimated to $O(n \log n)$.

For our problem with segments, we augment the partition tree for the set A as follows: For every canonical set $M \subseteq A$ we create a partition tree for the set $M' = \{b_i;\ a_i \in M\}$, and we store it with the corresponding node of the partition tree for A (the partition tree for A is called the *primary* or *first level* one, the trees for the sets M' are called the *secondary* or *second level* ones).

How does one apply the resulting data structure to the segment counting problem? First we express the set $A \cap R$ as a disjoint union of certain canonical subsets $M_1, \ldots, M_m$, and then for each such M_i we use the appropriate secondary partition tree to count the points of M_i' lying below the line h. Adding these counts together over all canonical sets M_i, we obtain the desired number of segments s_i with a_i above h and b_i below h.

On the first sight it seems that the described two-level data structure should be much less efficient than the original partition trees used for its construction. By a simple calculation we can convince ourselves that this is not the case, and that the required space is $O(n \log n)$ only (this is because of the total size of the canonical subsets) and the query time remains still close to $\sqrt{n}$. This is because there are only few large canonical sets in the decomposition of $A \cap R$, and the computation on the second level is fast for small canonical sets.

The principle used in the above example is quite universal. Usually it is applicable whenever the query is a conjunction of several conditions (or, geometrically, an intersection of several regions) and for each condition (region) we already have a suitable efficient data structure.

The idea of multilevel data structures appears in Bentley's data structures for orthogonal range searching [Ben80]. In the context of partition trees such a construction was introduced by Dobkin and Edelsbrunner [DE87]. Recently it has been used quite frequently, see e.g., the papers [OSS90], [CSW92], [Mat92b], [AS93] and many others. The descriptions of such data structures often look complicated, especially if there are more levels. In [Mat92c] an abstract framework is proposed for description and analysis of such data structures.

Let us restate the principle of multilevel data structures in a somewhat abstract setting and give two more examples. First we introduce the notion of decomposition schemes. Consider a range searching problem in an abstract form, such as we mentioned in section 3. It is given by a set system $(P, \mathcal{P})$, usually defined as in equation (2). From an abstract point of view, in most range searching algorithms one proceeds as follows: Another system $\mathcal{C}(P)$ of subsets of P, the so-called *canonical subsets*, is defined, and a rule is given how to express each range $R \in \mathcal{P}$ as a disjoint union of canonical sets from $\mathcal{C}(P)$. Let us call this pair, the set system $\mathcal{C}(P)$ plus the rule how to decompose sets of $\mathcal{P}$, a *decomposition scheme* for $(P, \mathcal{P})$,

Such a decomposition scheme can be turned into a range searching data structure (provided that the decompositions can be found efficiently), simply by storing the total weight of points for each canonical subset, and this is how most of the range searching data structures work, although the canonical subsets are not mentioned explicitly.

For partition trees as described in section 4, canonical subsets are the point sets lying in regions of the partition schemes at the nodes of the partition tree. For the one-dimensional data structure for range searching with intervals in Example 2.2, the canonical subsets are just canonical intervals, and each interval can be partitioned into $O(\log n)$ canonical intervals. In the data structure for halfspace range searching with logarithmic query time explained in section 5, each node of the tree defines canonical subsets corresponding to simplices of the cutting stored in it. If the node corresponds to a subset G of hyperplanes and Δ_i is one of the simplices in its cutting, then the corresponding canonical subset is formed by all the hyperplanes of G lying completely above Δ_i. Any range, which is the set of hyperplanes lying above a query point, can be expressed as a disjoint union of $O(\log n)$ canonical sets.

In our subsequent development, we need that a decomposition scheme can operate not only on the system $(P, \mathcal{P})$ itself, but also on systems induced by subsets of P. This means that for a subset $P' \subset P$, we also have a decomposition scheme for $(P', \{P' \cap R;\ R \in \mathcal{P}\})$. This is usually trivially true for geometric problems, since if we can build a decomposition scheme for a set we can also build one for its subset. Such a decomposition scheme will be called a *hereditary* one.

Let now P be a set of basic objects (not necessarily points, in the initial example of this section these would be the segments). Let $\mathcal{P}_1, \mathcal{P}_2$ be two set systems on P, and suppose that we have a decomposition scheme $\mathcal{D}_1$ for $(P, \mathcal{P}_1)$ and a hereditary decomposition scheme $\mathcal{D}_2$ for $(P, \mathcal{P}_2)$. In the example with segments, $\mathcal{P}_1$ would be all subsets of the form $\{s_i \in P;\ a_i \in H\}$ for some halfplane H, and similarly $\mathcal{P}_2 = \{s_i \in P;\ b_i \in H\}$ for some halfplane H. The decomposition scheme for $(P, \mathcal{P}_1)$ is given by building the partition tree on the set A of the a_i-endpoints, and the decomposition scheme for $(P, \mathcal{P}_2)$ by a partition tree on the b_i-endpoints (thus formally we work with sets of segments, although the partition trees actually deal with their endpoints).

Our goal is to derive a decomposition scheme for a more complicated system $(P, \mathcal{P})$, where $\mathcal{P}$ consists of all sets of the form $R_1 \cap R_2$ with $R_1 \in \mathcal{P}_1$ and $R_2 \in \mathcal{P}_2$. In the example with segments, the ranges we are interested in are indeed of this form, namely the sets of segments whose a_i endpoints lie in an upper halfplane H_1 and b_i endpoints in a lower halfplane H_2; in fact, H_1 and H_2 are complementary halfplanes, but we take no advantage of this. We define what is called the *composition* of the decomposition schemes $\mathcal{D}_1$ and $\mathcal{D}_2$, denoted by $\mathcal{D} = \mathcal{D}_1 \circ \mathcal{D}_2$. To decompose a set $R = R_1 \cap R_2 \in \mathcal{P}$, we first decompose, using $\mathcal{D}_1$, the set $R_1 \in \mathcal{P}_1$ into canonical subsets $C_1, C_2, \ldots, C_m \in \mathcal{C}_1(P)$. Then for each C_i, we consider the decomposition scheme $\mathcal{D}_2$ operating on the subsystem of $\mathcal{P}_2$ induced by C_i, and decompose the set $C_i \cap R_2$ into canonical sets $C_{i1}, C_{i2}, \ldots, C_{ik_i} \in \mathcal{C}_2(C_i)$. The union of these collections for all

$i = 1, 2, \ldots, m$ gives us a decomposition of $R_1 \cap R_2$. The canonical sets $\mathcal{C}$ in the resulting decomposition scheme $\mathcal{D}$ will thus be all canonical sets from $\mathcal{C}_2(C)$ for some $C \in \mathcal{C}_1(P)$.

What we did in the example with segments can thus be rephrased as first composing the two decomposition schemes and then turning the resulting decomposition scheme into a range searching data structure by precomputing the weights of all canonical sets. Knowing the parameters of both the decomposition schemes, it is a routine calculation to derive the parameters of the composed scheme. The total number of canonical subsets determines the space requirements of the data structure, and the maximal number of canonical sets appearing in a decomposition of a range is related to query time.

As one basic example, let us consider orthogonal range searching. Let P be an n-point set in $\mathbb{R}^d$. We let $\mathcal{P}_i$ be the set system defined on P by intervals on the x_i-axis. For each $(P, \mathcal{P}_i)$ we have the decomposition scheme $\mathcal{D}_i$ with canonical sets being the canonical intervals along the x_i-axis (see Example 2.2). A range of the form $R_1 \cap R_2 \cap \ldots \cap R_d$ with $R_i \in \mathcal{P}_i$ corresponds to a subset of P lying in an axis-parallel box. The d-wise composition $\mathcal{D} = \mathcal{D}_1 \circ \mathcal{D}_2 \circ \ldots \circ \mathcal{D}_d$ is thus a decomposition scheme for the set system defined by axis-parallel boxes. By turning this decomposition scheme into a range searching data structure, we recover an abstract version of Bentley's *range trees* [Ben80].

Another important example is the application of multilevel data structures for the already mentioned simplex range searching with polylogarithmic query time from [CSW92]. The halfspace range searching method described in section 5 yields a decomposition scheme $\mathcal{D}_0$ which partitions the point set in a query halfspace into $O(\log n)$ canonical subsets (we stated this above in the dual form). Since a simplex is an intersection of $d+1$ halfspaces, we can obtain a simplex decomposition scheme by a $(d+1)$-wise composition of $\mathcal{D}$ with itself, and this gives the desired simplex range searching data structure. Easy calculations show that the resulting data structure has query time $O(\log^d n)$ and occupies memory $O(n^{d+\varepsilon})$.

Let us conclude with few remarks. Asymptotically, one usually loses a factor of n^ε in space and query time with each level of a multilevel data structure (compared to the efficiency of the data structures used for the levels). In [Mat92c], decomposition schemes are described which allow one to build multilevel data structures while losing a polylogarithmic factor per level only. On the other hand, it seems that a practical efficiency of a data structure will be lost quite quickly with an increasing number of levels (again, this is not substantiated by any implementation experience). Thus, it makes a big difference if we use a simplex decomposition scheme directly or if we create it by composing halfspace decomposition schemes $d+1$ times.

We should also point out that while most of the simplex range searching algorithms provide decomposition schemes in the above explained sense, the halfspace emptiness and halfspace reporting algorithms discussed in section 6.1 do not provide a halfspace decomposition scheme (in some sense, they can only decompose very "shallow" halfspaces, i.e. ones which contain few points of P only). Thus, if we want to build a data structure for testing the emptiness of complicated ranges, where one of the defining conditions is a halfspace, we may only use the halfspace emptiness data structure in the lowest (last) level of a multilevel data structure.

6.4 Searching with more general ranges

In the previous sections, we considered range searching with ranges bounded by hyperplanes. Many applications naturally lead to searching in ranges with nonlinear, curved boundaries. How general query ranges should be considered? It seems that nontrivial results can be expected mainly for ranges determined by a small (constant) number of real parameters. The most important such case are subsets of $\mathbb{R}^d$ defined by a conjunction of at most k polynomial inequalities of maximum degree D, where d, k and D are constants. We call such sets *elementary cells* for short[11].

Range searching problems with various kinds of elementary cells include many specific problems motivated by applications. Some of the results mentioned below can be formulated for still somewhat more general ranges, but writing out all the necessary assumptions would become somewhat lengthy.

Perhaps the simplest among nonlinear ranges are circular disks in the plane and balls in higher dimensions. The corresponding range searching problem arises when we are interested in points lying in at most a given distance from a given point. Notice that balls do not appear in this second, "application oriented" formulation directly. This is quite typical, as range searching problems with nonlinear ranges are usually obtained by a suitable re-formulation of the original specification of the problem. For instance, if we are asking for points whose distance from a given segment does not exceed a given number, an equivalent formulation is range searching with ranges of the shape of a racecourse. If we want to detect points which see a segment s_1 under a larger angle than a segment s_2, we get a range reporting query with a rather complicated range defined by inequalities of degree 6, etc.

Ball range searching and related problems, usually referred to as *proximity problems*, have a rich literature, see e.g., [CCPY86], [AHL90], [CW89] and others. Range searching with other ranges was paid less attention to. An important earlier work considering very general geometric range searching problems is [YY85]; it contains several important ideas which were further developed and found many applications later on. A recent work considering problem of range searching in elementary cells is [AM92].

To some extent, methods developed for simplex and halfspace range searching can also be applied for searching with elementary cells. We now outline the results. We concentrate on ranges defined by a *single* polynomial inequality. Ranges described by a conjunction of several inequalities can be handled similarly, sometimes by applying multilevel data structures, where each level processes one of the inequalities.

A disk $C = C(a_1, a_2, a_3) \subset \mathbb{R}^2$ with center (a_1, a_2) and radius a_3 is described by the inequality $(x_1 - a_1)^2 + (x_2 - a_2)^2 \leq a_3^2$. More generally, we consider ranges of the form

$$R_f(a) = \{x \in \mathbb{R}^d;\ f(x, a) \geq 0\},$$

where f is a fixed polynomial in $d + p$ variables $x_1, \ldots, x_d, a_1, \ldots, a_p$. The polynomial f specifies the type of the considered ranges (disks, conics, cylinders,

[11]In the literature, the term *Tarski cells* is used for a similar but somewhat more general notion.

$\dots$), and a is a p-dimensional parameter vector determining the particular range of the given type (a particular disk etc.). The set of admissible ranges for such a problem is $\mathcal{R}_f = \{R_f(a);\ a \in \mathbb{R}^p\}$.

One possible way of solving the range searching problems with ranges from $\mathcal{R}_f$ is a direct transformation to halfspace range searching in space of higher dimension, the so-called *linearization* of the problem. Let us consider the example with disks, where

$$f(x_1, x_2, a_1, a_2, a_3) = a_3^2 - (x_1 - a_1)^2 - (x_2 - a_2)^2 = a_3^2 - a_1^2 - a_2^2 + 2a_1x_1 + 2a_2x_2 - x_1^2 - x_2^2.$$

We define a mapping $\varphi : \mathbb{R}^2 \to \mathbb{R}^4$ by $\varphi(x_1, x_2) = (x_1, x_2, x_1^2, x_2^2)$. A key property of this mapping is the following: For every disk $C = C(a_1, a_2, a_3)$ there exists a halfspace $H = H(a_1, a_2, a_3)$ in $\mathbb{R}^4$ such that $C = \varphi^{-1}(H)$ or in other words the points of the plane mapped by φ into the halfspace H are exactly those of the disk C. It is easy to see that a suitable halfspace is $H = \{(t_1, t_2, t_3, t_4) \in \mathbb{R}^4;\ 2a_1t_1 + 2a_2t_2 - t_3 - t_4 + a_3^2 - a_1^2 - a_2^2 \geq 0\}$. A mapping φ with the above described property is called a *linearization*. A linearization allows us to transform the disk range searching problem in the plane with a point set P to the halfspace range searching problem in $\mathbb{R}^4$ with the point set $\varphi(P)$.

The linearization φ was obtained by introducing one new coordinate t_j for each monomial in the variables x_1, x_2 occurring in the polynomial f. A problem with an arbitrary polynomial f can be linearized in the same way, and thus we obtain nontrivial range searching algorithms for ranges from $\mathcal{R}_f$. This important observation was made by Yao and Yao [YY85].

The example with disks, which we used to demonstrate the general method, also shows that this method is sometimes wasteful in the dimension of the image space. It is well-known that the disks in the plane can be linearized in the above sense in dimension 3. Geometrically, each disk can be obtained as a vertical projection of the intersection of a certain halfspace in $\mathbb{R}^3$ with the paraboloid $z = x^2 + y^2$ onto the plane $z = 0$. Thus a better linearization is obtained by mapping the plane onto the paraboloid by the mapping $(x, y) \mapsto (x, y, x^2 + y^2)$. This transformation, sometimes called a *lifting*, has many important consequences in computational geometry, e.g., a close relationship between Voronoi diagrams in the plane and convex polytopes in $\mathbb{R}^3$, see [Ede87]. A linearization of a minimum dimension for a given polynomial f can be found effectively, by solving a system of linear equations, see [AM92].

An important example of linearization in this sense is frequently used in problems dealing with lines in $\mathbb{R}^3$. A line in $\mathbb{R}^3$ is naturally described by 4 real parameters (coordinates). Given two lines ℓ, ℓ' in $\mathbb{R}^3$, the predicate "does ℓ lie above ℓ'" is nonlinear when expressed using this 4-parameter description of the lines. However, the predicate becomes linear if expressed using the so-called Plücker coordinates; these coordinates correspond to points in (projective) 5-dimensional space. Here the situation is further complicated by the necessity of dealing with oriented lines, which leads to interesting combinatorial and algorithmic problems. An initial work in this direction is Chazelle *et al.* [CEGS89], further developments are given by Pellegrini (e.g., [Pel92]), Pellegrini and Shor [PS92] and Agarwal [Aga93].

For some polynomials f, we get the best known range searching algorithms by a direct application of linearization. In other cases faster algorithms are obtained by generalizing the methods for halfspaces into the "nonlinear" setting. For example, for disk range searching in the plane the reduction to halfspace range searching in dimension 3 yields roughly $n^{2/3}$ query time with linear space. However, one can imitate e.g., the algorithm of [Mat92b], replacing halfplanes by circles and generalizing the necessary results appropriately, and arrive at a linear space algorithm with query time close to $\sqrt{n}$.

We will briefly discuss the complexity of algorithms constructed in this way for range searching problems with ranges in $\mathcal{R}_f$. More information can be found in [AM92]. Roughly speaking, the exponent in the query complexity for a linear space depends on d, the dimension of the space containing the point set, while the space requirement for a logarithmic query time depends mainly on p, the number of parameters determining a range (here we talk about bounds for specific algorithms; the true complexity might perhaps be smaller sometimes). For disk range searching in the plane we have $d = 2$, and the query time for linear space is approximately $n^{1-1/2} = \sqrt{n}$; the number of parameters $p = 3$, and storage close to $O(n^3)$ guarantees a logarithmic query time. From this suggestive sentence the reader might conclude that for a given d, p we obtain bounds close to $n^{1-1/d}$, resp. about n^p. This could (should?) indeed be the case, but a proof is only known if $d \leq 3$, resp. $p \leq 3$. For higher dimensions there is an obstacle, the unsolved problem of an efficient decomposition of an arrangement of algebraic surfaces.

We explain the required notions on a very informal level only, since exact definitions would occupy too much space. Let $\sigma_1, \ldots, \sigma_n$ be algebraic hypersurfaces in $\mathbb{R}^d$, i.e. subsets described by a single algebraic equation — the reader may imagine (hyper)spheres. Analogously as for a hyperplane arrangement, the hypersurfaces $\sigma_1, \ldots, \sigma_n$ partition $\mathbb{R}^d$ into cells (not necessarily convex ones anymore), each cell being characterized by a particular sign pattern for the polynomials defining $\sigma_1, \ldots, \sigma_n$. The number of such cells is $O(n^d)$ (the hidden constant depends on d and on the degrees of the defining polynomials). For imitating the techniques for halfspace range searching, we still need to partition each of the cells into elementary cells (as defined in the beginning of the current section). It is not important how the particular elementary cells used for the decomposition look like, it is essential that each of them is described by a constant number of parameters. For spheres, we could for instance use intersections of constantly many balls, spheres and complements of balls as the elementary cells.

If $\sigma_1, \ldots, \sigma_n$ are hyperplanes, it is not difficult to partition all cells of their arrangement into $O(n^d)$ simplices in total. It is conjectured that also for general algebraic hypersurfaces of bounded degree there exists a decomposition into approximately n^d elementary cells, but proving it seems to be hard. The best known general construction, due to Chazelle *et al.* [CEGS91] gives approximately n^{2d-3} elementary cells (for $d \geq 3$; in the planar case it is easy to obtain $O(n^2)$ elementary cells). This is almost optimal in dimension 3, but for higher dimensions there remains a significant gap between the lower bound of n^d and upper bound of approximately n^{2d-3}.

The relationship between this decomposition problem with range searching algorithms for ranges from $\mathcal{R}_f$ is as follows: If we can decompose the arrangement of any m hypersurfaces in $\mathbb{R}^d$ of a certain type (determined by the polynomial f) into $O(m^b)$ elementary cells, we obtain query complexity roughly $O(n^{1-1/b})$ with linear storage. Similarly decompositions into $O(m^b)$ elementary cells for any m hypersurfaces in $\mathbb{R}^p$ (again determined by f) imply logarithmic query time with space $O(n^{b+\varepsilon})$.

Let us remark that the described decomposition problem has several other motivations besides geometric range searching. For instance, it is very important for decision algorithms for the theory of real closed fields, although other aspects of the decomposition become important there (the degrees of polynomial defining the elementary cells in the decomposition etc.).

Even if the problem of decomposition can be solved satisfactorily, it is not clear whether the algorithms obtained using the above discussed techniques are close to optimal. For instance, consider the problem of range searching with cones in $\mathbb{R}^3$ of the form $z \geq (x-a)^2 + (y-b)^2$, a, b parameters determining the cone (this problem is equivalent to preprocessing a set of circles in the plane so that given a query point, the circles enclosing it can be found quickly). Here it is only known how to get a roughly $O(n^{2/3})$ query time with linear space; in the arithmetic model, however, the query complexity can be made only $O(\sqrt{n})$ with $O(n)$ generators (see section 3). Currently it is unclear if also a better algorithm exists, or if the arithmetic model is too unrealistic here and a stronger lower bound holds in some other model of computation.

In conclusion of this section, we mention recent combinatorial results which may prove very significant for the theory of geometric range searching. Let $\sigma_1, \ldots, \sigma_n \subset \mathbb{R}^d$ be graphs of algebraic functions $f_1, \ldots, f_n : \mathbb{R}^{d-1} \to \mathbb{R}$, respectively, of a constant bounded degree. The *lower envelope* of $\sigma_1, \ldots, \sigma_n$ is the graph of the pointwise minimum of $f_1, \ldots, f_n$. After a series of previous partial results, Halperin and Sharir [HS93] and Sharir [Sha93] recently proved nearly tight upper bound on the worst-case combinatorial complexity of the lower envelope — $O(n^{d-1+\varepsilon})$, for an arbitrarily small constant $\varepsilon > 0$.

Some range searching problems (typically range emptiness problems) can be formulated as detecting whether a query point lies below the lower envelope of a collection of algebraic surfaces in a small dimension d. Knowing the complexity of the lower envelope, one might hope to obtain algorithms with, say, polylogarithmic query time and storage $O(n^{d-1+\varepsilon})$, or even some form of space/query time tradeoff. Currently no general algorithm (essentially point location below the lower envelope) taking advantage of the new combinatorial bound is known. The obstacle is the decomposition of the lower envelope into constant complexity cells (again!). A special case (with a particular type of surfaces) was solved by Mohaban and Sharir [MS93b]. They consider testing whether a query line lies above all spheres of a given collection, and the analysis of their algorithm utilizes the new bound (for $d = 4$).

Related research aims at bounding the combinatorial complexity of a single cell in an arrangement of n algebraic surfaces of bounded degree in $\mathbb{R}^d$. One might again hope that this complexity is close to n^{d-1}; see [AS92] for partial results in this direction.

Parametric search

Parametric search is a general strategy for algorithm design. Roughly speaking, it produces algorithms for searching from algorithms for verification, under suitable assumptions.

Let us consider a problem in which the goal is to find a particular real number, t^*, which depends on some input objects. We consider these input objects fixed. Suppose that we have two algorithms at our disposal: First, an algorithm O, which for a given number t decides among the possibilities $t < t^*$, $t = t^*$ and $t > t^*$ (although it does not explicitly know t^*, only the input objects); let us call such an algorithm O the *oracle*. Second, an algorithm G (called the *generic* algorithm), whose computation depends on the input objects and on a real parameter t, and for which it is guaranteed that its computation for $t = t^*$ differs from the computation for any other $t \neq t^*$. We can use algorithm O also in the role of G, but often it is possible to employ a simpler algorithm for G. Under certain quite weak assumptions about algorithm G, the parametric search produces an algorithm for finding t^*.

The main idea is to simulate the computation of algorithm G for the (yet unknown) parameter value $t = t^*$. The computation of G of course depends on t, but we assume that all the required information about t is obtained by testing the signs of polynomials of small (constant bounded) degree in t. The coefficients in each such polynomial may depend on the input objects of the algorithm and on the outcomes of the previous tests, but not directly on t. The sign of a particular polynomial can be tested also in the unknown t^*: We find the roots $t_1, \ldots, t_k$ of the polynomial p, we locate t^* among them using the algorithm O and we derive the sign of $p(t^*)$ from it. In this way we can simulate the computation of the algorithm G at t^*.

If we record all tests involving t made by algorithm G during its computation, we can then find the (unique) value t^* giving appropriate results in all these tests, thereby solving the search problem.

In this version we need several calls to the oracle for every test performed by algorithm G. The second idea is to do many tests at once whenever possible. If algorithm G executes a group of mutually independent tests with polynomials $p_1(t), \ldots, p_m(t)$ (meaning that the polynomial p_i does not depend on the outcome of the test involving another polynomial p_j), we can answer all of them by $O(\log n)$ calls of the oracle: We compute the roots of all the polynomials $p_1, \ldots, p_m$ and we locate the position of t^* among them by binary search. Parametric search will thus be particularly efficient for algorithms G implemented in parallel, with a small number of parallel steps, since the tests in one parallel step are necessarily independent in the above mentioned sense.

Parametric search was formulated by Megiddo [Meg83], the idea of simulating an algorithm at a generic value appears in [ES76], [Gus83], [Meg79]. A technical improvement, which sometimes reduces the running time by a logarithmic factor, was suggested by Cole [Col87]. A generalization of parametric search to higher dimension, where the parameter t is a point in $\mathbb{R}^d$ and the oracle can test the position of t^* with respect to a given hyperplane, appears in [CSY87], [CM93b], [NPT92], [Mat93a]. Currently is parametric search a quite popular technique also in computational geometry; from numerous recent works we select more or less randomly [CSSS89], [AASS93], [AST92], [CEGS92].

Algorithms based on parametric search, although theoretically elegant, appear quite complicated for implementation. In many specific problems, parametric search can be replaced by a randomized algorithm (see [DMN92], [Mat91a]) or by other techniques (e.g., [CEGS92], [KS93]) with a similar efficiency.

6.5 Ray shooting and linear optimization

In this section we consider another type of generalization of geometric range searching problems. A good and important example is the *ray shooting problem*, whose special case was already mentioned as Problem 2.3. In such problems we are given some set Γ of geometric objects (planes, triangles, balls, ...) and the goal is to construct a data structure such that for a given point o and direction θ we can quickly determine the object of Γ hit by a ray ρ sent from the point o in the direction θ (more formally, ρ is a semiline originating in o and the question is which object of Γ is the one intersecting ρ closest to o).

> This problem is very popular in computer graphics, where it arises as an auxiliary problem in determining visibility of objects, hidden surface elimination, ray tracing and in other situations.

The ray shooting problem has a somewhat different flavor than the range searching problems, but it can be solved efficiently using data structures for suitable derived range searching problems. We illustrate this relation on a specific problem, ray shooting in a convex polytope. Here the role of Γ is played by a set H of hyperplanes in $\mathbb{R}^d$. For simplicity let us assume that none of the hyperplanes of H is vertical, and let U denote the upper unbounded cell of the arrangement of H, i.e. the set of points lying above all the hyperplanes. In our ray shooting problem, we only permit rays originating in U.

> This problem is useful in a number of applications. For instance, the problem of determining the closest point from a set $P \subset \mathbb{R}^{d-1}$ to a given point, the so-called *post office problem*, can be transformed to it. In dimensions higher than 3 this yields the most efficient algorithms for the post office problem, which can moreover be dynamized, see [MS93a].

Let us suppose that the given query ray ρ with origin $o \in U$ intersects the first hyperplane from H in a point x^*. Finding this point is our goal (together with the appropriate hyperplane). Given any point $x \in \rho$, we can decide whether x lies before x^* or after x^*: It suffices to test whether the segment ox intersects at least one hyperplane of H, and by the assumption $o \in U$ this happens iff $x \notin U$. The test whether x lies in U, that is, whether it lies above all hyperplanes of H, is the dual version of the halfspace emptiness problem, and thus the algorithms discussed in section 6.1 are suitable for such tests.

Therefore, although we do not know x^* yet, we can efficiently determine whether some point $x \in \rho$ precedes or follows after x^*, and we would like to find x^* using tests of this type. The method of interval halving suggests itself; it can determine the position of x^* with some required numerical precision. This approach may not be bad in practice, but in our infinite precision computation model it does not suffice since an arbitrarily small interval along ρ may still contain many intersections with hyperplanes of H, and thus we cannot determine the first intersected hyperplane in a bounded number of steps. However, we can apply the method of *parametric search*, see the box on page 37, which allows us to find x^* exactly and usually quite efficiently. In connection with this method, data structures for the halfspace emptiness problem can be applied for ray shooting in a convex polytope as well. The space requirement remains the same (as the data structure is identical), and the query time increases by a small power of $\log n$.

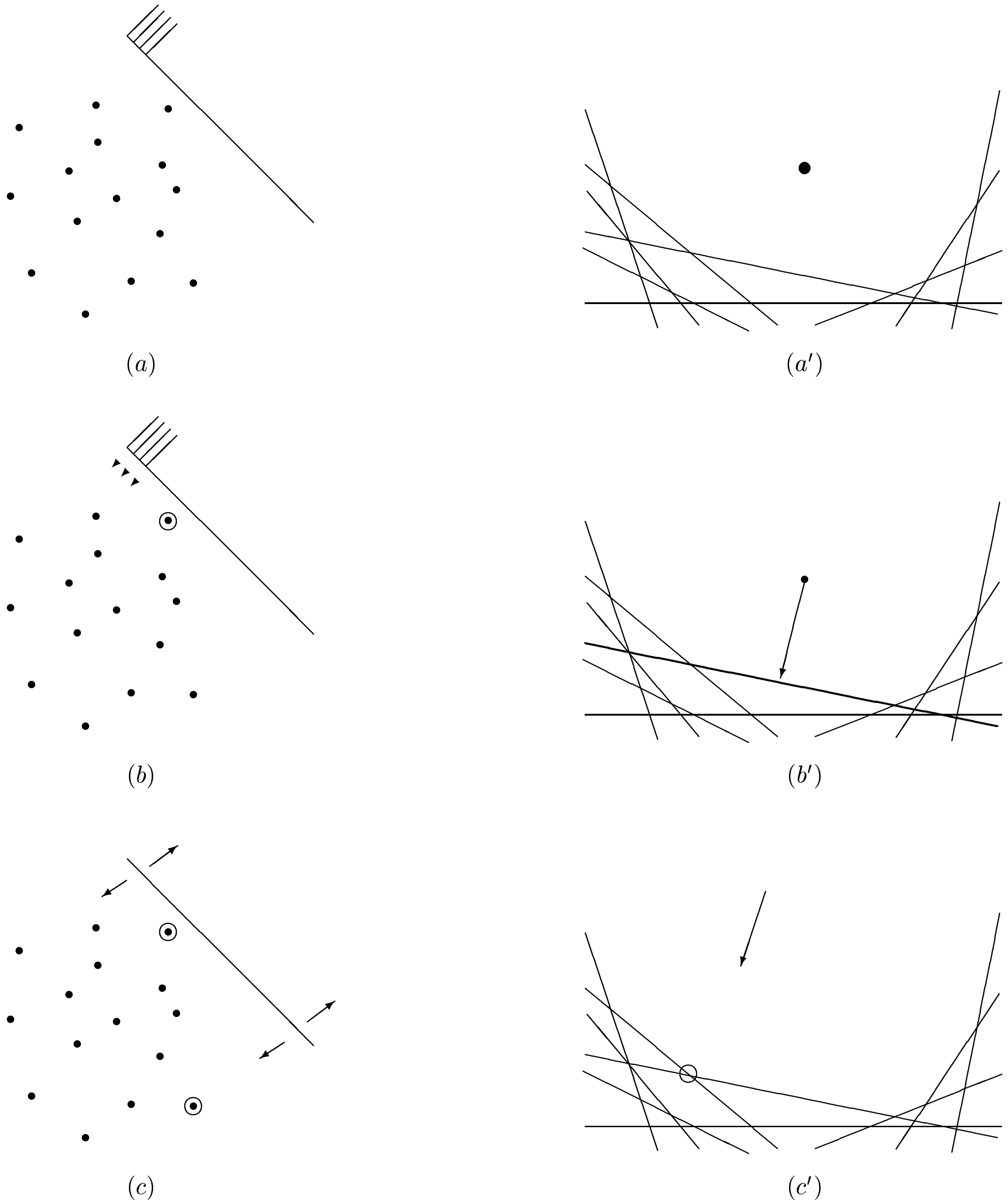


Figure 4: Adding degrees of freedom to the halfspace emptiness query.

The above described passage from halfspace emptiness testing to ray shooting in convex polytope is illustrated in fig. 4. Fig. (a) indicates a halfspace emptiness query schematically, (a') illustrates the dual version — testing whether a query point lies above all hyperplanes. Fig. (b') depicts the ray shooting problem; the dual version (b) means that we translate the bounding hyperplane of a given halfspace, and we are interested in the first point hit by this hyperplane. In both the primal and the dual versions, we can interpret the passage from (a) to (b) as adding one degree of freedom to the query object (point, halfspace) and looking for an extremal position within the given freedom.

> The approach explained for ray shooting in convex polytopes is also applicable for other versions of the ray shooting problem. First we formulate the derived range searching problem, namely a test whether some initial segment ox of an admissible ray intersects at least one object from Γ. For this problem, we build a data structure by the techniques for geometric range searching (usually we construct a suitable multilevel data structure). Then we apply parametric search and obtain a ray shooting algorithm. The application of this strategy for particular ray shooting problems need not be quite routine, however.
>
> The idea of employing range searching data structures for ray shooting is a natural one and it appears in several papers, see e.g., [dBHO$^+$]. The above described systematic approach via parametric search was suggested in [AM93] and demonstrated on several examples. Some particular data structures for halfspace emptiness queries can be adapted directly for ray shooting in convex polytopes, which eliminates parametric search and speeds up query answering somewhat, see [MS93a].

Let us return to ray shooting in convex polytopes and its interpretation as membership queries for the convex polytope with one added degree of freedom, see fig. 4. From this point of view it is natural to add still more degrees of freedom to the query object, at most all the d degrees of freedom, and look for some extremal position. Such a situation is indicated in fig. 4(c'): instead of a single point or an (oriented) ray we are given a linear function c (which can be interpreted as a direction in d-space, that is, a generalization of the orientation of the ray), and we want to find a vertex x_0 of the polytope U maximizing the function c.

The resulting problem is in fact a linear programming problem, but in a special situation, where the constraints are given in advance, while the optimized function comes as a query[12]. In [Mat93a] it is shown that algorithms for halfspace emptiness queries can be transformed to solve also this linear programming problem, with query complexity increased by a polylogarithmic factor only (and with the same data structure). This result uses a multidimensional version of parametric search, and the construction goes by induction on the number of degrees of freedom of the query object. The resulting algorithm is quite complicated, and it would be useful to find a simpler (maybe randomized) variant.

[12]The dual version is not so intuitive in this case, it is the finding of an empty halfspace maximizing a given linear function. Some optimal halfspace is always determined by a d-tuple of points from the given point set or in other words it is supported by a facet of the convex hull, see fig. 4(c).

With a dynamic halfspace emptiness data structure as a basis, one may also insert and delete constraints. Using this machinery one can answer various queries concerning the convex hull of the set P, as we have mentioned in the end of section 6.2.

Let us mention an application of this result to one classical computational geometry problem, that of finding extremal points. The input is an n-point set $P \subset \mathbb{R}^d$, and we want to detect which points of P are extremal, i.e. are vertices of the convex hull of P. This problem can be solved by computing a combinatorial representation of the convex hull of P, but for dimensions $d \geq 4$ this method is fairly inefficient, as the convex hull of an n-point set in $\mathbb{R}^d$ can have combinatorial complexity of the order $n^{\lfloor d/2 \rfloor}$. Testing whether a point is extremal can be formulated as a linear programming problem in dimension d with n constraints. For different points these linear programs only differ by two constraints, so we can use linear programming in the preprocessing/query mode. We build a data structure for the appropriate constraints, and with its help we answer n queries, thereby determining the extremal points[13]. Balancing the preprocessing time and query time suitably, we obtain, for example, a total time $O(n^{4/3+\varepsilon})$ in dimension 4, which is the most efficient known method.

6.6 More on applications

The computational geometry literature with applications of range searching is too extensive to be reviewed here. A survey paper containing many such applications is [Aga91]. Let us only stress one point, namely that most of the problem where range searching has been applied are not of the preprocessing/query type. An example is the computation of extreme points mentioned in the end of the previous section.

Let us give one more (rather artificial) example to illustrate our point. Suppose that we want to count the number of pairwise intersections for a set S of n segments in the plane. An approach via range searching is to build a data structure which, given a query segment q, counts the number of segments of S intersected by it. Then we query the structure by each segment of S, sum the answers and divide the sum by 2 (we must somehow deal with the singular cases of segments intersecting themselves, but this can be taken care of). With the methods reviewed above, the required data structure is designed in a more or less routine manner (using the multilevel construction and halfplane decomposition schemes). When the preprocessing and query time are balanced appropriately, we obtain an $O(n^{4/3+\varepsilon})$ algorithm.

The reader might suspect that this is a very sloppy way of solving this problem, and that we could do better with some "global" method, rather than imposing the preprocessing/query mode not present in the original problem. This is true only partially: There *do* exist somewhat better and simpler algorithms (ones based on cuttings, see e.g., [Aga90]), but their complexity is only slightly better, about $O(n^{4/3} \log^c n)$ for a small constant c (we don't give a specific value, since using some newer results on cuttings one can improve published algorithms somewhat), and it is believed that this might be roughly the

[13]It would seem that we must use a dynamic data structure, but in fact one can use a special treatment for this problem and get away with a static structure.

actual complexity of the problem[14]. Thus, the application of range searching tools gave us a theoretically good algorithm very quickly.

This situation seems to be no exception. For many problems, the algorithm based on range searching indicates what complexity we can expect also for other, problem specific and hopefully simpler and more practical algorithms. And for many problems which are more complicated, no such specific algorithms have been elaborated and the range searching approach has remained the best published solution. Examples are hidden surface removal problems (see e.g. [dBHO$^+$]), counting circular arc intersections [APS93a], problems concerning lines in space (e.g. [Pel92]) and the above mentioned extreme points detection to quote only few.

7 Last paragraph

This text was intended as a survey article about geometric range searching. Although its extent is much larger than originally planned, it does not include everything which it perhaps should include. The choice of the material necessarily reflects taste and knowledge of the author, and in topics mentioned only briefly (such as the broad area of applications) we cannot but refer to the literature.

Acknowledgment. I would like to thank Pankaj K. Agarwal and Raimund Seidel for reading preliminary versions of this paper and numerous useful comments.

[14]No lower bound larger than $n \log n$ is known, and problems of this type seem much less tractable than the query type problems as far as lower bounds are concerned.

Dim	Query type	Space	Query time	Preprocessing	Update time (amortized)	Source
			Simplex range searching — lower bounds			
2	general[1]	m	$n/\sqrt{n}$	—	—	[Cha89]
$d \geq 3$	general[1]	m	$n/(\log n\ m^{1/d})$	—	—	[Cha89]
$d \geq 2$	reporting[2]	m	$(n^{1-\varepsilon}/m^{1/d}) + k$	—	—	[CR92]
			Simplex range searching — upper bounds			
2	general	n	$\sqrt{n}\log n$	polynom.[4]	—	[CW89]
3	general	$n\log n$	$n^{2/3}\log^2 n$	polynom.[4]	—	[CW89]
$d \geq 2$	general	n	$n^{1-1/d}$	$n^{1+\varepsilon}$	—	[Mat92c]
$d \geq 2$	general	n	$n^{1-1/d}\log^c n$	$n\log n$	$\log^2 n$	[Mat92b]
$d \geq 2$	general	n^d	$\log^{d+1} n$	$n^d(\log n)^\varepsilon$	—	[Mat92c]
$d \geq 2$	general	$n^{d+\varepsilon}$	$\log^{d+1} n$	$n^{d+\varepsilon}$	$n^{d-1+\varepsilon}$	[CSW92][3]
$d \geq 2$	general	m	$(n/m^{1/d})\log^{d+1}\frac{m}{n}$	$n^{1+\varepsilon} + m(\log n)^\varepsilon$	—	[Mat92c]
$d \geq 2$	general	$m^{1+\varepsilon}$	$(n/m^{1/d})\log^{d+1} n$	$m^{1+\varepsilon}$	$m^{1+\varepsilon}/n$	[CSW92][3]
			Halfspace range searching — lower bound			
$d \geq 2$	general[1]	m	$\frac{(n/\log n)^{1-(d-1)/d(d+1)}}{m^{1/d}}$	—	—	[BCP93]
			Halfspace range searching — upper bounds[5]			
$d \geq 2$	general	$n^d/\log^d n$	$\log n$	$n^d/\log^{d-\varepsilon} n$	—	[Mat92c]
$d \geq 2$	general	m	$n/m^{1/d}$	$n^{1+\varepsilon} + m(\log n)^\varepsilon$	—	[Mat92c]
2	reporting	n	$\log n + k$	$n\log n$	—	[CGL85]
3	reporting	$n\log n$	$\log n + k$	$n\log^3 n\log\log n$ [6]	—	[AHL90]
$d \geq 4$	reporting	$n\log\log n$	$n^{1-1/\lfloor d/2\rfloor}\log^c n + k$	$n\log n$	—	[Mat92a]
$d \geq 4$	emptiness	n	$n^{1-1/\lfloor d/2\rfloor}\log^c n$	$n\log n$	—	[Mat92a]
$d \geq 4$	emptiness	n	$n^{1-1/\lfloor d/2\rfloor}2^{c\log^* n}$	$n^{1+\varepsilon}$	—	[Mat92a]
$d \geq 2$	reporting	n	$n^{1-1/\lfloor d/2\rfloor+\varepsilon} + k$	n	$\log^2 n$	[AM91]
$d \geq 4$	emptiness[7]	$n^{\lfloor d/2\rfloor}\log^c n$	$\log n$	$n^{\lfloor d/2\rfloor}\log^c n$	—	[MS93a]
$d \geq 4$	emptiness	$n^{\lfloor d/2\rfloor}/\log^{\lfloor d/2\rfloor-\varepsilon} n$	$\log n$	$n^{\lfloor d/2\rfloor}/\log^{\lfloor d/2\rfloor-\varepsilon} n$ [8]	—	[MS93a]
$d \geq 2$	reporting	$n^{\lfloor d/2\rfloor+\varepsilon}$	$\log n + k$	$n^{\lfloor d/2\rfloor+\varepsilon}$	$n^{\lfloor d/2\rfloor-1+\varepsilon}$	[Cla88b][9]
$d \geq 4$	emptiness[7]	$m\log^c n$	$n/m^{1/d}\log n$	$m\log^c n$	—	[MS93a]
$d \geq 2$	reporting	$m^{1+\varepsilon}$	$(n/m^{1/d})\log n + k$	$m^{1+\varepsilon}$	$m^{1+\varepsilon}/n$	[AM91]

Explanations. The $O()$ symbol for upper bounds and the $\Omega()$ symbol for lower bounds are omitted. The letter m denotes a parameter which can be chosen at will; for simplex range searching and general halfspace range searching, its range is $[n, n^d]$, and for halfspace emptiness or halfspace reporting it is $[n, n^{\lfloor d/2\rfloor}]$. The letter ε denotes a positive constant which can be made arbitrarily small by adjusting the parameters of the algorithm, c is a generic letter for variouus specific constants (depending on d). The algorithms achieving the preprocessing times are deterministic unless stated otherwise.

[1]The bounds hold in the arithmetic model.

[2]The bound holds in the pointer machine model.

[3]Dynamization is due to [AS93].

[4]The preprocessing can be made $O(n^{1+\varepsilon})$ using known methods (this has not been published).

[5]All upper bounds for simplex range searching also apply.

[6]Within this time, a randomized preprocessing algorithm produces a data structure having the given parameters with high probability.

[7]The algorithms also work for ray shooting in a convex polytope, and can also be adjusted for halfspace reporting queries with some additional effort (the published version only deals with emptiness queries).

[8]Randomized expected time.

[9]Dynamization is due to [AM91].

Table 1: Results on the simplex and halfspace range searching problems

References

[AASS93] P. K. Agarwal, B. Aronov, M. Sharir, and S. Suri. Selecting distances in the plane. *Algorithmica*, 9:495–514, 1993.

[Aga90] P. K. Agarwal. Partitioning arrangements of lines: II. Applications. *Discrete Comput. Geom.*, 5:533–573, 1990.

[Aga91] P. K. Agarwal. Geometric partitioning and its applications. In J.E. Goodman, R. Pollack, and W. Steiger, editors, *Computational Geometry: Papers from the DIMACS special year*. Amer. Math. Soc., 1991.

[Aga93] P. K. Agarwal. On stabbing lines for polyhedra in 3d. Tech. Rept. CS-1993-09, Department Computer Science, Duke University, 1993.

[AHL90] A. Aggarwal, M. Hansen, and T. Leighton. Solving query-retrieval problems by compacting Voronoi diagrams. In *Proc. 22nd Annu. ACM Sympos. Theory Comput.*, pages 331–340, 1990.

[AHW87] N. Alon, D. Haussler, and E. Welzl. Partitioning and geometric embedding of range spaces of finite Vapnik-Chervonenkis dimension. In *Proc. 3rd Annu. ACM Sympos. Comput. Geom.*, pages 331–340, 1987.

[AM91] P. K. Agarwal and J. Matoušek. Dynamic half-space range reporting and its applications. Tech. Report CS-1991-43, Duke University, 1991. Extended abstract, including also results of D. Eppstein: *Proc. 33. IEEE Symposium on Foundations of Computer Science* (1992), pages 51–60.

[AM92] P. K. Agarwal and J. Matoušek. On range searching with semialgebraic sets. In *Proc. 17th Internat. Sympos. Math. Found. Comput. Sci.*, volume 629 of *Lecture Notes in Computer Science*, pages 1–13. Springer-Verlag, 1992. Also to appear in *Discrete Comput. Geom.*

[AM93] P. K. Agarwal and J. Matoušek. Ray shooting and parametric search. *SIAM Journal on Computing*, 22(4):794–806, 1993.

[APS93a] P. K. Agarwal, M. Pellegrini, and M. Sharir. Counting circular arc intersections. *SIAM Journal on Computing*, 22:778–793, 1993.

[APS93b] B. Aronov, M. Pellegrini, and M. Sharir. On the zone of a surface in a hyperplane arrangement. *Discrete Comput. Geom.*, 9(2):177–186, 1993.

[AS92] B. Aronov and M. Sharir. Castles in the air revisited. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 146–156, 1992. Also to appear in *Discrete & Computational Geometry.*

[AS93] P.K. Agarwal and M. Sharir. Applications of a new partition scheme. *Discrete & Computational Geometry*, 9:11–38, 1993.

[AST92] P. K. Agarwal, M. Sharir, and S. Toledo. Applications of parametric searching in geometric optimization. In *Proc. 3rd ACM-SIAM Sympos. Discrete Algorithms*, pages 72–82, 1992.

[Avi84] D. Avis. Non-partitionable point sets. *Inform. Process. Lett.*, 19:125–129, 1984.

[BCP93] H. Brönnimann, B. Chazelle, and J. Pach. How hard is halfspace range searching. *Discrete & Computational Geometry*, 10:143–155, 1993.

[Ben80] J. L. Bentley. Multidimensional divide-and-conquer. *Commun. ACM*, 23(4):214–229, 1980.

[BS80] J. L. Bentley and J. B. Saxe. Decomposable searching problems I: static-to-dynamic transformation. *J. Algorithms*, 1:301–358, 1980.

[CCPY86] B. Chazelle, R. Cole, F. P. Preparata, and C. K. Yap. New upper bounds for neighbor searching. *Inform. Control*, 68:105–124, 1986.

[CEG+90] K. Clarkson, H. Edelsbrunner, L. Guibas, M. Sharir, and E. Welzl. Combinatorial complexity bounds for arrangements of curves and spheres. *Discrete Comput. Geom.*, 5:99–160, 1990.

[CEGS89] B. Chazelle, H. Edelsbrunner, L. J. Guibas, and M. Sharir. Lines in space: combinatorics, algorithms, and applications. In *Proc. 21st Annu. ACM Sympos. Theory Comput.*, pages 382–393, 1989.

[CEGS91] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir. A singly-exponential stratification scheme for real semi-algebraic varieties and its applications. *Theoret. Comput. Sci.*, 84:77–105, 1991.

[CEGS92] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir. Diameter, width, closest line pair, and parametric searching. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 120–129, 1992.

[CF] B. Chazelle and J. Friedman. Point location among hyperplanes and vertical ray shooting. *Computational Geometry: Theory and Applications*. To appear.

[CF90] B. Chazelle and J. Friedman. A deterministic view of random sampling and its use in geometry. *Combinatorica*, 10(3):229–249, 1990.

[CGL85] B. Chazelle, L. J. Guibas, and D. T. Lee. The power of geometric duality. *BIT*, 25:76–90, 1985.

[Cha86] B. Chazelle. Filtering search: a new approach to query-answering. *SIAM J. Comput.*, 15:703–724, 1986.

[Cha88] B. Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Comput.*, 17:427–462, 1988.

[Cha89] B. Chazelle. Lower bounds on the complexity of polytope range searching. *J. Amer. Math. Soc.*, 2:637–666, 1989.

[Cha90a] B. Chazelle. Lower bounds for orthogonal range searching, I: the reporting case. *J. ACM*, 37:200–212, 1990.

[Cha90b] B. Chazelle. Lower bounds for orthogonal range searching, II: the arithmetic model. *J. ACM*, 37:439–463, 1990.

[Cha91] B. Chazelle. An optimal convex hull algorithm for point sets in any fixed dimension. Tech. report CS-TR-336-91, Princeton University, 1991. Preliminary version: Proc. 32nd IEEE Symposium on Foundations of Computer Science (1991). To appear in *Discrete & Computational Geometry.*

[Cha93] B. Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete Comput. Geom.*, 9(2):145–158, 1993.

[Che85] L. P. Chew. Building Voronoi diagrams for convex polygons in linear expected time. Report, Dept. Math. Comput. Sci., Dartmouth College, Hanover, NH, 1985.

[Cla87] K. L. Clarkson. New applications of random sampling in computational geometry. *Discrete Comput. Geom.*, 2:195–222, 1987.

[Cla88a] K. L. Clarkson. Applications of random sampling in computational geometry, II. In *Proc. 4th Annu. ACM Sympos. Comput. Geom.*, pages 1–11, 1988.

[Cla88b] K. L. Clarkson. A randomized algorithm for closest-point queries. *SIAM J. Comput.*, 17:830–847, 1988.

[CM93a] B. Chazelle and J. Matoušek. On linear-time deterministic algorithms for optimization problems in fixed dimension. In *Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*, pages 281–290, 1993.

[CM93b] E. Cohen and N. Megiddo. Strongly polynomial-time and NC algorithms for detecting cycles in dynamic graphs. *Journal of the ACM*, 40:791–832, 1993.

[Col85] R. Cole. Partitioning point sets in 4 dimensions. In *Proc. 12th Internat. Colloq. Automata Lang. Program.*, volume 194 of *Lecture Notes in Computer Science*, pages 111–119. Springer-Verlag, 1985.

[Col87] R. Cole. Slowing down sorting networks to obtain faster sorting algorithms. *Journal of the ACM*, 34:200–208, 1987.

[CP86] B. Chazelle and F. P. Preparata. Halfspace range search: an algorithmic application of k-sets. *Discrete Comput. Geom.*, 1:83–93, 1986.

[CR91] B. Chazelle and B. Rosenberg. The complexity of computing partial sums off-line. *Internat. J. Comput. Geom. Appl.*, 1(1):33–45, 1991.

[CR92] B. Chazelle and B. Rosenberg. Lower bounds on the complexity of simplex range reporting on a pointer machine. In *International Colloquium on Automata, Languages and Programming*, 1992. Also to appear in *Computational Geometry: Theory and Applications.*

[CSSS89] R. Cole, J. Salowe, W. Steiger, and E. Szemerédi. An optimal-time algorithm for slope selection. *SIAM J. Comput.*, 18:792–810, 1989.

[CSW92] B. Chazelle, M. Sharir, and E. Welzl. Quasi-optimal upper bounds for simplex range searching and new zone theorems. *Algorithmica*, 8:407–429, 1992.

[CSY87] R. Cole, M. Sharir, and C. Yap. On k-hulls and related problems. *SIAM Journal on Computing*, 16(1):61–67, 1987.

[CW89] B. Chazelle and E. Welzl. Quasi-optimal range searching in spaces of finite VC-dimension. *Discrete Comput. Geom.*, 4:467–489, 1989.

[CY85] R. Cole and C. K. Yap. Geometric retrieval problems. *Inform. Control*, 63:39–57, 1985.

[dBHO$^+$] M. de Berg, D. Halperin, M. Overmars, J. Snoeyink, and M. van Kreveld. Efficient ray shooting and hidden surface removal. *Algorithmica.* To appear. Extended abstract: Proc. 7. ACM Symposium on Computational Geometry, pages 21–30, 1991.

[DE84] D. Dobkin and H. Edelsbrunner. Organizing point sets in two and three dimensions. Tech. Report F130, Technische Universität Graz, 1984.

[DE87] D. Dobkin and H. Edelsbrunner. Space searching for intersecting objects. *Journal of Algorithms*, 8:348–361, 1987.

[DMN92] M. B. Dillencourt, D. M. Mount, and N. S. Netanyahu. A randomized algorithm for slope selection. *Internat. J. Comput. Geom. Appl.*, 2:1–27, 1992.

[EC91] I. Emiris and J. Canny. A general approach to removing degeneracies. In *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 405–413, 1991.

[EC92] I. Emiris and J. Canny. An efficient approach to removing geometric degeneracies. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 74–82, 1992.

[Ede87] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Heidelberg, West Germany, 1987.

[EGH+89] H. Edelsbrunner, L. Guibas, J. Hershberger, R. Seidel, M. Sharir, J. Snoeyink, and E. Welzl. Implicitly representing arrangements of lines or segments. *Discrete Comput. Geom.*, 4:433–466, 1989.

[EH84] H. Edelsbrunner and F. Huber. Dissecting sets of points in two and three dimensions. Report F138, Inst. Informationsverarb., Tech. Univ. Graz, Graz, Austria, 1984.

[EKM82] H. Edelsbrunner, D. G. Kirkpatrick, and H. A. Maurer. Polygonal intersection searching. *Inform. Process. Lett.*, 14:74–79, 1982.

[EM90] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph.*, 9:66–104, 1990.

[ES76] M. Eisner and D. Severance. Mathematical techniques for efficient record segmentation in large shared database. *Journal of the ACM*, 23:619–635, 1976.

[EW86] H. Edelsbrunner and E. Welzl. Halfplanar range search in linear space and $O(n^{0.695})$ query time. *Inform. Process. Lett.*, 23:289–293, 1986.

[Fre81] M. L. Fredman. Lower bounds on the complexity of some optimal data structures. *SIAM J. Comput.*, 10:1–10, 1981.

[FvW93] S. Fortune and C. van Wijk. Efficient exact arithmetic for computational geometry. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 163–172, 1993.

[GBT84] H. N. Gabow, J. L. Bentley, and R. E. Tarjan. Scaling and related techniques for geometry problems. In *Proc. 16th Annu. ACM Sympos. Theory Comput.*, pages 135–143, 1984.

[GPS91] J.E. Goodman, R. Pollack, and W. Steiger, editors. *Computational Geometry: Papers from the DIMACS special year.* Amer. Math. Soc., 1991.

[Gus83] D. Gusfield. Parametric combinatorial computing and a problem of program module distribution. *Journal of the ACM*, 30:551–563, 1983.

[Hau91] D. Haussler. Sphere packing numbers for subsets of the Boolean n-cube with bounded Vapnik-Chervonenkis dimension. Tech. Report UCSC-CRL-91-41, University of California at Santa Cruz, 1991.

[HS93] D. Halperin and M. Sharir. New bounds for lower envelopes in three dimensions, with applications to visibility in terrains. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 11–18, 1993.

[HW87] D. Haussler and E. Welzl. Epsilon-nets and simplex range queries. *Discrete Comput. Geom.*, 2:127–151, 1987.

[KS93] M. J. Katz and M. Sharir. An expander-based approach to geometric optimization. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 198–207, 1993.

[KSP82] J. Komlós, E. Szemerédi, and J. Pintz. A lower bound for Heilbronn's problem. *J. London Math. Soc.*, 25(2):13–24, 1982.

[Mat90] J. Matoušek. Construction of ϵ-nets. *Discrete Comput. Geom.*, 5:427–448, 1990.

[Mat91a] J. Matoušek. Randomized optimal algorithm for slope selection. *Inform. Process. Lett.*, 39:183–187, 1991.

[Mat91b] J. Matoušek. Approximations and optimal geometric divide-and-conquer. In *Proc. 23rd ACM Symposium on Theory of Computing*, pages 506–511, 1991. Also to appear in *J. Comput. Syst. Sci.*

[Mat91c] J. Matoušek. Cutting hyperplane arrangements. *Discrete & Computational Geometry*, 6(5):385–406, 1991.

[Mat92a] J. Matoušek. Reporting points in halfspaces. *Comput. Geom. Theory Appl.*, 2(3):169–186, 1992.

[Mat92b] J. Matoušek. Efficient partition trees. *Discrete Comput. Geom.*, 8:315–334, 1992.

[Mat92c] J. Matoušek. Range searching with efficient hierarchical cuttings. In *Proc. 8th ACM Symposium on Computational Geometry*, pages 276–285, 1992. Discrete & Computational Geometry 1993, to appear.

[Mat93a] J. Matoušek. Linear optimization queries. *J. Algorithms*, 14:432–448, 1993. The results combined with results of O. Schwarzkopf also appear in *Proc. 8th ACM Sympos. Comput. Geom.*, 1992, pages 16–25.

[Mat93b] J. Matoušek. On vertical ray shooting in arrangements. *Comput. Geom. Theory Appl.*, 2(5):279–285, March 1993.

[Meg79] N. Megiddo. Combinatorial optimization with rational objective functions. *Math. Oper. Res.*, 4:414–424, 1979.

[Meg83] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM*, 30:852–865, 1983.

[Meh84] K. Mehlhorn. *Multi-dimensional Searching and Computational Geometry*, volume 3 of *Data Structures and Algorithms.* Springer-Verlag, Heidelberg, West Germany, 1984.

[MS93a] J. Matoušek and O. Schwarzkopf. On ray shooting in convex polytopes. *Discrete & Computational Geometry*, 1993. To appear.

[MS93b] S. Mohaban and M. Sharir. Ray shooting amidst spheres in three dimensions and related problems. Manuscript, 1993.

[Mul88] K. Mulmuley. A fast planar partition algorithm, I. In *Proc. 29th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 580–589, 1988.

[Mul91a] K. Mulmuley. A fast planar partition algorithm, II. *J. ACM*, 38:74–103, 1991.

[Mul91b] K. Mulmuley. Randomized multidimensional search trees: dynamic sampling. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 121–131, 1991.

[Mul91c] K. Mulmuley. Randomized multidimensional search trees: further results in dynamic sampling. In *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 216–227, 1991.

[Mul91d] K. Mulmuley. Randomized multidimensional search trees: lazy balancing and dynamic shuffling. In *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 180–196, 1991.

[Mul92] K. Mulmuley. Randomized geometric algorithms and pseudo-random generators. In *Proc. 33rd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 90–100, 1992.

[Mul93] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms*. Prentice Hall, New York, 1993.

[MW92] J. Matoušek and E. Welzl. Good splitters for counting points in triangles. *J. Algorithms*, 13:307–319, 1992.

[MWW93] J. Matoušek, E. Welzl, and L. Wernisch. Discrepancy and ε-approximations for bounded VC-dimension. *Combinatorica*, 1993. To appear. Also in Proc. 32. IEEE Symposium on Foundations of Computer Science (1991), pages 424–430.

[NPT92] C. H. Norton, S. A. Plotkin, and É. Tardos. Using separation algorithms in fixed dimensions. *J. Algorithms*, 13:79–98, 1992.

[OSS90] M. Overmars, H. Schipper, and M. Sharir. Storing line segments in partition trees. *BIT*, 30:385–403, 1990.

[Ove83] M. H. Overmars. *The design of dynamic data structures*, volume 156 of *Lecture Notes in Computer Science*. Springer-Verlag, 1983.

[Ove88] M. H. Overmars. Efficient data structures for range searching on a grid. *J. Algorithms*, 9:254–275, 1988.

[Pac93] J. Pach, editor. *New Trends in Discrete and Computational Geometry*, volume 10 of *Algorithms and Combinatorics*. Springer-Verlag, 1993.

[Pel92] M. Pellegrini. Incidence and nearest-neighbor problems for lines in 3-space. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 130–137, 1992.

[PS85] F. P. Preparata and M. I. Shamos. *Computational Geometry: an Introduction*. Springer-Verlag, New York, NY, 1985.

[PS92] M. Pellegrini and P. Shor. Finding stabbing lines in 3-space. *Discrete & Computational Geometry*, 8:191–208, 1992.

[PT92] F. P. Preparata and R. Tamassia. Efficient point location in a convex spatial cell-complex. *SIAM J. Comput.*, 21:267–280, 1992.

[PY86] M. S. Paterson and F. F. Yao. Point retrieval for polygons. *J. Algorithms*, 7:441–447, 1986.

[Rab76] M. O. Rabin. Probabilistic algorithms. In J. F. Traub, editor, *Algorithms and Complexity*, pages 21–30. Academic Press, New York, NY, 1976.

[Rot76] K. Roth. Developments in the triangle problem. *Adv. Math.*, 22:364–385, 1976.

[Sch91] O. Schwarzkopf. Dynamic maintenance of geometric structures made easy. In *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 197–206, 1991.

[Sch92] O. Schwarzkopf. Ray shooting in convex polytopes. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 286–295, 1992.

[Sei91a] R. Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Comput. Geom. Theory Appl.*, 1:51–64, 1991.

[Sei91b] R. Seidel. Small-dimensional linear programming and convex hulls made easy. *Discrete Comput. Geom.*, 6:423–434, 1991.

[Sha93] Micha Sharir. Almost tight upper bounds for lower envelopes in higher dimensions. In *Proc. 34th Annu. IEEE Sympos. Found. Comput. Sci. (FOCS 93)*, page ??, 1993.

[SO90] H. Schipper and M. H. Overmars. Dynamic partition trees. In *Scandawian Workshop on Algorithms Theory*, volume 2, pages 404–417. Springer-Verlag, 1990. LNCS 447; also to appear in *BIT*.

[Wel88] E. Welzl. Partition trees for triangle counting and other range searching problems. In *Proc. 4th Annu. ACM Sympos. Comput. Geom.*, pages 23–33, 1988.

[Wel92] E. Welzl. On spanning trees with low crossing numbers. Tech. Report B-92-2, FU Berlin, FB Mathematik, 1992.

[Wil82] D. E. Willard. Polygon retrieval. *SIAM J. Comput.*, 11:149–165, 1982.

[Wil85] D. E. Willard. New data structures for orthogonal range queries. *SIAM J. Comput.*, 14:232–253, 1985.

[WL85] D. E. Willard and G. S. Lueker. Adding range restriction capability to dynamic data structures. *J. ACM*, 32:597–617, 1985.

[Yao83] F. F. Yao. A 3-space partition and its applications. In *Proc. 15th Annu. ACM Sympos. Theory Comput.*, pages 258–263, 1983.

[YDEP89] F. F. Yao, D. P. Dobkin, H. Edelsbrunner, and M. S. Paterson. Partitioning space for range queries. *SIAM J. Comput.*, 18:371–384, 1989.

[YY85] 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.