Randomized External-Memory Algorithms for Some Geometric Problems

A. Crauser P. Ferragina K. Mehlhorn U. Meyer E. Ramos
Max-Planck-Institut für Informatik
Im Stadtwald, 66123 Saarbrücken, Germany
{crauser,paolo,mehlhorn,umeyer,ramos}@mpi-sb.mpg.de

Abstract

We show that the well-known random incremental construction of Clarkson and Shor [14] can be adapted via *gradations* to provide efficient external-memory algorithms for some geometric problems. In particular, as the main result, we obtain an optimal randomized algorithm for the problem of computing the trapezoidal decomposition determined by a set of N line segments in the plane with K pairwise intersections, that requires $\Theta(\frac{N}{B}\log_{M/B}\frac{N}{B}+\frac{K}{B})$ expected disk accesses, where M is the size of the available internal memory and B is the size of the block transfer. The approach is sufficiently general to obtain algorithms also for the problems of 3-d half-space intersections, 2-d and 3-d convex hulls, 2-d abstract Voronoi diagrams and batched planar point location, which require an optimal expected number of disk accesses and are simpler than the ones previously known. The results extend to an external-memory model with multiple disks. Additionally, under reasonable conditions on the parameters N, M, B, these results can be notably simplified originating practical algorithms which still achieve optimal expected bounds.

1 Introduction

There is a growing interest in algorithms working on sets of data that are too large to be fit in the *internal memory* of computers, and that consequently need to perform input/output accesses to *external storage devices*, like disks and CD-ROMs (see e.g. [4, 11, 19, 21, 29, 37]). These devices are roughly 10^6 times slower than internal memory in terms of access time. In many applications, this disparity has given rise to an input/output (or I/O) bottleneck, in which the time spent on moving data between internal and external memory dominates the overall execution time [20]. Such an I/O bottleneck is increasing in significance since the gap between the speed of (mechanical) disks versus (electronic) internal memories is growing, especially with the use of parallel computers [30]. Therefore, it is more than ever urgent to minimize the I/O communication in large-scale applications.

Computer graphics [33] as well as Geographic Information Systems [16, 23] are nowadays a rich source of large-scale computational problems. Here we need to design appropriate external-memory techniques and data structures that efficiently cope with the enormous amount of spatial data which are searched, stored and manipulated. In these applications, most of the subproblems require the processing of geometric primitives, so that the research on large-scale geometric algorithms is important and challenging.

Problems. A first goal of our paper is the design of an I/O-efficient algorithm for the *segment intersections* problem which consists of computing the *arrangement* of a set of N line segments in the plane with K pairwise intersections in an *output sensitive* manner. Computing the intersections I/O-optimally has been posed as an open problem in [5]. A second goal is to investigate the applicability of *random sampling* to the design of efficient geometric algorithms in the external-memory setting, and to develop a general randomized approach suitable to solve I/O-efficiently not only the segment intersections problem, but also several others geometric problems like convex hulls, Voronoi diagrams, batched planar point location.

We study these problems in the *external-memory model*, introduced in [37]. Here a computer consists of a processing unit, an internal memory of size M and an (unbounded) external memory partitioned into blocks of size B, $B \leq M$. Each access to the external memory transfers from/to the internal memory one block of B items, e.g., integers, pointers, characters. The goal is to design algorithms which take advantage of the block transfer and thus exhibit *locality of reference.* The complexity of an algorithm is then evaluated in this model by providing asymptotical bounds for the total number of disk accesses (I/Os) performed by the various operations, and for the number of internal operations executed (CPU time). For the sake of presentation we adopt the notation $n = N/B$, $m = M/B$, $k = K/B$.

Previous Work. Several (internal memory) algorithms for the segment intersections problem are known that execute an optimal number of (internal) operations $O(N \log N + K)$, both sequential (deterministic [9] and randomized [14, 26]) and parallel (deterministic [2] and randomized [13]). In the external-memory model, however, no I/O-optimal solution is known and the best algorithm has been devised by Arge et al. [5]. It is deterministic, involved, requires $O((n+k)\log_m n)$ I/Os, which is non-optimal, and computes the arrangement *only* for the case $k = 0$ (i.e., no crossings). Goodrich et al. [21] and Arge et al [5] presented some external-memory techniques that solve I/O-optimally other geometric problems. In particular, 2-d and 3-d convex hulls (using $O(n \log_m n)$ I/O operations), and the batched planar point location (using $O((n+k)\log_m n)$ I/O operations where now K is the number of queries). However, the resulting algorithms are complicated and require the use of several non-trivial I/O-optimal subroutines to solve some specific subproblems

like 3-d maxima and 2-d convex hull. Furthermore, the approaches are not easily extensible to other geometric problems, like the segment intersections problem in which we are interested.
Recent results [11, 35, 17] have shown a close relationship between parallel algorithms and external-memory algorithms, so that it is natural to look at the work on parallel geometric solutions to see if some of those results can be adapted to work efficiently in the external-memory model. The parallel algorithms cited above for the segment intersections problem are based on *random sampling* (the one in [2] is obtained via derandomization), and follow a divide-and-conquer approach (originated in [14]) in which a random sample is used to divide the problem into subproblems that are then solved independently. The main difficulty which arises with this approach is the need to bound the total size of the subproblems; as a result, some problem-dependent tricks are usually needed to achieve optimal results—like *filtering* [32] (also called *pruning*) and *vertex-accounting* [8]. The necessity of bounding the subproblems size was overcome in [13] by limiting the divide-and-conquer approach to only one level of recursion and using suboptimal algorithms for the induced subproblems. Among these parallel approaches, only the algorithm in [32] has been adapted so far to work in external memory by Goodrich et al. [21] thus achieving I/O-optimal solutions for the 3-d halfspace intersection problem (hence for 3-d convex hull and 2-d Euclidean Voronoi diagrams, see comments above).
Another approach based on random sampling is the random incremental construction, *RIC* (also originated in [14]). This approach adds the *objects* (e.g. segments) one at time, $S_i = S_{i-1} \cup \{s\}$, with the choice made at random, and updates the arrangement $\mathcal{T}(S_i)$ at each step accordingly. Although the RIC approach does not have the problem of controlling the total size of the subproblems, it is inherently sequential in its original formulation. However, there is a natural way to parallelize it using a *gradation*, that is, a geometrically growing random sequence of subsets of the input objects, $\emptyset = S_0 \subseteq \cdots \subseteq S_l = S$. In this case, $\mathcal{T}(S_i)$ is constructed from $\mathcal{T}(S_{i-1})$ by adding the objects belonging to $S_i - S_{i-1}$. Gradations have been mostly used to build data structures, as in the work of Mulmuley [27] and Mulmuley and Sen [28] (in fact, each *level* of the gradation is constructed from scratch using a regular RIC algorithm). Chazelle [8] and Brönnimann et al. [7] used a gradation approach as an intermediate step to derandomize the incremental construction of convex hulls, and this was followed by Amato et al. [2] to devise a parallel algorithm. The RIC approach with gradations has not received much attention otherwise, perhaps because sequentially it provides no advantage over the standard RIC algorithms, and in parallel it gives only expected bounds while there the emphasis is most often on high probability bounds.

Our Results. We show that the RIC approach can be adapted via *gradations* to provide a randomized incremental technique for external memory that results in expected I/O optimal algorithms for several geometric problems. The algorithms are also optimal in the expected number of internal operations performed (CPU time). By a suitable choice of the parameters of the gradation, the incremental step from $\mathcal{T}(S_{i-1})$ to $\mathcal{T}(S_i)$ can be performed by using efficient internal memory algorithms together with simple I/O efficient procedures—like straightforward movement of data in blocks or, in the most sophisticated case, external-memory sorting routines [29, 37].
As the main result, we obtain an I/O-optimal randomized algorithm for the segment intersections problem that requires $O(n \log_m n + k)$ expected I/Os. Since the algorithm necessarily computes the pairwise intersections among the input segments, this settles a question posed in [5]. The algorithm can also be adapted to handle degeneracies (particularly, to achieve output sensitivity with respect to the number of actual intersection points, not of pairwise intersections).
Our technique allows also to design algorithms with optimal expected I/O-bounds for the problems of 2-d and 3-d convex hulls, 2-d abstract Voronoi diagrams and batched point location in a planar

subdivision. Although I/O-optimal algorithms for these problems are already known (but for 2-d Voronoi diagrams, only in the Euclidean case), our RIC approach via gradations represents a uniform and simpler solution to all of them.[1]
Our algorithms apply to all values of the model parameters B and M, and problem size N. The more involved technicalities used in our algorithms are only needed to cope with extreme values of M, N and B, e.g., $B > M/\log^2 M$ and $\log n > (MB)^{1/4}$. For practical values of N, M and B, we show how to simplify the proposed algorithms in order to achieve an implementation requiring only modules that are, for example, available in LEDA ([25]), TPIE([36]), and LEDA-SM ([15]), like optimal internal memory algorithms for trapezoidal decompositions, point location in trapezoidal decompositions, topological sorting of graphs, and optimal external-memory algorithms for building, scanning, and sorting lists of constant-sized items.
Finally, our results also extend to the external-memory model with D parallel disks, in which an I/O-operation can transfer D disk blocks simultaneously, one from/to each disk. It seems likely that the approach is also efficient for a model with multiple CPU's and multiple disks, as the ones described in [17, 37].

Basic Preliminaries. The RIC approach is valid with great generality within the framework of *configuration spaces* [14, 27]. However, for the sake of simplicity, the presentation in Sections 2— 5 is restricted to the *segment intersections*, that is, computing the *trapezoidal decomposition* of the arrangement of a set S of N segments with a set $\mathcal{K}(S)$ of pairwise intersection points. Let $K = K(S) = |\mathcal{K}(S)|$. This decomposition is obtained by extending vertically each segment endpoint, and each intersection point between segments, upward and downward until it hits another segment; the trapezoids are the resulting connected regions of the plane. Given a subset $R \subseteq S$ of segments (*objects*), the set of resulting trapezoids (*cells*) is denoted by $\mathcal{T}(R)$. For

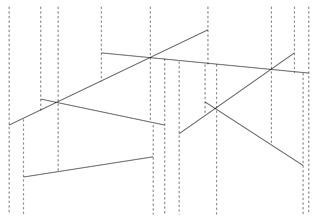

Figure 1: Trapezoidal decomposition

$\sigma \in \mathcal{T}(R)$, S_σ denotes the set of segments in S that intersect the interior of σ, and is called the *conflict list*. Let $N_\sigma = |S_\sigma|$. A *p-sample* R from S is obtained by choosing every $s \in S$ into R independently with probability p. [2] Note that the size of $\mathcal{T}(S)$ is $f(S) = |S| + K(S)$ (for simplicity of exposition, throughout the paper we ignore multiplicative constants if that only affects the multiplicative factor in the final bounds) and similarly for a p-sample R the expected size of $\mathcal{T}(R)$ is $f(p, S) = p|S| + p^2 K(S)$. There are two main properties of this sampling process that are relevant for the analysis of the algorithms [14, 12, 27]. First, *the average conflict list size*

[1] The external-memory RIC approach also extends to other higher dimensional problems like convex hulls, hyperplane arrangements, etc.. However, in these cases, the need for a *sorting step* leads to algorithms with I/O bounds which are suboptimal by a factor $O(\log_m n)$. For simplicity of exposition and importance/applicability of achieved results, we prefer to limit our presentation to the lower dimensional problems.

[2] The sampling result in Appendix A.2, as well the complexity of our algorithms, also hold in the sampling model where a subset R of size $r = pN$ is chosen at random among all subsets of that size.

is at most $1/p$. More precisely, for a constant $C > 0$:[3]

$$\mathbf{E}\left[\sum_{\sigma\in\mathcal{T}(R)} N_\sigma\right] \le C\frac{f(p,S)}{p}. \tag{1}$$

Second, *the deviation of the conflict list size is* $O(\log s)$ *with high probability.* More precisely, for $s \ge pN$, given $c > 0$ there is $C > 0$, such that with probability at least $1 - 1/s^c$:

$$\max_{\sigma\in\mathcal{T}(R)} N_\sigma \le C\ \frac{\log s}{p}. \tag{2}$$

The size function $f(p,S) = p|S| + p^2K(S)$ is also valid for the other problems we will investigate, by setting $K(S) = 0$. Though somewhat greater generality is possible, the final analysis that we present in Appendix B is restricted to this function. For the optimal algorithms under general conditions, the concept of a $(1/r)$*-cutting* for S is also needed [10, 24]:[4] *A decomposition of the underlying space into a set* T *of disjoint cells such that* $\max_{\sigma\in T} N_\sigma \le N/r$. In particular, the following fact is used (see Appendix A.3 for its proof and more details):

Fact 1.1 *There is a randomized algorithm that, given a set* S *of* N *objects, constructs a* $(1/r)$*-cutting for* S *of size* $O(r^c)$*, requiring* $O(r^cN)$ *expected operations, where c is a small constant depending on the problem.*

Contents. In Sections 2 and 3 we present the RIC approach via gradations and its implementation in the external memory model, using the segment intersections problem as a concrete example. In Section 4, we provide further algorithmic details for the segment intersections algorithm. In Section 5 we describe simpler algorithms under certain practical restrictions. In Section 6, we describe the changes needed to include other applications and describe three of them. In the last section, we indicate the extension to multiple disks. In the Appendix, we state some additional facts on sampling in configuration spaces, provide the analysis for the RIC approach via gradations, and we indicate how to handle degeneracies in the segment intersections problem.

2 RIC via Gradations

For the sake of simplicity, we present the RIC approach via gradations using the segment intersections problem as a concrete example. The extension to other problems will be clear throughout the paper, except for some problem dependent details that will be carefully discussed in Section 6. To facilitate the presentation we use the following additional notation: Given a set of objects X and a set of cells T, we write $T[X]$ to denote the set of conflict lists X_σ for $\sigma \in T$, and write $|T[X]|$ to denote $\sum_{\sigma\in T}|X_\sigma|$.

[3] This assumes $f(p/2,S) = O(f(p,S))$ which holds for all of our applications. A more general version is actually needed in the analysis and reviewed in the Appendix A.2.

[4] As shown in Sections 5.1 and 5.2, by means of the high probability bound in Eqn. (2), we can avoid the use of cuttings for a large, and practically reasonable, range of values of the parameters N, B, M. In these cases, we can obtain considerably simpler (practical) algorithms that still achieve optimal expected bounds. Thus, cuttings are a useful tool only for our theoretical results.

2.1 Gradation

The algorithm is a variant of the RIC approach [14] and follows [8, 7]. Given parameters μ and μ', it chooses a sequence of subsets of S, called a *gradation*:

$$\emptyset = S_0 \subseteq S_1 \subseteq \ldots \subseteq S_{l-1} \subseteq S_l = S$$

where S_{i-1} is a $(1/\mu)$-sample from S_i for $i < l$ and S_{l-1} is a $(1/\mu')$-sample from $S_l = S$.[5] Then, it iteratively constructs the decomposition $\mathcal{T}(S_i)$, for all i, $1 \leq i \leq l$. At the beginning of the i-th *round*, the decomposition $T_{i-1} = \mathcal{T}(S_{i-1})$ and its conflict lists $T_{i-1}[S]$ are available (initially, $S_0 = \emptyset$ and thus T_0 consists of a single "unbounded" cell having the whole S as conflict list), then the algorithm constructs the new decomposition $T_i = \mathcal{T}(S_i)$ by using T_{i-1} and $T_{i-1}[S]$. We refer to the l-th round as the *last* round, and to all the others as *early* rounds. Let R_i be $S_i - S_{i-1}$, the set of objects added in the i-th round, and for each cell $\sigma \in T_{i-1}$, let $R_{i,\sigma}$ be the subset of the objects in R_i which are conflicting with σ. Because of the random sampling, the sets $R_{i,\sigma}$ will be well balanced on the average. More precisely, in each early round, the average is at most μ, and in the last round the average is at most μ' (recall Eqn. (1)).

2.2 A Round

The i-th round computes T_i and $T_i[S]$ in three steps:

1. *Intermediate decomposition:* For each $\sigma \in T_{i-1}$, identify $R_{i,\sigma}$ by scanning S_σ and taking from it the segments which belong to R_i. Then compute the restriction of $\mathcal{T}(R_{i,\sigma})$ to σ, denoted T_σ. This results in an intermediate decomposition $T_i^I = \bigcup_{\sigma \in T_{i-1}} T_\sigma$.

2. *Intermediate conflict lists:* For each $\sigma \in T_{i-1}$, compute the conflict lists $T_\sigma[S]$ by taking each $s \in S_\sigma$ at a time and, after an *initial search* in T_σ that locates an endpoint of s, determine the conflicts of s with the cells in T_σ by appropriately walking through it. These are the intermediate conflict lists $T_i^I[S]$.

3. *Clean-up:* Obtain T_i and its conflict lists $T_i[S]$ from the intermediate decomposition T_i^I and its conflict lists $T_i^I[S]$. Observe that $\tau \in T_i$ can be *chopped* into pieces $\tau \cap \sigma$, for $\sigma \in T_{i-1}$. So we need to stitch together τ from its pieces $\tau \cap \sigma$ and also build its conflict list from the conflict lists of its pieces.

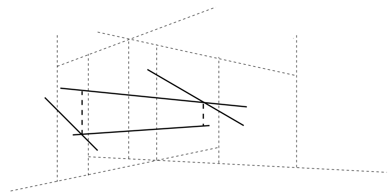

Figure 2: A chopped trapezoid

[5]The need for two parameters will become clear when discussing the external-memory implementation.

Remarks. (1) In Step 1, if $|R_{i,\sigma}|$ is $O(1)$ then a nonoptimal polynomial algorithm suffices. In our external-memory implementation, $|R_{i,\sigma}|$ is relatively large and hence, we will require the use of an optimal (internal memory) algorithm. (2) In Step 2, it is not true for all applications of our method that T_σ consists of cells $\tau \cap \sigma$ for $\tau \in \mathcal{T}(R_{i,\sigma})$; rather the decomposition is redefined within σ. As a result, in Step 3 it is not always the case that τ is obtained by stitching an example is given Section 6.1. (3) In Step 3, we need to recover $\mathcal{T}(S_i)$ so that we can apply the sampling results for S_i with respect to S. If the clean-up is not performed, then we can only use the sampling results *locally* in each cell for S_i with respect to S_{i-1} and we can not longer prove that the approach results in an optimal algorithm.

Analysis of internal running time. For simplicity, here as well as in later analysis, we ignore multiplicative constants in the bounds. The steps are implemented so that: (i) in Step 1, an optimal algorithm is used which requires $t_0(X) = |X| \log |X| + K(X)$ operations; (ii) in Step 2, the initial search is executed in $O(\log |R_{i,\sigma}|)$ operations by means of planar point location, and the walk to determine the conflicts is executed using an expected number of operations proportional to the number of retrieved conflicts (this is well-known [26]: the next trapezoid in the walk is determined by checking all the neighbors of the current trapezoid; the analysis at the end of Appendix C shows that the cost of this is as claimed); and finally (iii) Step 3 is executed in a number of operations which is proportional to the total size of the intermediate conflict lists $T_i^I[S]$, and to the total size of the resulting conflict lists $T_i[S]$. Consequently, the total number of operations required in the i-th round is

$$\sum_{\sigma \in T_{i-1}} \left(K(R_{i,\sigma}) + N_\sigma \log |R_{i,\sigma}| + \sum_{\tau \in T_\sigma} N_\tau \right). \tag{3}$$

The last round is simpler in that Step 2 is not needed at all, and in Step 3 no conflict lists are computed. In the expression above for the cost, only the first two terms are needed (note that $R_{l,\sigma} = S_\sigma$). In Appendix B, the expectation of Eqn. (3) is evaluated, and adding over all the rounds the following is shown.

Theorem 2.1 *The RIC approach via gradations solves the segment intersections problem using an optimal expected number of operations $O(N \log N + K)$.*

3 The Algorithm for External Memory

We now present an I/O-efficient implementation of the previous algorithm (recall the notation $m = M/B$, $n = N/B$ and $k = K/B$). First, as a technical point, we assume that the gradation is constructed before the algorithm starts, so that for each object $s \in S$ there is an associated *tag* that indicates the round in which s is *inserted*. The tag is carried by each copy of s (in each conflict list) so that, in the i-th round, the sets $R_{i,\sigma}$ can be easily determined by scanning the conflict lists S_σ. This is important in an efficient external memory implementation where we cannot assume random access to data.
The choice of parameters μ and μ' is done as follows: $\mu = m^{1/2}$ and $\mu' = \max\{B, M^{1/2}\}$. Therefore, the expected number of levels in the gradation is $l = O(\log_\mu(N/\mu')) = O(\log_m n)$.[6]. The main observations that lead to an I/O efficient implementation are:

[6]This also holds with high probability $1 - 1/n^c$, since $n(1/\mu)^{(c+1)\log_\mu n} = 1/n^c$, for a proper constant $c > 0$.

(i) The choice of μ' implies that $f(1/\mu', S) = O(f(S)/B)$. As a result, in each early i-th round, the algorithm does not need to handle T_{i-1} in an I/O-efficient manner: even incurring *one* I/O per operation of the internal memory algorithm is acceptable. Only the last round must handle T_l in an I/O-efficient manner, but then this is aided by the knowledge of T_{l-1}.

(ii) The choice of μ implies that in each early round the average size of T_σ is at most $\mu^2 = M/B$ (since the average size of $R_{i,\sigma}$ is at most μ). Therefore, ignoring deviations, T_σ can be computed in internal memory and one block of memory allocated for each cell of T_σ, allowing the conflict lists to be written in an I/O-efficient manner. Similarly, in the last round, the choice of μ' implies that, also ignoring deviations, the average size of $R_{l,\sigma} = S_\sigma$ is at most μ', for each $\sigma \in T_{l-1}$. If $B^2 \leq M$ then $\mu' = M^{1/2}$ and an optimal internal memory algorithm can be used to construct T_σ, because the decomposition fits in internal memory. If $B^2 > M$ then $\mu' = B$ and T_σ can be larger than M, so that an I/O-optimal algorithm must be devised to handle small inputs of size less than B.

To simplify the presentation, we describe the algorithm in two stages. First, we ignore the deviations from the average values of $|R_{i,\sigma}|$ and describe an algorithm that performs an optimal number of I/Os. Specifically, we assume $|R_{i,\sigma}| \leq \mu$ for $i < l$ and $|R_{l,\sigma}| \leq \mu'$. Then, using a refinement approach of Chazelle and Friedman [10], we obtain an algorithm that works without any assumptions on the $R_{i,\sigma}$'s, at the cost of complicating the algorithm. In Section 5, we will show that the original simpler but incomplete algorithm actually leads to a solution achieving optimal I/O bounds under certain reasonable (i.e., practical) conditions on N, M and B.

3.1 Ignoring Deviations

We discuss the I/O operations needed to implement the basic algorithm of Section 2.2 in external memory under the hypothesis that $|R_{i,\sigma}| \leq \mu$ for $i < l$ and $|R_{l,\sigma}| \leq \mu'$. We will refer later to this algorithm as Algorithm **I**. The number of internal memory operations is the same as the one analyzed in Theorem 2.1. As before, to simplify the equations, we ignore multiplicative constants.

3.1.1 The Early Rounds

We detail below the external-memory implementation of the three steps forming the i-th early round in the basic algorithm.

Step 1. Each cell $\sigma \in T_{i-1}$ in turn, and its conflict list S_σ, are loaded in *internal memory* so that $R_{i,\sigma}$ is determined by checking the tags and $T_\sigma = \mathcal{T}(R_{i,\sigma})$ is computed. Since the size of T_σ is at most $\mu^2 \leq M/B$, then T_σ can be constructed by an optimal internal-memory algorithm without performing any I/Os. Thus, the number of I/Os required by this step is:

$$\sum_{\sigma \in T_{i-1}} \left(\left\lceil \frac{N_\sigma}{B} \right\rceil + \left\lceil \frac{|R_{i,\sigma}|}{B} \right\rceil + \left\lceil \frac{|T_\sigma|}{B} \right\rceil \right) = O\left(|T_{i-1}| + \frac{|T_{i-1}[S]|}{B} + \frac{|T_i^I|}{B} \right).$$

Step 2. By the hypothesis $|T_\sigma| \leq \mu^2 = M/B$, hence we can reserve in internal memory a buffer of size B for each cell $\tau \in T_\sigma$. The conflict lists $T_\sigma[S]$, for $\sigma \in T_{i-1}$, are computed by scanning S_σ and walking through T_σ in internal memory. As a conflict of s with some τ is determined, it is written into the buffer associated with τ. As a buffer becomes full, it is written to external memory. In this way, the number of I/Os is proportional to the size of the scanned and returned conflict lists, divided by B. Therefore, the overall number of I/Os required by this step is:

$$\sum_{\sigma \in T_{i-1}} \left\lceil \frac{N_\sigma}{B} \right\rceil + \sum_{\sigma \in T_{i-1}} \sum_{\tau \in T_\sigma} \left\lceil \frac{N_\tau}{B} \right\rceil \le \frac{|T_{i-1}[S]|}{B} + \frac{|T_i^I[S]|}{B} + |T_i^I| + |T_{i-1}| = O\left(\frac{|T_i^I[S]|}{B} + |T_i^I|\right).$$

Step 3. We have the intermediate decomposition T_i^I and we need to determine the decomposition T_i. As noted earlier, a trapezoid $\tau \in T_i$ may occur in T_i^I chopped into pieces $\tau \cap \sigma$, for $\sigma \in T_{i-1}$. To achieve optimal bounds, the stitching of pieces $\tau \cap \sigma$ has to be performed using a number of I/Os proportional to the total number of blocks of size B required to hold the conflict lists $T_i^I[S]$, and the resulting conflict lists $T_i[S]$. Achieving this goal presents some difficulties because we cannot afford to use sorting since this would lead to a suboptimal algorithm paying an extra-factor $\log_m n$ in the final I/O-complexity. Our approach is to first consider the partial ordering between trapezoids induced by their vertical adjacencies; then, compute a linear order by *topologically sorting* that partial order; and finally, traverse the decomposition T_{i-1} by following this linear order. A difficulty is that we do not know how to topologically sort with a "linear" number of I/Os, that is, $|T_{i-1}|/B$. Fortunately, $|T_{i-1}|$ is "small" so that one can afford to use an optimal internal-memory algorithm that even perform one I/O per internal memory operation. As T_{i-1} is traversed, the chopped trapezoids of T_i are put together by maintaining a list of those that cross the right vertical boundary of the trapezoid in T_{i-1} currently considered. This list is matched to the list of chopped trapezoids in T_i that cross the left vertical boundary of the adjacent trapezoid in T_{i-1} when it comes under consideration. Further details are given in Section 4.1.

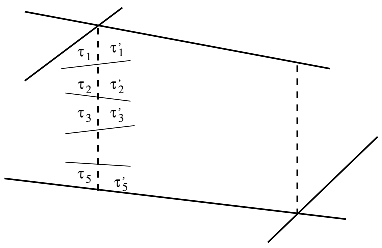

Figure 3: Trapezoids crossing a vertical edge

The conclusion is that Step 3 can be performed using a number of I/Os which is proportional to the total number of blocks required to hold the conflict lists $T_i^I[S]$ and the resulting conflict lists $T_i[S]$, plus the I/Os needed for the topological sort (i.e., $|T_i^I|$):

$$\sum_{\sigma \in T_{i-1}} \sum_{\tau \in T_\sigma} \left\lceil \frac{N_\tau}{B} \right\rceil + \sum_{\sigma \in T_i} \left\lceil \frac{N_\sigma}{B} \right\rceil \le \frac{|T_i^I[S]|}{B} + \frac{|T_i[S]|}{B} + |T_i^I| + |T_i| = O\left(\frac{|T_i^I[S]|}{B} + |T_i^I|\right).$$

Essentially, Steps 1–3 require an *I/O-efficient* handling of the conflict lists $T_{i-1}[S]$, $T_i^I[S]$; in fact, we devised external-memory algorithms that manage these lists by executing a number of I/Os proportional to their sizes divided by the block size B. On the other hand, Steps 1–3 allow us to handle in an *I/O-inefficient* way the decompositions T_{i-1}, T_i^I and T_i; in fact, we employed *inefficient* external-memory algorithms that manage those decompositions by executing a number of I/Os which is proportional to their whole size.

3.1.2 The Last Round

The goal is now to compute the final decomposition $T_l = \mathcal{T}(S)$ from T_{l-1} in a reduced number of I/Os. We recall that S_{l-1} is a $(1/\mu')$-sample from S, and we assumed that $|R_{l,\sigma}| \le \mu'$. We need an optimal algorithm that handles the small case, that is, it computes $\mathcal{T}(R_{l,\sigma})$ for $|R_{l,\sigma}| \leq \mu' \leq M$ requiring $t_1(R_{l,\sigma}) = (N_\sigma/B)\log_m(N_\sigma/B) + K_\sigma/B$ I/Os, where $N_\sigma = |R_{l,\sigma}|$ and $K_\sigma = |\mathcal{K}(R_{l,\sigma}) \cap \sigma|$ (that is, K_σ is the number of pairwise intersections of $R_{l,\sigma}$ inside σ). This is trivial if $K_\sigma = 0$ because we can use an internal-memory algorithm that requires linear space. But when this is not the case $\mathcal{T}(R_{l,\sigma})$ might not fit in internal memory at once and a proper *optimal external-memory* algorithm is needed. The algorithm for such a "small case" can be obtained again using sampling: Take a sample of size $\sqrt{M}$, compute its decomposition using an internal memory algorithm, compute its conflict lists, compute each of the resulting subproblems again in internal memory, and finally put together the result. Further details are given in Section 4.2 where it is shown that the resulting algorithm requires $t_1(R_{l,\sigma})$ expected I/Os (note that this expectation is over the additional randomization required by this small case algorithm, not the randomization introduced by the gradation). Thus, the last round consists of the following substeps:

l.1 For each trapezoid $\sigma \in T_{l-1}$, compute its decomposition T_σ according to the segments $R_{l,\sigma} = S_\sigma$ using the small size case algorithm (Section 4.2), since $N_\sigma \leq \mu' \leq M$.

l.2 Obtain $\mathcal{T}(S)$ from the collection T_σ, for all $\sigma \in T_{l-1}$.

As already pointed out, Step l.1 requires $t_1(R_{l,\sigma})$ expected I/Os for each $\sigma \in T_{l-1}$. Summing over all the trapezoids in T_{l-1}, and taking expectation, the expected number of I/Os is $O(n \log_m n + k)$. [7]
Finally, Step l.2 is indeed exactly the same as Step 3 of the i-th early round (above), with the further simplification that now there are no conflict lists to merge. We therefore just need to stitch appropriately the trapezoids in the collections T_σ's, for $\sigma \in T_{l-1}$. Hence, the I/O-cost is $O(|T_l^I|/B + |T_{l-1}|)$.
We note that in the discussion above we ignored the deviations, so that we are actually assuming that the internal memory can always accommodate $R_{i,\sigma}$, its decomposition and corresponding buffers. In this situation (and proceeding with the same analysis adopted for the basic algorithm in Appendix B), we can prove the following result:

Theorem 3.1 *Let us assume that the internal memory can always accommodate $R_{i,\sigma}$, its decomposition T_σ and the corresponding $|\mathcal{T}(R_{i,\sigma})|$ buffers of size B. Then Algorithm* **I** *solves the segment intersections problem using $O(n \log_m n + k)$ expected I/Os, which is optimal.*

The algorithm can be modified to handle degeneracies and still achieve optimal complexity: $O(n \log_m n + i)$ where $i = I/B$ and I is the number of intersection points (which can be much smaller than the number of pairwise intersections). See Appendix C for details.

3.2 Handling Deviations

Now we show how to remove the assumptions made on $|R_{i,\sigma}|$, and cope with its *deviation*. We will refer to this algorithm as Algorithm **H**. Specifically, this is achieved by using a refinement approach of Chazelle and Friedman [10, 24] (see Fact A.2 in Appendix A.3 for details). For $i < l$, the idea is to use Fact 1.1 and hence construct a $(1/t_\sigma)$-cutting $\hat{T}_\sigma$ for $R_{i,\sigma}$ restricted to σ, for each $\sigma \in T_{i-1}$, where $t_\sigma = |R_{i,\sigma}|/\mu$ is called the *excess* of σ. Since each cell $\tau \in \hat{T}_\sigma$ satisfies the desired constraint on the size of its conflict list (i.e. $\leq \mu$), we can apply to it the Steps 1 and 2 of the i-th round in Algorithm **I**. Then, to take care of the *independent* refinement performed by the cutting process on each cell $\sigma \in T_{i-1}$, we perform *two levels of clean-up*; both levels are similar

[7] Using Equation (5) in Appendix A.2 with $g(x) = (x/B)\log_m(x/B)$.

to Step 3 in Algorithm **I**. The first clean-up obtains T_σ and $T_\sigma[S]$ from the T_τ's and $T_\tau[S]$'s with $\tau \in \hat{T}_\sigma$, and the second one obtains the final decomposition $T_i = \mathcal{T}(S_i)$ and its conflict lists $T_i[S]$ from T_i^I and $T_i^I[S]$.[8] Similarly, for the last round.

Complete i-th round. The modified algorithm for the i-th early round (i.e., $i < l$) is described below. For simplicity of exposition, we also specify between square brackets the small changes which are needed to get the modified last round. Note that Step c.2.1 (and Step c.1 in the last round) use an algorithm for the special case $N \leq M$ which is described in Section 4.2.

- c.1. For each $\sigma \in T_{i-1}$, use Fact 1.1 to construct a $(1/t_\sigma)$-cutting $\hat{T}_\sigma$ for $R_{i,\sigma}$ restricted to σ, where $t_\sigma = |R_{i,\sigma}|/\mu$. Thus each $\tau \in \hat{T}_\sigma$ satisfies the desired relation $|R_{i,\tau}| \leq \mu$. Additionally, compute the conflict lists S_τ in $\hat{T}_\sigma[S]$ by checking each $s \in S_\sigma$ versus each $\tau \in \hat{T}_\sigma$. This computation allows also to determine the list $R_{i,\tau}$. [In the last round, we have $t_\sigma = |R_{l,\sigma}|/\mu'$ so that $|R_{l,\tau}| \leq \mu'$.]
- c.2. For each $\sigma \in T_{i-1}$ do the following:
  - c.2.1. For each $\tau \in \hat{T}_\sigma$, load $R_{i,\tau}$ in internal memory and compute $T_\tau = \mathcal{T}(R_{i,\tau})$. [In the last round, apply the small size case algorithm (see Section 3.1.2).]
  - c.2.2. For each $\tau \in \hat{T}_\sigma$, compute the conflict lists $T_\tau[S]$, by scanning S_τ and walking in T_τ. [In the last round, this step is not executed.]
  - c.2.3. Obtain $T_\sigma = \mathcal{T}(R_{i,\sigma})$ and its conflict lists $T_\sigma[S]$ from the collection of T_τ's and their conflict lists $T_\tau[S]$, where $\tau \in \hat{T}_\sigma$. [In the last round, we just obtain T_σ.]
- c.3. Obtain T_i and its conflict lists $T_i[S]$ from the intermediate decomposition T_i^I and its conflict lists $T_i^I[S]$. [In the last round, we just obtain $T_l = \mathcal{T}(S)$.]

Steps c.2.1–c.2.3 are just like the corresponding Steps 1–3 of Algorithm **I**, and Step c.3 here is like the corresponding Step 3 of Algorithm **I**. From Fact 1.1, the size of a $(1/t_\sigma)$-cutting for S_σ is $O(t_\sigma^c)$ and the expected number of required operations to construct it is $O(t_\sigma^c N_\sigma)$. Consequently, it turns out to be essential for the overall I/O-efficiency of our complete algorithm that *on the average* t_σ^c behaves *as a constant* when multiplied by N_σ^c, by $|R_{i,\sigma}|$ and by $\sum_{\tau \in T_\sigma} N_\tau$. More precisely, the following relations are proved in Appendix B.1 (here $\mathbf{E}_i$ and $\mathbf{E}_{i-1,i}$ denote expectations over the random choices of S_i, and of both S_{i-1}, S_i respectively, see Appendix B).

Lemma 3.2 *The following relations hold for $f(p,S) = p|S| + p^2 K(S)$ and positive constants c, e:*

(i) $\mathbf{E}_{i-1,i}\left[\sum_{\sigma \in T_{i-1}} t_\sigma^c N_\sigma^e\right] \leq \frac{1}{q_{i-1}^e} f(q_{i-1}, S)$.

(ii) $\mathbf{E}_{i-1,i}\left[\sum_{\sigma \in T_{i-1}} t_\sigma^c |R_{i,\sigma}|\right] \leq \mu f(q_{i-1}, S)$ *for* $i < l$ *and* $\mu' f(q_{l-1}, S)$ *for* $i = l$.

(iii) $\mathbf{E}_{i-1,i}\left[\sum_{\sigma \in T_{i-1}} t_\sigma^c \sum_{\tau \in T_\sigma} N_\tau\right] \leq \frac{1}{q_i} f(q_i, S)$

[8] In some other applications, e.g. 3-d halfspace intersection, the clean-up can be easily performed in one level.

The important point is that the upper bounds on the right side of the inequalities are the same as for the quantities on the left side without the t_σ factors. These relations allow us to prove that, within a constant factor, the I/O cost of Algorithm **H** is the same as for Algorithm **I**. We verify this for the i-th round as follows.

Step c.1: Following the proof of Fact 1.1 in Appendix B, the decomposition $\hat{T}_\sigma$ and its conflict lists with $R_{i,\sigma}$ can be computed in $O(t_\sigma^c(|R_{i,\sigma}|/B+1))$ expected I/Os. Lemma 3.2(ii) shows that this I/O-bound is hidden by the overall cost of the algorithm. Lemma 3.2(i), with $e=1$, shows that we can also afford to compute the conflict lists with S_σ by means of a brute-force approach.

Step c.2: Lemma 3.2(i-iii) shows that the I/O bounds obtained for Step 2 of Algorithm **I** also apply to Steps c.2.1–c.2.3 of Algorithm **H**. For example, Lemma 3.2(iii) gives an upper bound on the expected size of the conflict lists S_τ, for $\tau \in \hat{T}_\sigma$, which is within a constant factor of the expected size of the conflict lists S_τ, for $\tau \in T_\sigma$ (note that $\sum_{\tau\in\hat{T}_\sigma} N_\tau \le t_\sigma^c \sum_{\tau\in T_\sigma} N_\tau$).

Step c.3: This is the same as Step 3 of Algorithm **I**, so its I/O-bound also applies here.

Theorem 3.3 *Algorithm* **H** *solves the segment intersections problem using* $O(n\log_m n + k)$ *expected I/Os, which is optimal.*

4 Details on the Segment Intersections Algorithm

In the previous section, some important details of the segment intersections algorithm were omitted because they were too specific and distracting from the goal of presenting the main ideas underlying our approach. In this section we provide those details; specifically, we discuss the implementation of the clean-up step and the small-size case algorithm.

4.1 Clean-up Step

Let us assume that the graph G_{i-1} describing the vertical adjacencies between the trapezoids of T_{i-1} is available (a *node* in G_{i-1} corresponds to a trapezoid in T_{i-1} and an *arc* in G_{i-1} corresponds to an adjacency through a vertical edge between two trapezoids in T_{i-1}, see Fig. 4). Let us consider an arc in G_{i-1} directed according to the left to right ordering of the corresponding trapezoids. This is an acyclic ordering, so we can choose a linear extension (i.e., *topological sort*) and traverse G_{i-1} according to that order. Usually, performing such an ordering on a generic graph is not I/O-efficient, but here we exploit the fact that in these earlier rounds ($i < l$), the underlying graph G_{i-1} has *small size* and thus we can afford to pay even *one* I/O per visited node. Hence, we can compute this ordering by adopting a standard internal-memory algorithm which pays *one I/O* per step, and thus requires a number of I/Os linear in the size of the graph G_{i-1}, that is $|G_{i-1}| = |T_{i-1}|$ I/Os (this term is hidden by the next I/O-bounds). We can then proceed to visit

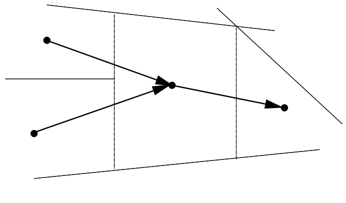

Figure 4: Adjacency graph

the trapezoids according to the induced order. At a generic step of the traversal, we say that an arc of G_{i-1} is *waiting* if its starting node has been visited but its ending node is yet to be visited;

we also say that the corresponding vertical edge is waiting and that the trapezoid currently under consideration is *active*. We assume that the trapezoids τ in T_σ are stored with a particular order: First appear those trapezoids that touch the left vertical edge in order from top to bottom, then those that do not touch the vertical edges, and then those that touch the right vertical edge from top to bottom. During the visit of G_{i-1}, as the next active trapezoid $\sigma \in T_{i-1}$ is considered, we take the (ordered) list of trapezoids which are still *not completed* and thus are associated with the one or the two waiting vertical edges e_1 and e_2 (the neighbors of σ on the left). We merge these lists with the (ordered) list of trapezoids of T_σ, by taking $O(((|T_\sigma|/B)+1)$ I/Os, since the first two lists are surely smaller than the latter. As a result, some new trapezoids in T_i are completed, some are completely inside σ, some other are started, and still others continue (i.e., stab σ). The not completed trapezoids are stored on the disk, associated with the (at most two) waiting vertical edges to the right of σ (thus preserving the invariant). Therefore, the overall I/O-cost of merging trapezoids in the T_σ's to get T_i is $\sum_{\sigma \in T_{i-1}}((|T_\sigma|/B)+1)$.
Within the same I/O-bound, it is possible to compute the next graph G_i, thus preserving the invariant. We are left with the problem of computing the conflict lists $T_i[S]$, which are large and thus must be managed properly. As before, a sorting operation would not yield the optimal bound. Instead, we exploit the geometric structure of the problem and proceed as follows. We assume that the trapezoids $\tau \in T_\sigma$ are stored in the particular order described before and now have also associated their conflict lists. The algorithm visits G_{i-1} according to the induced order. When a trapezoid $\sigma \in T_{i-1}$ becomes active, the algorithm proceeds by matching the trapezoids in the one or the two corresponding waiting vertical edges e_1 and e_2 (the neighbors of σ on the left), and updating their conflict lists. Namely, if τ was incomplete in e_i and matches with $\tau' \in T_\sigma$, then we *merge* τ and τ' by augmenting the conflict list of τ with those conflicts of τ' that do not intersect e_i. These conflicts are guaranteed to refer to *new* segments which don't intersect τ and thus have been not yet inserted in its conflict list. This approach allows to merge the conflict lists without *rescanning* their segments over and over. Clearly, we must pay at least *one* disk access per trapezoid in T_σ but this is not much of a problem due to the small size of G_{i-1} (see considerations above). In conclusion, the number of I/Os necessary to do this merging process is bounded by the number of trapezoids in T_{i-1}, in T_i^I and in T_i, and by the size of all the corresponding conflicts divided by B. Thus, Step 3 can be performed using a number of I/Os which is proportional to the total number of blocks of size B required to hold the conflict lists $T_i^I[S]$, and the resulting conflict lists $T_i[S]$.

4.2 Small Size Case Algorithm

We consider a set of segments S of size $N \leq M$ and with K pairwise intersections (please note that the variables S, N and K are local to this subsection). This computation is non-trivial since K can be as large as M^2 and, hence, we cannot afford to even construct $\mathcal{T}(S)$ completely in internal memory. For simplicity, in describing the algorithm we will ignore (in Step 1 below) the deviations in the conflict list sizes resulting from sampling. We can nevertheless cope with them by adopting the *cutting* technique as we have done for the general algorithm in Section 3.2.

1. Take a p-sample R from S where $p = 1/\sqrt{M}$ and compute $T_0 = \mathcal{T}(R)$ in internal memory. As already indicated, we ignore the deviations and thus $\max_{\sigma \in \mathcal{T}_0} N_\sigma \leq 1/p = \sqrt{M}$. Also compute the adjacency graph G_0 of T_0 and a topological sort L_0 (in internal memory).

2. For each $s \in S$ compute the conflicts with T_0 by walking on it in internal memory and by storing the conflicts on the disk as they are determined. Then sort all these conflicts to bring together the ones corresponding to S_σ, for each $\sigma \in T_0$, according to ordering L_0.

3. For each $\sigma \in T_0$ compute $T_\sigma = \mathcal{T}(S_\sigma)$ in internal memory (see assumption in Step 1).
4. Obtain $\mathcal{T}(S)$ from the collection of T_σ's, where $\sigma \in T_0$: First, traverse G_0 according to the ordering L_0, and assign addresses to the final trapezoids $\tau \in \mathcal{T}(S)$ by loading in turn the T_σ's (if a trapezoid $\tau \in \mathcal{T}(S)$ is divided into many pieces in T_0, then only the piece appearing first is given an address). Then, the different pieces of each $\tau \in \mathcal{T}(S)$ are brought together using sorting. At this point, all the pieces of $\tau \in \mathcal{T}(S)$ can find out the final address of τ by a simple scan. Subsequently, we undo the sort so that each trapezoid can determine the address of its neighbors. This information is propagated to all trapezoid pieces by again sorting, scanning and undoing the sorting. Finally, a traversal of G_0 according to the ordering L_0, is used to write the resulting adjacency graph of $\mathcal{T}(S)$.

We elaborate further on the use of sorting in Steps 2 and 4. In Step 2, each determined conflict is made into a pair indicating the conflicting segment and the position of the conflicting trapezoid in L_0. Sorting these pairs brings together the conflicts belonging to the same trapezoid of T_0 according to the ordering L_0. In Step 4, for each $\sigma \in T_0$ and each chopped $\tau \in T_\sigma$, we set up a quadruple consisting of the top segment of τ, the bottom segment of τ, the x-coordinate of the left boundary of τ, and the x-coordinate of the right boundary of τ. Sorting these quadruples brings together all pieces that correspond to the same trapezoid $\tau \in \mathcal{T}(S)$, and moreover brings these pieces in their left to right ordering.
We point out that we can afford to sort segments and trapezoids here, because the two present steps are executed only once (in the last round). Hence, the I/O-complexity of sorting is hidden by the overall I/O-cost of the whole algorithm as we formally prove below.
We show that the total expected number of I/Os is $O(n \log_m n + k)$. The critical part is to verify that the sort in Steps 2 and 4 can be performed within this bound. Indeed, the number of sorted items is equal to the number of conflicts and its expected value is $N + (1/\sqrt{M})K$. Hence, using multiway mergesort [37], the expected number of I/Os is (ignoring constants):

$$(n + k/\sqrt{M}) \log_m (n + k/\sqrt{M}) = n \log_m (n + k/\sqrt{M}) + k/\sqrt{M} \log_m (n + k/\sqrt{M}).$$

First note that the second term on the right side is always $O(k)$ because N is assumed to be smaller than M, K is then smaller then M^2 and also we have $\log_m (n + k/\sqrt{M})) = O(1)$. This term is dominant if $k/\sqrt{M} > n$. On the other hand, if $k/\sqrt{M} \leq n$, then the first term is $O(n \log_m n)$. Therefore, the expected number of I/Os is indeed $O(n \log_m n + k)$.

5 Simplified Algorithms Under Practical Conditions

5.1 First Version

We want to obtain an algorithm that achieves the optimal I/O bound but does not make use of cuttings. Such a simpler algorithm can be designed as a slight variant of Algorithm **I** working under some reasonable (and practical) conditions on the parameters N, M and B. These conditions are not very restrictive because they hold for current computers and applications. The new algorithm, called Algorithm $\mathbf{I'}$, is derived from Algorithm **I** as follows: (i) the gradation parameters are $\mu = m^{1/4}/2C$ and $\mu' = M/(2C^2 \log M \log n)$, where C is determined by Eqn. (2) with $c = 2$; (ii) if at the generic i-th round $|R_{i,\sigma}|^2 > m$, then the new algorithm performs as many I/Os as they are necessary to access data that do not fit in internal memory (even paying one I/O per internal memory operation); (iii) the small size case is now $N \leq M/C \log M$ and the algorithm used here is the one described in Section 3.1.2 (without the use of cuttings as the resulting subproblems have size at most $\sqrt{M}$). The following lemma establishes conditions for N, M, B under which Algorithm $\mathbf{I}'$ performs, within a constant factor, the same expected number of I/Os as Algorithm $\mathbf{I}$.

Lemma 5.1 *Algorithm* $\mathbf{I}'$ *solves the segment intersections problem using an optimal expected number of I/Os, under the conditions* $\max\{6C^2 \log C, B^{1/2}\} \leq n$, $B \leq M/\log^2 M$ *and* $\sqrt{\log m}/C^2 \leq \log n \leq m^{1/4}$.

Proof. First, note that $\mu' \geq 1$ when $\log n \leq M/2C^2 \log M$ (this is weaker than the conditions imposed below). For the i-th round, let us denote by I_i (resp. I_i') the expected number of I/Os performed by Algorithm $\mathbf{I}$ (resp. Algorithm $\mathbf{I}'$), and let W_i be the expected number of internal operations performed by both algorithms. If ρ_i is the probability that condition (ii) above does not hold during the i-th round (i.e., the probability that the algorithm starts executing an I/O for each internal memory operation), then $I_i' \leq I_i + \rho_i W_i$. Therefore, adding over all the rounds, we get $I' \leq I + \sum_i \rho_i W_i \leq I + \alpha B I \max_i \rho_i$, where $\alpha = \log N/\log_m n$ (using $I = n \log_m n + k$ and $W = N \log N + K$). So we only need to establish the conditions under which $\max_i \rho_i \leq 1/\alpha B$ holds, and then $I' = O(I)$ will immediately follow.
We need to evaluate ρ_i, and thus we notice that $1 - \rho_i$ (resp. $1 - \rho_l$) is the probability that $|R_{i,\sigma}| \leq m^{1/2}$ (resp. $|R_{l,\sigma}| \leq M/C \log M$). Now, since S_{i-1} is a $(1/\mu)$-sample (resp. $(1/\mu')$-sample) from S_i, using Eqn. (2) with $c = 2$, we obtain that $|R_{i,\sigma}| \leq C\mu \log s$ (resp. $|R_{l,\sigma}| \leq C\mu' \log s$) with probability at least $1 - 1/s^2$, provided that $s \geq |S_i|/\mu$ (resp. $s \geq |S_l|/\mu' = N/\mu'$). Hence $\rho_i \leq 1/s^2$ if we ensure that $|R_{i,\sigma}| \leq m^{1/2}$ (resp. $|R_{l,\sigma}| \leq M/C \log M$) and $s \geq |S_i|/\mu$ (resp. $s \geq |S_l|/\mu' = N/\mu'$).
As far as the conditions on s are concerned, it suffices that we choose $s = 2C^2 n \log n$ and assume $B \leq M/\log M$ (which is weaker than the following conditions). As far as the conditions on $|R_{i,\sigma}|$ and $|R_{l,\sigma}|$ are concerned, it suffices that we impose $C\mu \log s \leq m^{1/2}$, $C\mu' \log s \leq M/C \log M$. By assuming $n \geq 6C^2 \log C$ (hence, it is $2 \log n \geq \log s$) and assuming $\log n \leq m^{1/4}$, both the above conditions hold.
It remains to fix conditions for which we have $\rho_i \leq 1/(\alpha B)$. Since $\rho_i \leq 1/s^2$, it suffices to impose $B^{1/2} \leq n$ (which actually implies $\log N \leq 3 \log n$) and $\log n \geq (1/C^2)\sqrt{\log m}$.
Still, we need to ensure that the choice of μ' does not affect the optimality of the algorithm because the size of T_{l-1} becomes too large (since we allow a number of I/Os linear in the size of T_i when $i \leq l-1$). So we need $N/\mu' + K/\mu'^2 \leq 2C^2(n \log_m n + k)$ or, splitting into two inequalities, $B \leq 2C^2 \mu' \log_m n$ and $B \leq 2C^2 \mu'^2$. These inequalities hold by imposing the additional conditions $B \leq M/\log^2 M$ and $2C \log n \leq M^{1/2}$ (the latter is weaker than $2 \log n \leq m^{1/4}$, assuming C is not too large). Putting together all the conditions imposed above, gives the statement of the theorem.
Finally, we observe that the small size case $N \leq M/C \log M$ does not need to use cuttings because for a sample of size $\sqrt{M}$, each of the trapezoids in its decomposition have conflict list size at most $C(M/C \log M) \log M/\sqrt{M} = \sqrt{M}$ with probability at least 1/2, so we can repeat the sampling until a good one is obtained. ■

5.2 Improved Conditions

We take advantage of the averaging results on the excess t_σ, to modify Algorithm $\mathbf{I}'$ so that it performs well under less restrictive conditions. The new Algorithm $\mathbf{I}''$ proceeds as Algorithm $\mathbf{I}'$ with the exception that in Step 2 of the early rounds it uses buffers of size $b_\sigma = \max(1, B/t_\sigma^2)$, instead of B, to produce the conflict lists $T_\sigma[S]$ (see Section 2.2). All these buffers can be allocated in internal memory as long as $|R_{i,\sigma}|^2 b_\sigma \leq M$. Since $|R_{i,\sigma}|^2 B/t_\sigma^2 \leq M$, it suffices that $|R_{i,\sigma}|^2 \leq M$.

If this is the case, since t_σ^2 behaves as a constant on the average, the expected total number of I/Os required by Step 2 is equal to the one required by Algorithm $\mathbf{I}'$. If $\mathcal{T}(R_{i,\sigma})$ does not fit in internal memory, the algorithm performs a large number of I/Os because it can possibly execute one I/O per internal-memory operation.

Theorem 5.2 *Algorithm $\mathbf{I}''$ solves the segment intersections problem using an optimal expected number of I/Os, under the conditions* $\max\{6C^2 \log C, B^{1/2}\} \leq n$, $B \leq M/\log^2 M$ *and* $\sqrt{\log m}/C^2 \leq \log n \leq (MB)^{1/4}$.

Proof. Lemma 3.2 is used to conclude that reducing the size of the buffers to b_σ does not affect the expected number of I/Os performed by Step 2. Therefore, as long as the internal memory can hold $R_{i,\sigma}$ and its decomposition, that is as long as $|R_{i,\sigma}| \leq M^{1/2}$, the expected total number of I/Os is equal to that for Algorithm $\mathbf{I}$ within a constant factor.
The effect of $|R_{i,\sigma}| \leq M^{1/2}$ not holding, is analyzed following an argument similar to the one in the proof Lemma 5.1. We want $C\mu \log s \leq M^{1/2}$ (in the early rounds) and $C\mu' \log s \leq M/C \log M$ (in the last round). Since we set $s = 2C^2 n \log n$ and we assumed $n \geq 6C^2 \log C$, it is $2 \log n \geq \log s$, so that from the first requirement we find the condition $\log n \leq M^{1/2}/m^{1/4} = (MB)^{1/4}$. The other conditions remain the same as the ones stated in Lemma 5.1. ■

The last condition is a considerable improvement over the one in Lemma 5.1 for Algorithm $\mathbf{I}'$. The conditions hold for the values of N, M, B in current computers and applications.

6 General Algorithm and Other Applications

The RIC approach via gradations extends to many other problems in the framework of configuration spaces [14, 27], see Appendix A.1. The outline of the basic algorithm is the same as that in Section 2. In most cases the resulting algorithm is optimal in the expected number of (internal) operations performed. The corresponding external memory implementation is also just as outlined in Section 3 (Algorithms $\mathbf{I}$ and $\mathbf{H}$), except that the choice of the parameters μ and μ' needs to be modified. Only the clean-up in Step 3 requires some additional comments.

This clean-up step is very problem dependent. In general, it requires some data movement on T_i^I that must be performed efficiently. This may require operations like integer sorting, connectivity, and graph traversal that are usually performed in $O(X)$ operations when operating internal memory, but cannot be performed with $O(X/B)$ I/Os [11]. However, they can be implemented in external memory by means of a sorting step which takes $\mathrm{sort}(X) = O((X/B)\log_m(X/B))$ I/Os to sort X items on the disk [37, 29]. In those cases, we have an additional I/O-term in Step 3:

$$\mathrm{sort}\left(\sum_{\sigma \in T_{i-1}} |T_\sigma|\right) = \mathrm{sort}(|T_i^I|).$$

Note that this cost involves the decompositions, not the conflict lists because they are not managed by the sorting process. This additive cost is acceptable in the applications where $f(S) = |S|$ and the corresponding desired I/O bound is $O(n \log_m n)$. Although the RIC approach via gradations extends to other problems like higher dimensional convex hulls and hyperplane arrangements, the achievable bound is a factor $\log_m n$ away from optimality because a sorting step seems unavoidable.

In the following subsections, we describe three further problems that can also be solved optimally by means of our RIC approach: The problems are intersection of 3-d halfspaces, batched planar point location, batch filtering and abstract Voronoi diagrams. For these problems we have $f(S) =$

$|S|$. The parameters of the gradation for algorithms **I** and **H** are $\mu = m$ and $\mu' = M$, and for Algorithm **I**$''$ they are $\mu = m^{1/2}/2C$ and $\mu' = M/2C \log n$. The conditions on N, M, B in the corresponding version of Theorem 5.2 are:

$$\max\{6C^2 \log C, B^{1/2}\} \le n,\ B \le M/\log m \text{ and } \sqrt{\log m}/C^2 \le \log n \le (MB)^{1/2}. \tag{4}$$

6.1 Intersection of Halfspaces in 3-d Space

We consider the configuration space in which the objects are halfspaces in 3-d space bounded by nonvertical planes from below. The cells are semi-infinite vertical triangular prisms bounded by a nonvertical triangle from below (that is, the set of all points on or above a nonvertical triangle in space). Given a set S of N halfspaces, we are interested in computing their intersections $\mathcal{I}(S)$. Combinatorially, the boundary of $\mathcal{I}(S)$ consists of vertices, edges, and faces (polygons). We assume non degeneracy, so that a vertex is determined by three halfspaces (an edge is determined by two halfspaces). $\mathcal{I}(S)$ can be decomposed into cells of constant complexity by first triangulating each face with a *fan* of edges from the lowest vertex to the other vertices, then the upward vertical extension of this triangulation produces a decomposition of $\mathcal{I}(S)$ into prisms. These prisms are the cells corresponding to S. For a prism σ in the decomposition of $\mathcal{I}(R)$, $R \subseteq S$, $\delta(\sigma, S)$ consists of those halfspaces whose bounding planes define the vertices of σ (hence a bounded number) and S_σ consists of those halfspaces whose bounding planes intersect σ. The number of faces, edges and vertices in the set S of 3-d halfspaces is $O(N)$, so Eqn. (1) becomes $\mathbf{E}\left[\sum_{\sigma \in \mathcal{T}(R)} N_\sigma\right] \le CN$. Figure 5 shows an example projected onto the xy-plane with two of its faces triangulated; the thick triangle is the projection of a prism. The triangulation of the face only partially inside the prism illustrates that cells (prisms) of T_i are not obtained just by stitching pieces in T_i^I. However, we note that a cell of T_i is determined by its three vertices and that its conflict list is equal to the

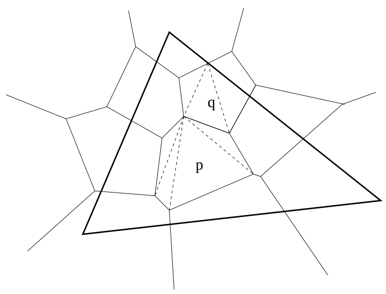

Figure 5: Projection of halfspace intersection

union of the conflict lists of its vertices (the conflict list of a vertex v with respect to S consists of those hyperplanes $s \in S$ which lie above v). The algorithm for the current problem follows the algorithm outlined in Section 3 (Algorithms **I** and **H**) with appropriate changes. We elaborate on those for Steps 2 and 3.

Walking. For a halfspace s, first a vertex in the intersection of its bounding plane h and the boundary of $\mathcal{I}(R)$ is located using a logarithmic number of operations, and then a walk proceeds along the intersection of h and the triangulation of the boundary of $\mathcal{I}(R)$, making use of the adjacency information between triangles, and thus using a number of operations linear in the number of conflicts found.

Clean-up. The vertices incident to a particular face are identified by creating three copies of each vertex, each with a key equal to each of the three incident planes, and then sorting them lexicographically. Then, for each face, the lowest vertex and the fan triangulation are determined. As already pointed out, the conflict list of a prism is determined as the union of the conflict lists of its vertices (so, actually, one can just maintain the conflict lists of the vertices).

Theorem 6.1 *The 3-d halfspace intersection problem is solved optimally by Algorithm* **H** *in* $O(n \log_m n)$ *expected I/Os and* $O(N \log N)$ *expected internal operations. Under the conditions in (4), the optimality is also achieved by the simpler Algorithm* $\mathbf{I}''$.

By standard geometric transformations [18], this also solves the 3-d convex hull and 2-d Euclidean Voronoi diagram problems. Clearly, a somewhat simpler algorithm also solves the 2-d convex hull problem.

6.2 Batched Point Location in a Planar Subdivision

In the batched planar point location one is given a set S of N interior disjoint edges (sharing vertices is allowed) that define a decomposition of the plane into *regions* (connected components), and a query set P of K points. For each point $x \in P$, one must determine the region which contains x. The goal is an algorithm that executes $O((n+k)\log_m n)$ expected I/Os. This result has already been obtained by Goodrich et al. [21] (for a monotone subdivision) and by Arge et al. [5] (for the general case and deterministically). But our aim here is a simple solution within our general approach. We specifically solve the equivalent problem of determining for each $x \in P$, the segment in S directly above x.

Mulmuley [27] has used gradations for the purpose of point location as follows. The incremental construction of $\mathcal{T}(S)$ determines a directed acyclic graph $G(S)$ as follows: the vertex set $G(S)$ consists of the trapezoids $\sigma \in T_i$ for $i = 0, \ldots, l$, and (σ, τ) is an edge if for some i, $\sigma \in T_{i-1}$, $\tau \in T_i$ and $\sigma \cap \tau \neq \emptyset$ (that is, $\tau \cap \sigma \in T_\sigma$). The point location for a point x is a search in $G(S)$ which starts at the root (the only trapezoid at level 0) and progresses from level to level keeping track of the trapezoid $\sigma \in T_i$ that contains x.

Our solution is an I/O-efficient implementation of this approach. More specifically, first, the trapezoidal decomposition $\mathcal{T}(S)$ induced by S is computed using our algorithm for the segment intersections problem (actually, since the set of edges S is interior disjoint, the algorithm to construct the trapezoidal decomposition can be considerably simplified: since $f(S) = |S|$, the clean-up can be performed using sorting and there is no need for a special small size case algorithm). Then the multiple search in $G(S)$ is made I/O-efficient by using the batch filtering technique of Goodrich et al [21] adapted here to work with our construction by gradations. The details are given in the next subsection 6.2. We can therefore state the following result:

Theorem 6.2 *Algorithm* **H** *solves the batched planar point location problem in* $O((n+k)\log_m n)$ *expected I/Os, and* $O(N \log N + K)$ *expected internal operations. Under the conditions in (4), the optimality is also achieved by the simpler Algorithm* $\mathbf{I}''$.

Batch Filtering

In the previous subsection, we observed that the point location for a point x is a search in the acyclic graph $G(S)$ which starts at the root (the only trapezoid at level 0) and progresses from level to level keeping track of the trapezoid $\sigma \in T_i$ that contains x. The multiple search in $G(S)$ for all the query points in the set P was made I/O efficient there by the appropriate use of the batch filtering technique of Goodrich et al. [21]. In what follows we show how our RIC approach is flexible enough to be adapted to solve this multiple search process without any loss in efficiency.

This way, the batched planar point location problem can be solved entirely in the RIC framework with expected optimal I/Os.
Let us first make the assumptions of Algorithm **I**, that is, ignore the deviations of $|R_{i,\sigma}|$. In the batched filtering technique, the K searches start at the root of the directed acyclic graph $G(S)$ and progress level by level (see the previous section). The search in level $i-1$ is completed for all K points before continuing with level i. At the beginning of the i-th round, for each $\sigma \in T_{i-1}$, there is the list P_σ of all points in P contained in σ. Then, the search proceeds in two steps analog to steps 2 and 3 of Algorithm **I**:

1. Each $\sigma \in T_{i-1}$ is considered in turn. For each $\tau \in T_\sigma$, an internal memory buffer of size B is reserved. Then, for each $x \in P_\sigma$ perform a point location in T_σ (in internal memory), and write the result into the corresponding buffer. As a buffer becomes full, it is written to external memory. This results in P_τ for each $\tau \in T_\sigma$.

2. For each $\tau \in T_i$, we bring together the sets $P_{\tau\cap\sigma}$ for some $\sigma \in T_{i-1}$. This only requires a movement of the data in blocks, as the information on the pieces of τ is already available from the construction of $\mathcal{T}(S)$.

The number of I/Os performed can be evaluated in two parts. The first part accounts one I/O per trapezoid of T_{i-1}, T_i^I and T_i. This results in a cost of $O(n)$ I/Os over all rounds. The second part accounts for the direct manipulation of the K points and it is $O(k)$ I/Os per round, for a total of $O(k \log_m n)$ I/Os. Thus, given a preprocessing of $O(n \log_m n)$ expected I/Os, the queries can be answered using $O(n + k \log_m n)$ I/Os. The overall cost is then $O((n+k)\log_m n)$ expected I/Os.

Now, let us consider the modifications necessary to handle the deviations, that is, $\mathcal{T}(S)$ is constructed using Algorithm **H**. For each $\sigma \in T_{i-1}$ and $x \in P_\sigma$, we need to perform a point location for x in $\hat{T}_\sigma$. This is done as follows: Consider each $\tau \in \hat{T}_\sigma$ in turn. Then scan P_σ checking whether $x \in \tau$; if that is the case, it is written into an internal memory buffer which is written to external memory when full. At this point, for each $\tau \in \hat{T}_\sigma$, the point location for P_τ in T_τ can be performed in internal memory. Afterwards, there are two clean-up steps, each of which consists of a simple data movement. In conclusion, there is an additional cost

$$\sum_{\sigma \in T_{i-1}} \frac{|P_\sigma|}{B} t_\sigma^2,$$

where t_σ^2 is an upper bound on the size of $\hat{T}_\sigma$. By an appropriate use of Eqn. (5), it is found that in the expectation of this sum, t_σ also behaves as a constant. Hence, the expectation of this cost is just $O(k)$. Thus, this results in an overall cost of $O((n+k)\log_m n)$ expected I/Os to answer the queries. However, we must point out that the analysis requires P to be fixed, so that to obtain the claim in general, we need to perform the construction of $\mathcal{T}(S)$ every time that a new set P of queries is given.

Theorem 6.3 *Algorithm* **H** *solves the batch filtering problem in $O((n+k)\log_m n)$ expected I/Os, and $O(N \log N + K)$ expected internal operations. Under the conditions in (4), the optimality is also achieved by the simpler Algorithm* $\mathbf{I}''$.

6.3 Abstract Voronoi Diagrams

Abstract Voronoi diagrams were introduced in [22]. They are defined by a system of bisecting curves in the plane. We can deal with a subset of them (see [3]), in which a Voronoi cell can be decomposed into cells of bounded size and hence a configuration space of bounded degree results. This subset still includes a large number of concrete important examples, e.g. Voronoi diagrams of line segments.

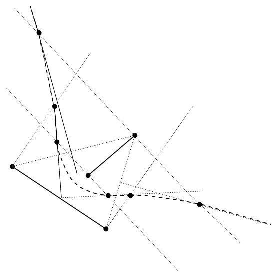

Figure 6: Two segments and its bisector

For each pair of objects, their *bisector* splits the plane into the points closer to each of the objects. These bisectors are assumed to be *piecewise algebraic curves* (a simple curve that consists of a constant number of pieces, each one algebraic of constant degree), and some topological and nondegeneracy assumptions are made about them. An important example, which we take up in the remaining, is the Voronoi diagram of line segments in the plane. The example in Fig. 6 shows the bisector of two segments, it consists of seven pieces (the thin lines are auxiliary), linear and quadratic. The Voronoi cell of an object O is the intersection over all the other objects O' of the region closer to O as determined by the bisector between O and O'. It is assumed that the Voronoi cells can be decomposed into smaller cells, called *trapezoids*, so that the resulting configuration space satisfies the bounded degree property. Figure 7 shows an example in which one of the Voronoi cells is decomposed into cells of bounded size.

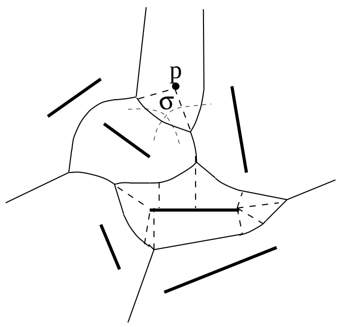

Figure 7: Voronoi diagram for 6 segments

The details of the algorithm are similar to those for the halfspace intersection problem. We only point out how to obtain the conflict lists during the clean-up step: Consider a trapezoid σ inside the Voronoi cell of a segment s; the conflict list of σ consists of those segments in S whose bisectors with s intersect an edge of σ which is also a Voronoi edge. This allows to perform the clean-up by using sorting to bring together the conflicts of each trapezoid. Figure 7 shows a (triangular) trapezoid σ and two bisectors that intersect its edges (note that only one of the intersected edges is a Voronoi edge).

Theorem 6.4 *Abstract Voronoi diagrams (as defined in [3]) are computed optimally by Algorithm* $\mathbf{H}$ *in* $O(n \log_m n)$ *expected I/Os, and* $O(N \log N)$ *expected internal operations. Under the conditions in (4), the optimality is also achieved by the simpler Algorithm* $\mathbf{I}''$.

7 Multiple Disks Model

We discuss the modifications required to extend our results to a system with D parallel disks. In this model, it is assumed that an I/O-operation can transfer D blocks, one from/to each disk. This implies that the main memory must have size $M = \Omega(DB)$. In the D-disk model a set of L items can be scanned with $\lceil L/(DB) \rceil$ I/Os provided the L/B disk blocks containing the items are evenly spread over the disks. Under the same assumption the set can be sorted with $O(L/(DB) \log_m(L/(DB)))$ I/Os, see [6, 29].

We modify our algorithms in such a way that the conflict list of every trapezoid spreads nearly evenly over the disks, in order to take advantage of the parallel block transfer. This requires the following changes to Steps 2 and 3 of Algorithm $\mathbf{I}$. We maintain a global buffer area of size DB in main memory, and in Step 2 we reserve a buffer of size B per trapezoid, as previously done. When the buffer of a trapezoid is full, we write the block to the global buffer area, and when the global buffer area is full, we choose a random permutation π of the integers $1 \ldots D$ and write the i-th block of the global buffer area to the disk $\pi(i)$ (emptying process). This guarantees that the conflict list of every trapezoid spreads nearly evenly over the disks (assuming the conflict list size is $\Omega(DB \log D)$ [6]).

We point out that the simpler approach consisting of storing the blocks in the global buffer area striped among the D disks would not work optimally. In fact, it might be the case that the conflict list of a given trapezoid is stored always on the same disk during a sequence of emptying processes. This way, in the next round of Algorithm $\mathbf{I}$, we could not retrieve this conflict list with an optimal number of parallel I/Os.

In Step 3, the conflict lists are handled in a similar way, and since the topological sort of T_{l-1} may require an I/O-operation for every internal operation, we set $\mu' = \max(DB \log D, M^{1/2})$ (the $\log D$ factor is so that conflict list sizes are $\Omega(DB \log D)$) and obtain an I/O-bound of $O(N/(DB) \log_m(N/(DB)) + K/(DB))$.

8 Concluding Remarks

It seems likely that our approach is also efficient for a model with D disks and p CPUs; such as the models proposed in [17, 37]. Steps 1 and 2 parallelize trivially by working on p trapezoids concurrently. Step 3 is more difficult to parallelize. If T_{l-1} is sufficiently small, topological sorting may be performed with a single processor. Otherwise, since T_{l-1} is a planar graph, it follows from the planar separator theorem that T_{l-1} can be divided in about p pieces of size $|T_{l-1}|/p$ each by removing no more than $\sqrt{|T_{l-1}| \cdot p}$ adjacencies between trapezoids. We assign each piece to a single processor and let the processor do the clean-up across adjacencies within the piece. We then do the clean-up across all removed adjacencies on a single processor. As long as $|T_{l-1}|/p \geq \sqrt{|T_{l-1}| \cdot p}$ or $p \leq |T_{l-1}|^{1/3}$ this should result in perfect speed-up.

Finally, we are working on the implementation of the simpler version of our algorithms (i.e., Algorithm $\mathbf{I}''$) in the framework of the `LEDA-SM` library [15], in order to evaluate their practical efficiency on real-world problems.

References

[1] A. Aggarwal and J. S. Vitter. The Input/Output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988.

[2] N. M. Amato, M. T. Goodrich, and E. A. Ramos. Computing faces in segment and simplex arrangements. In *Proc. 27th Annu. ACM Sympos. Theory Comput.*, 672–682, 1995.

[3] N. M. Amato and E. A. Ramos. On computing Voronoi diagrams by divide-prune-and-conquer. In *Proc. 12th Annual ACM Sympos. Comput. Geom.*, 1996. 672–682, 1995.

[4] L. Arge, P. Ferragina, R. Grossi, and J. S. Vitter. On sorting strings in external memory. *Proc. ACM Symposium on the Theory of Computing (STOC '97)*, 540–548, 1997.

[5] L. Arge, D. E. Vengroff, and J. S. Vitter. External-memory algorithms for processing line segments in geographic information systems. In *Proc. Annual European Symposium on Algorithms, LNCS 979*, 295–310, 1995.

[6] R. D. Barve, E. F. Grove, and J. S. Vitter. Simple randomized mergesort on parallel disks. In *Proc. ACM Symp. on Parallel Algorithms and Architectures*, pages 109–118, 1996.

[7] H. Brönnimann, B. Chazelle, and J. Matoušek. Product range spaces, sensitive sampling, and derandomization. In *Proc. 34th Annu. IEEE Sympos. Found. Comput. Sci.*, 1993, 400–409.

[8] B. Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete Comput. Geom.* **10** (1993) 377–409.

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

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

[11] Y.-J. Chiang, M. T. Goodrich, E. F. Grove, R. Tamassia, D. E. Vengroff and J. S. Vitter. External-memory graph algorithms. In *Proc. ACM-SIAM Symp. on Discrete Algorithms*, 139–149, 1995.

[12] K.L. Clarkson. Randomized geometric algorithms. In D.-Z. Du and F.K.Hwang, eds., *Computing in Euclidean Geometry*, Vol.1 of *Lecture Notes Series on Computing*, pages 117–162. World Scientific, Singapore, 1992.

[13] K. L. Clarkson, R. Cole, and R. Tarjan. Randomized parallel algorithms for trapezoidal diagrams. *International Journal of Computational Geometry & Applications*, pp. 117–1 33, 1992.

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

[15] A. Crauser and K. Mehlhorn. LEDA-SM, a platform for secondary memory computation, 1998, see WEB-pages of the authors.

[16] R. F. Cromp. An intelligent information fusion system for handling the archiving and querying of terabyte-sized spatial databases. In *S. R. Tate ed., Report on the Workshop on Data and Image Compression Needs and Uses in Scientific Community, CESDIS TR-93-99*, 75–84, 1993.

[17] F. Dehne, W. Dittrich, and D. Hutchinson. Efficient external memory algorithms by simulating coarse-grained parallel algori thms. *ACM Symposium on Parallel Algorithms and Architectures*, pp. 106–115, 1997.

[18] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer Verlag, Heidelberg, 1987.

[19] P. Ferragina and R. Grossi. A fully-dynamic data structure for external substring search. In *Proc. 27th Annu. ACM Sympos. Theory Comput.*, 693–702, 1995.

[20] G. A. Gibson, J. S. Vitter, and J. Wilkes. Report of the working group on storage I/O issues in large-scale computing. *ACM Computing Surveys*, 28(4), December 1996.

[21] M. T. Goodrich, J.-J. Tsay, D. E. Vengroff, and J. S. Vitter. External-memory computational geometry. In *Proc. IEEE Symp. on Foundations of Comp. Sci.*, 714–723, 1993.

[22] R. Klein. *Concrete and Abstract Voronoi diagrams.* LCNS 400, Springer-Verlag, 1988.

[23] R. Laurini and A. D. Thompson. *Fundamentals of Spatial Information Systems.* A.P.I.C. Series, Academic Press, NY 1992.

[24] J. Matoušek. Cutting hyperplane arrangements. *Discrete Comput. Geom.* **6** (1991) 385–406.

[25] K. Mehlhorn and S. Näher. LEDA: A platform for combinatorial and geometric computing. *Communications of the ACM* **38** (1995) 96–102.

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

[27] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms.* Prentice Hall, Englewood Cliffs, NJ, 1993.

[28] K. Mulmuley and S. Sen. Dynamic point location in arrangements of hyperplanes. *Discrete Comput. Geom.*, **8** (1992) 335–360.

[29] M. H. Nodine and J. S. Vitter. Greed sort: An optimal sorting algorithm for multiple disks. *Journal of the ACM* (1995) 919–933.

[30] Yale N. Patt. The I/O subsystem—a candidate for improvement. *Guest Editor's Introduction in IEEE Computer*, 27(3):15–16, 1994.

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

[32] J.H. Reif and S. Sen. Optimal parallel randomized algorithms for three-dimensional convex hulls and related problems. *SIAM J. Comput.* **21** (1992) 466–485.

[33] H. Samet. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS.* Addison Wesley, MA 1989.

[34] R. Seidel. Backward analysis of randomized geometric algorithms. In *New Trends in Discrete and Computational Geometry* (J. Pach, ed.), *Algorithms and Combinatorics*, Vol. 10, Springer-Verlag, 1993, pp. 37–67.

[35] J. Sibeyn and M. Kaufmann. BSP-like external-memory computation. *Proc. of Italian Conference on Algorithms and Complexity*, 1997.

[36] D.E. Vengroff. *TPIE User Manual and Reference.* Duke University, Technical Report 1995.

[37] J. S. Vitter and E. A. M. Shriver. Algorithms for parallel memory, I: Two-level memories. *Algorithmica*, 12(2–3):110–147, 1994.

A Sampling in Configuration Spaces

A.1 Configuration Spaces

We consider geometric algorithms in the framework developed by Clarkson and Shor[14] and Mulmuley [26, 27]. A geometric problem is formulated in terms of a *configuration space* consisting of a set $\mathcal{O}$ of *objects*; a set $\mathcal{C}$ of *cells*; a mapping $\mathcal{T}$ indicating for a set $S \subseteq \mathcal{O}$, the set $\mathcal{T}(S) \subseteq \mathcal{C}$ of cells determined by S; a mapping δ indicating for $\sigma \in \mathcal{C}$ and $S \subseteq \mathcal{O}$, the set of objects $\delta(\sigma, S)$ in S that *define* σ; and a mapping indicating for $\sigma \in \mathcal{C}$ and $S \subseteq \mathcal{O}$, the set of objects S_σ in S that *conflict* with σ, called the *conflict list* of σ in S. A configuration space satisfies the property of *bounded degree* if there are constants D and D' such that if $S \subseteq \mathcal{O}$ and $\sigma \in \mathcal{T}(S)$ then $|\delta(\sigma, S)| \leq D$, and if $S \subseteq \mathcal{O}$ with $|S| \leq D$ then $|\mathcal{T}(S)| \leq D'$; D is called the *dimension* of the configuration space. A configuration space satisfies the property of *locality* if for all $R \subseteq S \subseteq \mathcal{O}$ and $\sigma \in \mathcal{C}$ the following holds: $\sigma \in \mathcal{T}(R)$ iff $\delta(\sigma, S) \subseteq R$ and $S_\sigma \cap R = \emptyset$. In the applications, the objects $\mathcal{O}$ "live" in a Euclidean space, and a subset $S \subseteq \mathcal{O}$ determines a collection of cells (whose combinatorial complexity is not necessarily bounded), the *arrangement* $\mathcal{A}(S)$ of S, which we are interested in computing. Then $\mathcal{T}(S)$ is a *canonical decomposition* of $\mathcal{A}(S)$ into disjoint cells of bounded combinatorial complexity and, given a set S of objects, the problem is to compute $\mathcal{T}(S)$. We assume that the objects S and the cells $\mathcal{T}(S)$ form a *configuration space* satisfying the *bounded degree* and *locality* properties.

A.2 Sampling

Recall that a p-sample R from S is obtained by taking each element of S into R independently with probability p. Let $f(S)$ denote an upper bound on the size of $\mathcal{T}(S)$, and let $f(p, S)$ be an upper bound on the expected size of $\mathcal{T}(R)$ where R is a p-sample from S. We assume that $f(p/2, S) = O(f(p, S))$, which is the case in all our applications but not true in general. Let $N_\sigma = |S_\sigma|$. The following generalization of Eqn. (1) is needed in the analysis of the algorithms [14, 26, 27].

Fact A.1 *There is a constant $c > 0$ such that if the function g satisfies $g(tx)e^{-ct} = O(g(x))$ for $t \geq 1$, then there is a constant C such that for a finite set S of objects, if R is a p-sample from S, then*

$$\mathbf{E}\left[\sum_{\sigma \in \mathcal{T}(R)} g(N_\sigma)\right] \leq C\, g(1/p)\, f(p, S). \tag{5}$$

The left hand side of Eqn. (5) is a functional average of the conflict list sizes, and thus this equation indicates a sense in which the *average* conflict list size is at most $1/p$. Using $g(x) = x$ in Eqn. (5), we obtain Eqn. (1).

A.3 Cuttings

Fact 1.1 is well known but we sketch the proof as the procedure is essential in our optimal algorithms.

Proof. Let D be so that x objects determine $O(x^D)$ cells (bounded degree property). Obtain a p-sample R from S, with $p = (Cr)^2/N$ and compute $\mathcal{T}(R)$ and its conflict lists. This takes $O(r^{2D}N)$ time using brute force (there are at most $O(r^{2D})$ cells to check). Repeat until no conflict list is greater than $|S|/r$. From Eqn. (2), with probability at least 1/2, $\max_{\sigma \in \mathcal{T}(R)} N_\sigma \leq |S|/r$. Thus, the expected number of trials is $O(1)$ and, hence, the expected number of operations required for the construction is $O(r^{2D}N)$. ■

Optimal size cuttings. Neither the bound on the average of Eqn. (1) nor the bound on the deviation of Eqn. (2) are sufficient in our applications. Fortunately, there is the following result due to Chazelle and Friedman [10, 24]. Again, we sketch the proof.

Fact A.2 *There is a randomized algorithm that, given a set S of N objects, constructs a $(1/r)$-cutting for S of expected size $O(f(p, S))$, requiring $O((1/p)f(p, S))$ expected operations, where $p = r/N$.*

Proof. Let R be a p-sample from S. For each $\sigma \in \mathcal{T}(R)$, construct a $(1/t_\sigma)$-cutting T_σ for S_σ restricted to σ, where $t_\sigma = pN_\sigma$ (t_σ is called the *excess* of σ), using the algorithm in Fact 1.1. Then the cells in $T = \bigcup_\sigma T_\sigma$ have conflict list size at most $1/p$ because for each $\tau \in T_\sigma$, $N_\tau \leq N_\sigma/t_\sigma = 1/p$, and by using Eqn. (5), one can verify the claims on the size and the expected number of operations. ■

B Analysis of RIC via gradations

In this section we analyze the number of operations executed by the RIC via gradations. For the purpose of analysis, we look at the computation either from a *backward* or from a *forward* point of view. In the former case, S_{i-1} is seen as a $(1/\mu)$-sample from S_i for $i < l$ and as a $(1/\mu')$-sample for $i = l$, and this implies that S_i is a q_i-sample from S where $q_i = 1/\mu'\mu^{l-1-i}$. In the latter case, S_i is obtained from S_{i-1} by adding a p_i-sample R_i taken from $S - S_{i-1}$, where p_i satisfies $q_i = q_{i-1} + (1 - q_{i-1})p_i$ (i.e. $p_i = (q_i - q_{i-1})/(1 - q_{i-1})$). Note that $p_i \approx q_i$ (more precisely, $p_i \leq (1/\alpha)q_i$ for $q_{i-1} \leq 1 - \alpha$). In the analysis, we use $\mathbf{E}_i$ to denote the expectation for sampling S_i from S, and $\mathbf{E}_{i-1,i}$ to denote the expectation for sampling S_i and S_{i-1} from S. The expectations $\mathbf{E}_i$ can be evaluated using Eqn. (5). To compute the expectations $\mathbf{E}_{i-1,i}$ one can take advantage of either the forward or backward views, respectively:

$$\mathbf{E}_{i-1,i}[X] \;=\; \mathbf{E}_{i-1}[\mathbf{E}_i[X|S_{i-1}]] \;=\; \mathbf{E}_i[\mathbf{E}_{i-1}[X|S_i]].$$

The computation of the double expectation $\mathbf{E}_{i-1,i}$ presents some difficulty if we keep using an arbitrary function $f()$. So, we concentrate on $f(p, S) = p|S| + p^2K(S)$, $K(S) = |\mathcal{K}(S)|$, where $\mathcal{K}(S)$ is additive over cells and $|\mathcal{K}(S)| = O(|S|^2)$ ($\mathcal{K}(S)$ is the set of pairwise intersection points in the segment intersections problem). In the following, recall that for simplicity, we omit constant factors.

Lemma B.1 *Let $f(p, S) = p|S| + p^2K(S)$. We have:*

(i) $\mathbf{E}_i[|T_i|] \leq f(q_i, S)$ *and* $\mathbf{E}_i[|T_i[S]|] \leq \dfrac{f(q_i, S)}{q_i}$

(ii) $\mathbf{E}_{i-1,i}\left[\displaystyle\sum_{\sigma\in T_{i-1}} |\mathcal{K}(R_{i,\sigma}) \cap \sigma|\right] \leq f(q_i, S)$

(iii) $\mathbf{E}_{i-1,i}\left[\displaystyle\sum_{\sigma\in T_{i-1}} N_\sigma \log|R_{i,\sigma}|\right] \leq \log\left(\dfrac{q_i}{q_{i-1}}\right)\dfrac{f(q_{i-1}, S)}{q_{i-1}}$

(iv) $\mathbf{E}_{i-1,i}[|T_i^I|] \leq f(q_i, S)$ *and* $\mathbf{E}_{i-1,i}[|T_i^I[S]|] \leq \dfrac{f(q_i, S)}{q_i}.$

Proof. (i) The first part follows by definition and the second follows from Eqn. 1.
(ii) is clear since $\bigcup_{\sigma\in T_{i-1}} \mathcal{K}(R_{i,\sigma}) \subseteq \mathcal{K}(S_i)$.
(iii) We adopt the forward view. First, $\mathbf{E}_i[\log(|R_{i,\sigma}|)|S_{i-1}] \le \log(p_i N_\sigma) \approx \log(q_i N_\sigma)$. Then, setting $g(x) = x\log(q_i x)$ in Eqn. (5), the desired expectation is at most $(1/q_{i-1})\log(q_i/q_{i-1})\ f(q_{i-1}, S)$.
(iv) For the first part, $\mathbf{E}_{i-1,i}[|T_i^I|] = \mathbf{E}_{i-1,i}[\sum_{\sigma\in T_{i-1}}\sum_{\tau\in T_\sigma} 1] \le \mathbf{E}_{i-1}[\sum_{\sigma\in T_{i-1}}(|R_{i,\sigma}| + |\mathcal{K}(R_{i,\sigma}) \cap \sigma|)] \le f(q_i, S)/q_i$. For the second part, we have

$$\begin{aligned}\mathbf{E}_{i-1,i}\left[\sum_{\sigma\in T_{i-1}}\sum_{\tau\in T_\sigma} N_\tau\right] &= \mathbf{E}_{i-1}\left[\sum_{\sigma\in T_{i-1}}(N_\sigma + p_i|\mathcal{K}(S)\cap\sigma|)\right]\\ &\le N + q_{i-1}K + p_i K \le \frac{f(q_i,S)}{q_i}.\end{aligned}$$

■

Bound on Internal Operations. Using (i-iii) in the previous lemma, in Eqn. (3) and adding over all rounds, we obtain the following bound for the expected number of operations performed (ignoring constant factors):

$$\sum_{i=1}^{l} f(q_i,S) + \sum_{i=1}^{l}\log\left(\frac{q_i}{q_{i-1}}\right)\frac{f(q_{i-1},S)}{q_{i-1}} + \sum_{i=1}^{l}\frac{f(q_i,S)}{q_i} \le f(S) + (\mu'\log\mu')f(\frac{1}{\mu'},S) + \log\mu\sum_{i=1}^{l-1}\frac{f(q_{i-1},S)}{q_{i-1}}.$$

Finally, substituting $f(p,S) = p|S| + p^2K(S)$, we obtain the bound $|S|\log|S| + K(S)$.

Bound on I/O Operations. In the case $K(S) \neq 0$, a sorting is not executed in the clean-up step and hence the total number of I/Os in the i-th round is bounded by $|T_i^I| + \frac{|T_i^I[S]|}{B}$. In the case $K(S) = 0$, a sorting operation is used in the clean-up step and then the total number of I/Os in the i-th round is bounded by $\mathrm{sort}(|T_i^I|) + \frac{|T_i^I[S]|}{B}$. In both cases, using the results above and adding up over all rounds, the bound $n\log_m n + k$ results.

B.1 Proof of Lemma 3.2.

First, we verify (i):

$$\begin{aligned}\mathbf{E}_{i-1,i}\left[\sum_{\sigma\in T_{i-1}} t_\sigma^c N_\sigma^e\right] &= \frac{1}{\mu^c}\mathbf{E}_{i-1,i}\left[\sum_{\sigma\in T_{i-1}}|R_{i,\sigma}|^c N_\sigma^e\right] \le \left(\frac{p_i}{\mu}\right)^c \mathbf{E}_{i-1}\left[\sum_{\sigma\in T_{i-1}} N_\sigma^{c+e}\right]\\ &\le \left(\frac{p_i}{\mu}\right)^c \frac{1}{q_{i-1}^{c+e}}(q_{i-1}N + q_{i-1}^2 K) \le \frac{1}{q_{i-1}^e}(q_{i-1}N + q_{i-1}^2K).\end{aligned}$$

The verification of (ii) is similar. To verify (iii) we need to show that for a p-sample R from S:

$$\mathbf{E}\left[\sum_{\sigma\in\mathcal{T}(R)} N_\sigma^c K_\sigma\right] \le \frac{K}{p^e}.$$

This is shown using a configuration space in which the set of cells consists of the pairs (σ, p) where σ is a trapezoid and p is an intersection point in $\mathcal{K}(S)$. Let $\mathcal{T}'(R)$ be the number of these cells in $\mathcal{T}(R)$ for a p-sample R from S and let $f'(p,S)$ be its expected number, then

$$\mathbf{E}\left[\sum_{(\sigma,p)\in\mathcal{T}'(R)} N_\sigma^c\right] \le \frac{f'(p,S)}{p}.$$

The left hand side is the expectation that we want to compute, and since $f'(p, S) = K$ (each point is in a trapezoid), the right hand side is also as desired.
Now, using this observation, we can verify (iii):

$$\begin{aligned}
\mathbf{E}_{i-1,i}\left[\sum_{\sigma\in T_{i-1}} t_\sigma^c \sum_{\tau\in T_\sigma} N_\tau\right] &\leq \left(\frac{p_i}{\mu}\right)^c \mathbf{E}_{i-1}\left[\sum_{\sigma\in T_{i-1}} N_\sigma^c (N_\sigma + q_i K_\sigma)\right] \\
&\leq \left(\frac{p_i}{\mu}\right)^c \left(\frac{1}{q_{i-1}^{c+1}}(q_{i-1}N + q_{i-1}^2 K) + q_i \frac{1}{q_{i-1}^c} K\right) \\
&\leq N + q_i K.
\end{aligned}$$

C Handling Degeneracies in the Segments Problem

Proceeding similarly as in [34], the algorithm and analysis can be modified so that multiple intersection points (and also other degeneracies) can be handled. This is specially important as otherwise the output sensitivity is not very meaningful.
The following degeneracies may appear: (i) vertices (endpoints or intersection points) with equal x-coordinate (this includes the possibility of a vertical segment), and (ii) vertices that are (interior) intersection points of more than 2 segments. To deal with the first one we use a symbolic perturbation so that points with the same x-coordinate are ordered left to right according to the increasing value of the y-coordinate. This creates trapezoids of null width but provides the resulting configuration space with the bounded degree property. As for the algorithm, we only need to point out that in the walk (in internal memory) to compute the conflict lists, to locate the next trapezoid requires to consider all the neighbors either adjacent through the upper and lower boundaries or through the vertices: as shown in the figure, if the exit point is a vertex of the trapezoid, then one needs to consider the latter ones. The analysis below shows that it is fine to check all these neighbors.

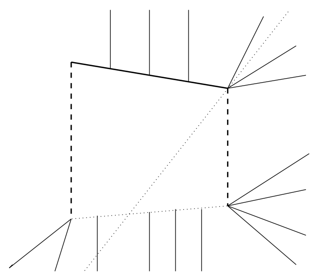

Figure 8: Trapezoid and its neighbors

Analysis. There are two points in the analysis that need to be verified to cover the case of degeneracies. First, we want to verify that $f(p, S) = O(pN + p^2 I(S))$, where $I(S)$ is the number of intersection points of $\mathcal{A}(S)$. This is not trivial, because a particular vertex v can be the intersection of more than 2 segments. Second, we verify that the exhaustive search among the neighbors during the walk is very efficient on the average.
First, we deal with the bound for $f(p, S)$. Let $\mathcal{I}(S)$ be the points of the arrangement $\mathcal{A}(S)$ that are interior intersection points for at least two segments (one such point could also be endpoint for other segments), and for $v \in \mathcal{I}(S)$ let d_v be the number of such segments for which its endpoints

are not adjacent to v. Thus, $f(p,S)$ is bounded by pN plus

$$\sum_{v\in\mathcal{I}(S)}\left(1-(1-p)^{d_v}-d_v p(1-p)^{d_v-1}\right),$$

since a vertex in $\mathcal{I}(S)$ appears in $\mathcal{I}(R)$ if at least two of the segments intersecting at v are taken into R. We want to show that this is $O(p^2 I)$ where $I=I(S)=|\mathcal{I}(S)|$. Let $d=\frac{1}{I}\sum_{v\in\mathcal{I}(S)} d_v$, the average of the d_v's. Then, we have

$$\begin{aligned}\sum_{v\in\mathcal{I}(S)}\left(1-(1-p)^{d_v}-d_v p(1-p)^{d_v-1}\right) &= \sum_{v\in\mathcal{I}}\left(1-(1-p)^{d_v-1}(1+(d_v-1)p)\right)\\ &\le I\left(1-(1-p)^{d-1}(1+(d-1)p)\right)\\ &\le I(1-(1-(d-1)p)(1+(d-1)p))\\ &= I(d-1)^2p^2.\end{aligned}$$

where in the second line we have used convexity of the function $g(x)=1-(1-p)^x(1+xp)$, and in the third line we have used $(1-p)^c\ge 1-cp$. Now, note that $\sum_v d_v$ is bounded by the number of edges in $\mathcal{A}(S)$ not incident to endpoints. Since the graph of the arrangement is planar, then we can bound the number of edges by three times the number of vertices plus 6. So $d\le(3I+6)/I\le 9$ (note that we defined d_v so that here the number of vertices is just I, not $I+2N$). Thus, we conclude that $f(p,S)=O(p|S|+p^2I(S))$.

Next, we deal with the cost of walking. More precisely, we argue that walking requires an expected number of operations bounded by the expected number of conflicts found. For a trapezoid σ in $\mathcal{T}(S)$, let η_σ be the number of neighbors of σ, including those that share just a vertex (as shown in the figure above). We want to verify that

$$\mathbf{E}\left[\sum_{\sigma\in\mathcal{T}(R)} N_\sigma\eta_\sigma\right]\le\frac{f(p,S)}{p}$$

where R is a p-sample from S. This follows by considering a configuration space in which the set of cells consists of pairs (σ,τ) where σ and τ are neighbors, and the segments conflicting with (σ,τ) are those conflicting with σ. This is a bounded degree configuration space, so for a p-sample R from S we have

$$\mathbf{E}\left[\sum_{(\sigma,\tau)\in\mathcal{T}'(R)} N_\sigma\right]\le\frac{f'(p,S)}{p},$$

where $\mathcal{T}'(R)$ is the set of adjacent pairs (σ,τ) in $\mathcal{T}(R)$ and $f'(p,S)$ is the expected size of $\mathcal{T}'(R)$. Because of the planarity of the trapezoidal diagram, we have $f'(p,S)=O(f(p,S))$, and so this is just the desired relation.