I/O-Efficient Batched Range Counting and Its Applications to Proximity Problems*

Tamás Lukovszki[1], Anil Maheshwari[2], and Norbert Zeh[2]

[1] Heinz-Nixdorf-Institut, Universität Paderborn, Germany
tamas@hni.uni-paderborn.de
[2] School of Computer Science, Carleton University, Ottawa, Canada
{maheshwa,nzeh}@scs.carleton.ca

Abstract. We present an algorithm to answer a set Q of range counting queries over a point set P in d dimensions. The algorithm takes $O\left(\text{sort}(|P|+|Q|) + \frac{(|P|+|Q|)\alpha(|P|)}{DB}\log_{M/B}^{d-1}\frac{|P|+|Q|}{B}\right)$[1] I/Os and uses linear space. For an important special case, the $\alpha(|P|)$ term in the I/O-complexity of the algorithm can be eliminated. We apply this algorithm to constructing t-spanners for point sets in $\mathbb{R}^d$ and polygonal obstacles in the plane, and finding the K closest pairs of a point set in $\mathbb{R}^d$.

1 Introduction

Motivation. Range searching and range counting problems have applications to spatial databases, geographic information systems, statistical analysis, and problems in computational geometry [1]. Conical Voronoi diagrams and θ-graphs appeared first in the early 80's as the base structure of the so called *region approach* for solving nearest neighbor problems and constructing a minimum spanning tree for a given set S of N points in d-dimensional Euclidean space [23]. These structures have numerous further applications and a rich history in various fields of computer science, e.g., in motion planning [11], construction of spanner graphs [16, 18, 19], problems on communication networks [15], for approximating the Euclidean minimum spanning tree [23] of a given point set, and for real-time walkthrough in virtual scenes [12]. In most of these domains, realistic data sets are too large to fit into the internal memory of state-of-the-art computers. Therefore special techniques are required to reduce the overhead involved in accessing secondary storage (i.e., disks). Existing internal memory techniques cannot be adapted, as random access into secondary memory is far too expensive.

Range counting. Range counting is a special kind of range searching problem. Given a point set P and a query range $q = [x_1, x_1'] \times [x_2, x_2'] \times \cdots \times [x_d, x_d']$,

* Research supported by NSERC, NCE GEOIDE, and DFG-SFB376.

[1] $\text{sort}(N)$ denotes the I/O-complexity of sorting N data items in external memory. See *Model of Computation.*

the standard range searching problem is to report all points in $P \cap q$. It is easy to answer a single query of this type in optimal $\Theta(\text{scan}(N))$ I/Os. Typically, however, we are either presented with a batch of queries (the *batched* scenario), or we want to build a data structure that can answer queries in $o(\text{scan}(N))$ I/Os per query, depending on the size of the output (the *online* scenario). I/O-efficient solutions for the online range searching problem have been presented in [3, 7, 20]. When solving the batched range searching problem, our goal is to minimize the total number of I/Os spent on answering all queries. An I/O-efficient solution to this problem has been presented in [2].

While range searching asks to report all points in $P \cap q$, range counting asks to report a value $\bigotimes_{p \in P \cap q} \lambda(p)$, where $\otimes$ is a commutative and associative operator and $\lambda(p)$ is a label assigned to point p. Important special cases include counting the elements in $P \cap q$ (the labels are 1, and $\otimes$ is standard addition) or finding the minimum point in $P \cap q$ w.r.t. some arbitrary weighting scheme.

θ-Graphs. Given a set $\{p_0, \ldots, p_k\}$ of points in $\mathbb{R}^d$ such that the vectors $(p_i - p_0)$, $0 \leq i \leq d$ are linearly independent, we define the *simplicial cone* spanned by points $p_0, \ldots, p_k$ as the set $\{p_0 + \sum_{i=1}^{d} \lambda_i (p_i - p_0) : \lambda_i \geq 0\}$. We call p_0 its *apex*. A *θ-frame* is a set C of simplicial cones such that each cone has its apex at the origin, $\bigcup_{c \in C} c = \mathbb{R}^d$, and each cone $c_i \in C$ contains a ray l_i emanating from the origin such that for any other ray l in the cone emanating from the origin, $\angle l_i l \leq \theta/2$. Denote l_i as the *cone axis* of c_i. In [19], it is shown how to construct a θ-frame of size $(d/\theta)^{O(d)}$.

Given a θ-frame C and a point p, let $c_0(p), \ldots, c_{k-1}(p)$ and $l_0(p), \ldots, l_{k-1}(p)$ be translated copies of cones $c_0, \ldots, c_{k-1} \in C$ and rays $l_0, \ldots, l_{k-1}$ such that the apexes of cones $c_i(p)$ and the endpoints of rays $l_i(p)$ coincide with point p. For $p, q \in P$ and $0 \leq i < k$, the distance $dist_{c_i}(p, q)$ between p and q w.r.t. cone c_i is defined as the Euclidean distance between p and the projection of q onto the translated ray $l_i(p)$ if q is contained in $c_i(p)$, and infinity otherwise. The Euclidean distance between two points p and q is denoted by $dist_2(p, q)$. For each point $p \in P$ and $0 \leq i < k$, let $P_{c_i(p)} = P \cap c_i(p)$.

The *K-th order θ-graph* $G_{\theta,K}(P)$ is defined as follows: The points of P are the vertices of $G_{\theta,K}(P)$. For every point $p \in P$ and every cone $c_i \in C$, we add directed edges from p to the $K^* = \min(K, |P_{c_i(p)}|)$ vertices in $P_{c_i(p)}$ that are closest to p w.r.t. the distance function $dist_{c_i}$.

A *t-spanner*, $t > 1$, for a point set P is a straight line graph with vertex set P such that for each pair $p, q \in P$ of vertices, the shortest path from p to q in G is at most t times longer than the Euclidean distance from p to q; the *length* of a path is the sum of the Euclidean lengths of its edges. We call such a shortest path a *t-spanner path* from p to q; t is called the *stretch factor* of G. In [19], it is shown that in a fixed dimension $d \geq 2$ and for $0 < \theta < \frac{\pi}{3}$, the θ-graph $G_\theta(P)$ is a $\left(\frac{1}{1 - 2\sin(\theta/2)}\right)$*-spanner* for P, and that it can be constructed in $O(N \log^{d-1} N)$ time using $O(N \log^{d-2} N)$ space.

A spanner graph G is *K-fault tolerant* if after removing at most K vertices or edges from G, the remaining graph still contains a path between each pair

p, q of vertices which is at most t times longer than the shortest path between p and q in the complete Euclidean graph after removing the same set of vertices or edges [17]. In [18], it is shown that the K-th order θ-graph is K-fault tolerant.

The θ-graph $G_\theta(O)$ for a set O of simple polygonal obstacles in the plane is defined as follows: The vertex set of $G_\theta(O)$ is the set of obstacle vertices in O. Each vertex v is connected to the nearest *visible* vertex in each cone $c(v)$ w.r.t. $dist_c$. This graph has been introduced in [11] to solve the approximate shortest path problem in two and three dimensions: Given two points s and t and a constant $\epsilon > 0$, find an obstacle avoiding path which is at most $(1+\epsilon)$ times longer than the shortest obstacle avoiding path between s and t.

Conical Voronoi diagrams. The *conical Voronoi diagram* $CVD_c(P)$ of a set P of points is closely related to the θ-graph. For a cone c, the *conical Voronoi region* V_p of a point $p \in P$ is the set of points in the plane having p as the closest point in P w.r.t. $dist_c$. $CVD_c(P)$ is the planar subdivision defined by the Voronoi regions V_p, $p \in P$. Similarly, the conical Voronoi diagram $CVD_c(O)$ of a set O of simple polygonal obstacles is the planar subdivision defined by the obstacles and the Voronoi regions V_p of the obstacle vertices, where V_p contains the points $x \in \mathbb{R}^2$ having obstacle vertex p as the closest *visible* vertex w.r.t. $dist_c$.

Model of computation and related results. Our algorithms are designed and analyzed in the Parallel Disk Model (PDM) [22]: An external memory consisting of D disks is attached to a machine with an internal memory of size M. Each of the D disks is divided into blocks of B consecutive data items. Up to D blocks, at most one per disk, can be transferred between internal and external memory in a single I/O-operation. The complexity of an algorithm is the number of I/O operations it performs. Many external memory (EM) algorithms and techniques for fundamental problems in computational geometry, graph theory, GIS, matrix computations, etc. have been developed in this model. Due to the lack of space we refer the reader to the survey of [21] and mention only the most relevant work here. It has been shown that sorting an array of size N takes $\mathrm{sort}(N) = \Theta\left(\frac{N}{DB}\log_{M/B}\frac{N}{B}\right)$ I/Os [21, 22]; scanning an array of size N takes $\mathrm{scan}(N) = \Theta\left(\frac{N}{DB}\right)$ I/Os. EM algorithms for computing pairwise intersections of orthogonal line segments, answering range queries in the plane, finding all nearest neighbors for a set of N points in the plane, dominance problems, and other geometric problems in the plane are discussed in [2, 3, 7, 13, 20]. General line segment intersection problems have been studied in [6]. For lower bounds on computational geometry problems in EM see [5]. See [4] for buffer trees, priority queues, and their applications.

Overview. In Sect. 2, we discuss our solution to the batched range counting problem. In Sect. 3, we use the solution for a special case of this problem to compute K-th order θ-graphs for point sets in d dimensions. We also discuss how to report a spanner path I/O-efficiently. In Sect. 4, we apply $O(\sqrt{K})$-th order θ-graphs to solve the K-closest pairs problem in d dimensions. Finally, in

Sect. 5, we show how to compute the θ-graph and the conical Voronoi diagram of a given set of disjoint simple polygonal obstacles in the plane I/O-efficiently.

In order to solve the batched range counting problem, one could use the range searching algorithm of [2] to report the points in each query range, and then count the points reported for each query. Using this strategy, the complexity of answering a query depends on the number of points in the query range, which can be as large as N. Our solution is independent of the number of points in each query range.

In [14], an asymptotically more efficient construction for spanners of point sets in d dimensions is presented. However, as this construction is based on a well-separated pair decomposition [10] of the point set, the constants hidden in the big-Oh notation are extremely large, the construction works only for point sets, and the approach cannot be used to construct K-fault tolerant spanners. For moderate dimensions, we expect our algorithm to compare favorably with the one of [14]. Similarly, the solution to the K-closest pair problem presented in [14] is asymptotically more efficent than ours; but for moderate dimensions we expect our algorithm to be more efficient.

Our construction of θ-graphs for sets of polygons is based on the construction of [11]. However, the algorithm of [11] is not I/O-efficient and cannot easily be made I/O-efficient. We prove a number of interesting properties of conical Voronoi diagrams to obtain an I/O-efficient solution to this problem.

2 Batched Higher-Dimensional Range Counting

In this section, we present I/O-efficient algorithms for the batched d-dimensional range counting problem. First we consider the important special case where w.l.o.g. $x'_1 = \infty$, for all query ranges $[x_1, x'_1] \times \cdots \times [x_d, x'_d]$. Our solution for this case is used in Sections 3 and 5. For the general case, we present a more complicated algorithm, which is by a factor of $O(\alpha(|P|))$ slower.

2.1 Colorable Problems

To solve the batched range counting problem, we apply the framework of colorable search problems defined in [2], although we extend it slightly. This framework can be used to derive an algorithm solving a search problem in $\mathbb{R}^d$, $d > 1$, from an algorithm solving the same search problem in $\mathbb{R}^1$.

Let $\mathcal{P}$ be a batched search problem answering a set Q of queries over a point set P in $\mathbb{R}^d$. Given a set C of colors, we define a coloring $\mathcal{C}$ assigning a color $c_q \in C$ to every query $q \in Q$ and a set $C_p \subseteq C$ of colors to every point $p \in P$. Every color $c \in C$ defines a point set $P_c = \{p \in P : c \in C_p\}$. Let $\mathcal{P}_\mathcal{C}$ be the problem of answering queries $q \in Q$ with respect to point sets P_{c_q}.

We call $\mathcal{P}$ $(\mathcal{I}_p, \mathcal{I}_r, S)$-*$m^c$-colorable* in dimension d, for some constant $0 < c \leq 1$, if for every coloring $\mathcal{C}$ with $|C| = O(\sqrt{m^c})$ and such that there are $O(m^c)$ different color sets assigned to the points in P, there exists an algorithm $\mathcal{A}$ that solves problem $\mathcal{P}_\mathcal{C}$ and can be divided into phases $\mathcal{A}_p^{(1)}, \mathcal{A}_r^{(1)}, \mathcal{A}_p^{(2)}, \mathcal{A}_r^{(2)}, \ldots,$

$\mathcal{A}_p^{(k-1)}, \mathcal{A}_r^{(k-1)}, \mathcal{A}_p^{(k)}$ so that phases $\mathcal{A}_p^{(1)}, \ldots, \mathcal{A}_p^{(k)}$ take $\mathcal{I}_p$ I/Os in total, the total I/O-complexity of phases $\mathcal{A}_r^{(1)}, \ldots, \mathcal{A}_r^{(k-1)}$ is $\mathcal{I}_r$, $\mathcal{A}$ uses S space, and phases $\mathcal{A}_p^{(1)}, \ldots, \mathcal{A}_p^{(k)}$ are independent of d.

The idea behind this definition is that we can use algorithm $\mathcal{A}$ to derive an algorithm $\mathcal{B}$ solving $\mathcal{P}$ in $\mathbb{R}^{d+1}$, only by replacing phases $\mathcal{A}_r^{(1)}, \ldots, \mathcal{A}_r^{(k-1)}$ by phases $\mathcal{B}_r^{(1)}, \ldots, \mathcal{B}_r^{(k-1)}$, in order to deal with the extra dimension. That is algorithm $\mathcal{B}$ consists of phases $\mathcal{A}_p^{(1)}, \mathcal{B}_r^{(1)}, \mathcal{A}_p^{(2)}, \mathcal{B}_r^{(2)}, \ldots, \mathcal{A}_p^{(k-1)}, \mathcal{B}_r^{(k-1)}, \mathcal{A}_p^{(k)}$.

We solve the search problem to be addressed by phase $\mathcal{B}_r^{(i)}$ using a buffer tree [4] of degree $\sqrt{m^c}$ and algorithm $\mathcal{A}_r^{(i)}$. The buffer tree is built over the coordinates of the points in P in the d-th dimension. Queries are filtered from the root of the tree to the leaves. At every level, each query q is answered w.r.t. the maximal multislab spanned by q. A point is colored with the colors of the multislabs that it is contained in. Hence, we have to solve a colored version of the d-dimensional problem at every level, which we do using $\mathcal{A}_r^{(i)}$. Adapting the proof of [2] to this more general framework, we obtain the following result.

Theorem 1. *If a batched search problem $\mathcal{P}$ is $(\mathcal{I}_p, \mathcal{I}_r, S)$-$m^c$-colorable in $\mathbb{R}^d$, then its $(d+1)$-dimensional version is $\left(\mathcal{I}_p, \mathcal{I}_r \cdot \log_{M/B} \frac{|P|}{B}, S\right)$-$\sqrt{m^c}$-colorable.*

Corollary 1. *If a batched search problem $\mathcal{P}$ is $(\mathcal{I}_p, \mathcal{I}_r, S)$-$m^c$-colorable in $\mathbb{R}^1$, then its d-dimensional version can be solved in $O\left(\mathcal{I}_p + \mathcal{I}_r \cdot \log_{M/B}^{d-1} \frac{|P|}{B}\right)$ I/Os using $O(S)$ blocks of external memory.*

We now apply this framework to solve the batched range counting problem.

2.2 Partially Unbounded Queries

The following algorithm shows that if w.l.o.g. $x_1' = +\infty$, for all queries $q \in Q$, the batched range counting problem is $\left(\text{sort}(|P|+|Q|), \text{scan}(|P|+|Q|), \frac{|P|+|Q|}{B}\right)$-$m$-colorable for $d = 1$: Sort all queries by left endpoints and all points by their x_1-coordinates (Phase $\mathcal{A}_p^{(1)}$). Scan P and Q in lock-step fashion, simulating a line sweep from $+\infty$ to $-\infty$. During the sweep, maintain a value $\Pi = \bigotimes\{\lambda(p) : p \in P \text{ and } x_1(p) \geq x_1(\ell)\}$, where $x_1(\ell)$ is the current position of the sweep line. When the sweep line passes a point p, update $\Pi_{\text{new}} \leftarrow \lambda(p) \otimes \Pi_{\text{old}}$. When the sweep line passes the left endpoint of a query q, report Π as the answer to query q (Phase $\mathcal{A}_r^{(1)}$). In order to make this solution m-colorable, maintain m separate products Π_{C_i}, one per color class C_i in the coloring. In order to report the answer to query q, compute $\Pi = \bigotimes\{\Pi_{C_i} : c_q \in C_i\}$ when the sweep passes the left endpoint of q. Using Cor. 1, we obtain the following result.

Theorem 2. *It takes $O\left(\text{sort}(|P|+|Q|) + \frac{|P|+|Q|}{DB} \log_{M/B}^{d-1} \frac{|P|}{B}\right)$ I/Os and linear space to answer a set Q of range counting queries over a point set P in $\mathbb{R}^d$, provided that w.l.o.g. $x_1' = +\infty$, for all queries $[x_1, x_1'] \times \cdots \times [x_d, x_d'] \in Q$.*

2.3 Bounded Queries

We turn to the general case now, where every query $q \in Q$ is a d-dimensional box. Again, we present a solution for the case $d = 1$ and generalize it to higher dimensions by applying Cor. 1. We define a sequence of functions $\phi_i(x)$ as follows: $\phi_0(x) = \lceil \frac{x}{2} \rceil$ and $\phi_i(x) = \min\{j \geq 0 : \phi_{i-1}^{(j)}(x) \leq B\}$, for $i > 0$, where $f^{(i)}(x)$ is defined as $f^{(0)}(x) = x$ and $f^{(i)}(x) = f(f^{(i-1)}(x))$, for $i > 0$. Thus, $\phi_1(x) = \log_2 x$, $\phi_2(x) = \log^* x$, and so on. It is an exercise to show that $\phi_{\alpha(N)}(N) = O(\alpha(N))$, where $\alpha(N)$ is the inverse of Ackermann's function. We develop a series of algorithms proving the following lemma.

Lemma 1. *For every $k > 0$, the one-dimensional batched range counting problem is $(\mathcal{I}_p, \mathcal{I}_r, S)$-$m^c$-colorable, where $\mathcal{I}_p \leq tk(\text{sort}(|Q|) + \text{scan}(|P|)) + t \cdot \text{sort}(|P|)$, $\mathcal{I}_r \leq tk(\text{scan}(|Q|) + \text{scan}(|P|)\phi_k(|P|))$, for some constant $t > 0$, and $S = O\left(\frac{|P|+|Q|}{B}\right)$.*

Proof sketch. Consider the case $k = 1$. The first phase $\mathcal{A}_p^{(1)}$ preprocesses P and Q as follows: Sort the points in P by their x_1-coordinates. Let T be a balanced binary tree over the points in P. (We do not construct T, but use it only as a conceptual tool.) With every node $v \in T$, associate a value x_v separating the points in the left subtree from the points in the right subtree. Associate every query $q \in Q$ with the highest node $v_q \in T$ such that q spans x_{v_q}. Split query q into two parts q_l and q_r to the left and right of x_{v_q}. This produces two sets Q_l and Q_r of left and right subqueries. Sort the queries in Q_l according to their corresponding nodes v_q, sorted top-down in T and left to right at every level; queries associated with the same node v are sorted by their left endpoints. The queries in Q_r are sorted analogously; but sort the queries associated with the same node by their right endpoints. This phase takes $O(\text{sort}(|Q|) + \text{scan}(|P|))$ I/Os and linear space.

Phase $\mathcal{A}_r^{(1)}$ answers left subqueries at each level h of T using a modification of the approach of Sect. 2.2. In particular, the running product Π is reset whenever the sweep passes a value x_v associated with a node v at level h. This takes $O(\text{scan}(|Q_h| + |P|))$ I/Os, where Q_h is the set of queries associated with nodes at level h. Right subqueries are answered using a similar procedure. As there are $\log_2 |P| = \phi_1(|P|)$ levels in T, and $\sum_{i=0}^{\log_2 |P|} |Q_h| = |Q|$, it takes $O(\text{scan}(|Q|) + \text{scan}(|P|)\phi_1(|P|))$ I/Os to answer all left and right subqueries. Once this is done, Phase $\mathcal{A}_p^{(2)}$ combines the answers to the left and right subqueries of each query in $O(\text{sort}(|Q|))$ I/Os. Choosing t appropriately, we obtain the claim for $k = 1$.

For $k > 1$, we define the tree T as follows: First associate the whole set P with the root r of T. Then split P into $|P|/\phi_{k-1}(|P|)$ subsets of size $\phi_{k-1}(|P|)$, and create one child of r for each subset. Apply this strategy recursively to the children of r. Consider a node v with children $w_0, \ldots, w_s$. Let $x_1, \ldots, x_s$ be values such that x_i separates the points associated with w_{i-1} from the points associated with w_i. Again, associate a query $q \in Q$ with the highest node $v_q \in T$ such that q spans some value x_i. Let x_l and x_r be the leftmost and rightmost

such values spanned by q, respectively. Split q into three subqueries q_l, q_m, and q_r to the left of x_l, between x_l and x_r, and to the right of x_r, respectively. Note that q_m does not exist if $x_l = x_r$. Now sort left and right subqueries by level and within each level as for the case $k = 1$. This modified Phase $\mathcal{A}_p^{(1)}$ still takes at most $(t/2)(\text{sort}(|Q|) + \text{scan}(|P|))$ I/Os.

Phase $\mathcal{A}_r^{(1)}$ now answers left and right subqueries as for $k = 1$. As the height of T is now $\phi_k(|P|)$, this takes at most $t(\text{scan}(|Q|) + \text{scan}(|P|)\phi_k(|P|))$ I/Os. Phase $\mathcal{A}_p^{(2)}$ combines the query results of q_l and q_r, and stores the result with q_m, in order to combine it with the answer to query q_m later. This takes at most $(t/2)\text{sort}(|Q|)$ I/Os. The remainder of the algorithm answers the middle subqueries of all queries in Q. Note that at every level of T, middle subqueries now stretch between values x_i. Call the interval bounded by two consecutive such values at the same level a *slab*. In order to answer middle subqueries at a particular level h, we compute a new point set P_h containing one point p_σ per slab σ. The label associated with p_σ is the product of the labels of all points in $P \cap \sigma$. We can now answer middle subqueries with respect to P_h without altering the solution. Due to the reduced size of P_h this takes $\mathcal{I}_p' \leq t(k-1)(\text{sort}(|Q_h|) + \text{scan}(|P_h|))$ and $\mathcal{I}_r' \leq t(k-1)(\text{scan}(|P| + |Q_h|))$ I/Os, by the induction hypothesis. Summing over all $\phi_k(|P|)$ levels, and adding the costs for answering left and right subqueries, we obtain the claim for $k > 1$. □

Choosing $k = 1$ for $d = 1$, and $k = \alpha(|P|)$ for $d > 1$, we obtain an $O(\text{sort}(|P| + |Q|))$ I/O algorithm for $d = 1$, and an $O\left(\frac{|P|\alpha^2(|P|)+|Q|\alpha(|P|)}{DB} \log_{M/B}^{d-1} \frac{|P|+|Q|}{B}\right)$ I/O algorithm for $d > 1$. Now partition P into contiguous subsets of size $\alpha(|P|)$ using $|P|/\alpha(|P|)$ splitters. Partition every query into left, middle, and right subqueries so that the left and right subqueries are maximized without spanning any splitter. For every query, answer the left and right subqueries by scanning the respective portion of P. This takes $O\left(\frac{|P|+|Q|}{DB}\alpha(|P|)\right)$ I/Os. Every middle subquery stretches between two splitters. Hence, we can represent every slab by a single point, thereby reducing the size of P to $|P|/\alpha(|P|)$, and we obtain the following theorem, applying Lem. 1 and Cor. 1.

Theorem 3. *It takes* $O\left(\text{sort}(|P| + |Q|) + \frac{(|P|+|Q|)\alpha(|P|)}{DB} \log_{M/B}^{d-1} \frac{|P|+|Q|}{B}\right)$ *I/Os and linear space to answer a set* Q *of range counting queries over a set* P *of points in* $\mathbb{R}^d$.

The *K-range minima problem* is to report the K points with minimum weight in each query range. Modifying the scan in Sect. 2.2 to maintain the K minima seen so far, it is easy to generalize the algorithm of this section to solve the K-range minima problem in $O\left(\text{sort}(|P| + |Q|) + \frac{(|P|+K|Q|)\alpha(|P|)}{DB} \log_{M/B}^{d-1} \frac{|P|+|Q|}{B}\right)$ I/Os, using $O\left(\frac{|P|+K|Q|}{B}\right)$ blocks of external memory.

3 Spanners for Point Sets

Computing K-th order θ-graphs. Our algorithm for constructing the K-th order θ-graph iterates over all cones $c \in C$ and computes the K closest points w.r.t. $dist_c$ in $c(p)$ for every point $p \in P$. Using an affine transformation, cone c can be transformed into the range $[0, +\infty) \times [0, +\infty) \times \cdots \times [0, +\infty)$, and reporting the K points closest to p in $c(p)$ translates into a K-range minima query for the modified cone $c'(p)$. Using Thm. 2, we obtain the following result.

Theorem 4. *Let P be a set of N points in $\mathbb{R}^d$, and $\theta < \pi$ be a constant angle. Then it takes $O\left(\text{sort}(N) + \frac{KN}{DB} \log^{d-1}_{M/B} \frac{N}{B}\right)$ I/Os and linear space to construct the K-th order θ-graph $G_{\theta,K}(P)$ of P.*

For fixed θ, the θ-graph has bounded out-degree, but not necessarily bounded in-degree. Using a two-phase approach, applying in both phases similar ideas to those presented in [8], we can compute in $O(\text{sort}(N))$ I/Os a spanner G of bounded in- and out-degree from a given θ-graph G'.[2]

Reporting paths in θ-graphs. A spanner path between two points s and t in a θ-graph can be computed as follows: For every point $p \in P$, determine the cone $c(p)$ containing t. Add the outgoing edge of p in cone $c(p)$ to a graph T. T is a tree with root t whose edges are directed towards the root. It follows directly from the arguments applied in the analysis of the spanning ratio of θ-graphs that the path from s to t in T is a spanner path. Hence, it is sufficient to report the path from s to t in T, which takes $O(\text{sort}(N))$ I/Os using time-forward processing [9].

4 *K*-Closest Pairs

The following theorem, which we prove in the full paper, can be used to find the K-closest pairs of a point set P in $O\big(\text{scan}\big(N\sqrt{K}\big)\big)$ I/Os, once an $O\big(\sqrt{K}\big)$-th order θ-graph has been constructed for P.

Theorem 5. *Let P be a set of points in $\mathbb{R}^d$, $0 < \theta < 2 \arccos \sqrt{4/5}$, $1 \le K \le n-1$, $K^* = \max(K, K^2/4)$, and $\{p, q\}$ be any of the K^* closest pairs in P. Then the K-th order θ-graph contains an edge between p and q.*

5 Spanners Among Polygonal Obstacles in the Plane

In this section, we prove the following theorem.

Theorem 6. *Given a set O of polygonal obstacles in the plane, with a total of N vertices, a linear size t-spanner for O can be computed in $O\left(\frac{N}{DB} \log_{M/(DB)} \frac{N}{DB}\right)$ I/Os using linear space.*

[2] The spanning ratio of G is $t = t' \cdot t'' \cdot \frac{1}{1 - 2\sin(B\pi/M)}$, where t' is the spanning ratio of G' and t'' is the spanning ratio of the single-sink spanners used in constructing G.

In [11], it is shown that the θ-graph $G_\theta(O)$ is such a t-spanner. Our algorithm to construct $G_\theta(O)$ follows the framework of [11]. However, we need to change the plane-sweep substantially, in order to perform it I/O-efficiently. For this purpose, we have to prove some new properties of conical Voronoi diagrams. The I/O-complexity of our algorithm becomes $O(\mathrm{sort}(N))$ if the endpoint dominance problem [6] can be solved in $O(\mathrm{sort}(N))$ I/Os.

The algorithm iterates over all cones $c \in C$ and computes the conical Voronoi diagram $CVD_c(O)$ (Fig. 1a) consisting of Voronoi regions V_x, $x \in P$. For each such region V_x and each point $p \in V_x$, x is the closest visible obstacle vertex in $c(p)$. Thus, for all obstacle vertices $y \in V_x$, the edge (y, x) is an edge of $G_\theta(O)$. Once $CVD_c(O)$ is computed, we add all such edges to the edge set of $G_\theta(O)$. In the full paper we show how to do this I/O-efficiently, so that Thm. 6 follows from the following result, which we prove in the rest of this section.

Theorem 7. *A representation of $CVD_c(O)$ as vertex and edge sets can be computed in $O\left(\frac{N}{DB}\log_{M/(DB)}\frac{N}{DB}\right)$ I/Os using $O(N/B)$ blocks of external memory.*

Assume that the coordinates have been transformed so that the cone axis of cone c points in positive y-direction. The construction of $CVD_c(O)$ uses a plane-sweep in this direction. Let c' be the reverse cone of c, i.e., the set of points p such that the apex of c' is contained in the cone $c(p)$. Then $V_x \subseteq c'(x)$. Denote the left and right boundaries of c' by h_l and h_r, respectively (Fig. 1b). Let H_l and H_r be two lines perpendicular to h_l and h_r, respectively, and directed upward. We denote the projections of a point p onto H_l and H_r by $H_l(p)$ and $H_r(p)$, respectively.

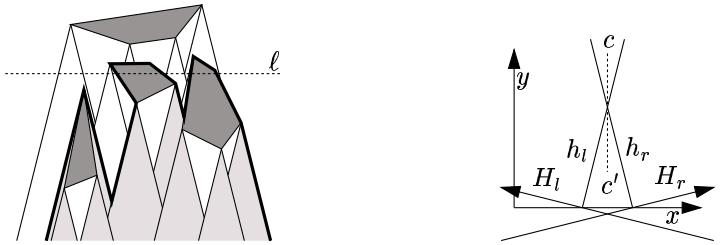
For the sake of simplicity, assume that the scene contains a dummy obstacle bounded from above by a horizontal line l_{bot} with y-coordinate $-\infty$. Now every cone $c'(x)$ becomes a triangle bounded by $h_l(x)$, $h_r(x)$, and a part of l_{bot}. As $V_x \subseteq c'(x)$, V_x is bounded from the left and right by $h_l(x)$ and $h_r(x)$ and from below by a polygonal chain consisting of parts of obstacle edges and/or parts of cone boundaries $h_l(y)$ and $h_r(y)$, for vertices $y \in P$ below x.

During the sweep, we maintain the invariant that the Voronoi regions V_y, for all vertices $y \in P$ below the sweep line ℓ, have been computed. Let P_ℓ be the set of vertices below ℓ and O_ℓ be the set of obstacles in O below or intersecting ℓ. Define the region $R_\ell = \bigcup_{y \in P_\ell} V_y \cup \bigcup_{o \in O_\ell} o$. As R_ℓ contains the dummy obstacle, it extends to infinity in both x-directions and in negative y-direction. The following lemma is proved in the full paper.

Lemma 2. *Region R_ℓ is connected.*

Lem. 2 implies that R_ℓ is bounded from above by a polygonal chain, even though it may contain holes. We call the upper boundary of R_ℓ the *horizon* of sweep line ℓ and denote it by $\mathcal{U}_\ell$ (Fig. 1a).

Let x be the current vertex on the sweep line ℓ, whose Voronoi region we want to compute. Let p_l be the first intersection point of $h_l(x)$ with $\mathcal{U}_\ell$ if $h_l(x)$ does not intersect the interior of an obstacle before intersecting $\mathcal{U}_\ell$. Otherwise, let $p_l = x$. We define a point p_r analogously with respect to $h_r(x)$. Then the edges $h_l^{\mathrm{vor}}(x) = (x, p_l)$, $h_r^{\mathrm{vor}}(x) = (x, p_r)$, and the part $\mathcal{U}_\ell(p_l, p_r)$ of $\mathcal{U}_\ell$ between



(a) (b)

Fig. 1. (a) A conical Voronoi diagram with the horizon $\mathcal{U}_\ell$ for sweep line ℓ shown in bold. Obstacles are shaded dark grey. Regions V_y for points y below the sweep line are shaded light grey. White regions below the horizon are holes in R_ℓ. (b) Illustrations of various definitions.

p_l and p_r bound a polygonal region R_x. Edge $h_l^{\text{vor}}(x)$ ($h_r^{\text{vor}}(x)$) is not defined if $p_l = x$ ($p_r = x$), and $R_x = \emptyset$ if $p_l = p_r = x$. The following lemma is shown in the full paper.

Lemma 3. *The region R_x defined as above is the Voronoi region of x. That is, $R_x = V_x$.*

Consider the region V_x. Lem. 3 implies that when the sweep line ℓ reaches vertex x, the whole boundary of V_x, except $h_l^{\text{vor}}(x)$ and $h_r^{\text{vor}}(x)$, has already been computed. To make the description of V_x complete, we need to compute $h_l^{\text{vor}}(x)$ and $h_r^{\text{vor}}(x)$. This can be done by performing two ray-shooting queries onto $\mathcal{U}_\ell$ in the directions of $h_l(x)$ and $h_r(x)$.

We answer ray-shooting queries in two stages. The first stage determines the points on obstacle boundaries hit by $h_l(x)$ and $h_r(x)$. The second stage determines whether $h_l(x)$ and $h_r(x)$ hit Voronoi edges $h_r^{\text{vor}}(y)$ and $h_l^{\text{vor}}(y')$ before hitting the respective obstacles.

The first stage is equivalent to solving the endpoint dominance problem for the set of obstacle vertices, which takes $O\left(\frac{N}{DB}\log_{M/(DB)}\frac{N}{DB}\right)$ I/Os [6]. Consider rays $h_l(x)$ and $h_r(x)$, for a vertex x on the boundary of an obstacle $o \in O$. Let $h_l^{\text{ext}}(x) = (x, y)$, where y is the first intersection of $h_l(x)$ with an obstacle. If $h_l(x) \setminus \{x\}$ intersects the interior of o before intersecting the boundary of o, $h_l^{\text{ext}}(x)$ is not defined. We define $h_r^{\text{ext}}(x)$ analogously. We say that segment $e = (x, y)$ *hits* segment $e' \subset \mathcal{U}_\ell$ if e and e' intersect in a point p and segment (x, p) does not intersect any other segment $e'' \subset \mathcal{U}_\ell$.

Observation 1. *The ray $h_l(x)$ ($h_r(x)$) hits a segment $h_r^{\text{vor}}(y)$ ($h_l^{\text{vor}}(y)$) if and only if segment $h_l^{\text{ext}}(x)$ ($h_r^{\text{ext}}(x)$) hits segment $h_r^{\text{vor}}(y)$ ($h_l^{\text{vor}}(y)$).*

If $h_l(x)$ does not hit an obstacle edge in $\mathcal{U}_\ell$, it hits a segment $h_r^{\text{vor}}(y)$ because segments $h_l^{\text{vor}}(y)$ and $h_l(y)$ are parallel. Analogously, $h_r(x)$ hits a segment $h_l^{\text{vor}}(y)$ if it does not hit an obstacle edge in $\mathcal{U}_\ell$. We show how to find the segment $h_r^{\text{vor}}(y)$ hit by $h_l^{\text{ext}}(x)$, and the segment $h_l^{\text{vor}}(y)$ hit by $h_r^{\text{ext}}(x)$, if any.

We do this in two separate plane sweeps. The first sweep finds segments $h_l^{\mathrm{vor}}(y)$ hit by segments $h_r^{\mathrm{ext}}(x)$, for all vertices $x \in P$. The second sweep finds all segments $h_r^{\mathrm{vor}}(y)$ hit by segments $h_l^{\mathrm{ext}}(x)$. As these two sweeps are similar, we only describe the first one. The difficulty is that segments $h_l^{\mathrm{vor}}(y)$ are not known before the first sweep. They are only computed by the second sweep. However, the goal of the first sweep is not to compute segments $h_l^{\mathrm{vor}}(y)$, but to determine whether a given segment $h_r^{\mathrm{ext}}(x)$ hits such a segment $h_l^{\mathrm{vor}}(y)$. The following lemma provides the tool to make this decision without knowing segments $h_l^{\mathrm{vor}}(y)$ explicitly. For a given vertex x on sweep line ℓ, let the *hiding set* of $h_l^{\mathrm{ext}}(x)$ be the set of endpoints z' of segments $h_l^{\mathrm{ext}}(z')$ with $H_l(z') > H_l(x)$[3] and $y(z') < y(x)$. The following lemma is proved in the full paper.

Lemma 4. *Given a vertex x on sweep line ℓ, let $h_l^{\mathrm{ext}}(x) = (x, y)$. Let z be the segment endpoint in the hiding set of $h_l^{\mathrm{ext}}(x)$ such that $H_r(z)$ is maximized. If the hiding set of $h_l^{\mathrm{ext}}(x)$ is empty, let $H_r(z) = -\infty$. Then there can be no vertex u above ℓ with $H_r(u) < \max\{H_r(y), H_r(z)\}$ such that $h_r^{\mathrm{ext}}(u)$ hits $h_l^{\mathrm{vor}}(x)$.*

Based on Lem. 4 we shorten segments $h_l^{\mathrm{ext}}(x)$ to segments $h_l^{\mathrm{vis}}(x)$ defined as follows: If point z in Lem. 4 exists and $H_r(z) > H_r(y)$, then $h_l^{\mathrm{vis}}(x) = (x, y')$, where y' is the point on $h_l^{\mathrm{ext}}(x)$ with $H_r(y') = H_r(z)$. Otherwise, $h_l^{\mathrm{vis}}(x) = h_l^{\mathrm{ext}}(x)$. Observe that $h_l^{\mathrm{vis}}(x) \subseteq h_l^{\mathrm{vor}}(x)$.

Corollary 2. *For a vertex $x \in P$, segment $h_r^{\mathrm{ext}}(x)$ hits segment $h_l^{\mathrm{vor}}(y)$ if and only if it hits $h_l^{\mathrm{vis}}(y)$.*

Next we describe how to compute segments $h_l^{\mathrm{vis}}(y)$ from the given set of segments $h_l^{\mathrm{ext}}(y)$. Given segments $h_l^{\mathrm{vis}}(y)$, we show how to compute the segment $h_l^{\mathrm{vis}}(x)$ (and thus segment $h_l^{\mathrm{vor}}(x)$) hit by $h_r^{\mathrm{ext}}(u)$ (if any), for all $u \in P$. Once we know the intersection point of $h_r^{\mathrm{ext}}(u)$ with this segment $h_l^{\mathrm{vis}}(x)$, we can shorten $h_r^{\mathrm{ext}}(u)$ to $h_r^{\mathrm{vor}}(u)$.

Lem. 4 says that in order to compute $h_l^{\mathrm{vis}}(x)$, we need to find the segment endpoint z below ℓ with $H_l(z) > H_l(x)$ and such that $H_r(z)$ is maximized over all such points. This is a partially unbounded range-maxima problem, which can be solved in $O(\mathrm{sort}(N))$ I/Os, by Thm. 2.

Given segments $h_l^{\mathrm{vis}}(x)$, ray-shooting queries for rays $h_r^{\mathrm{ext}}(u)$ can be answered in $O(\mathrm{sort}(N))$ I/Os using the distribution sweeping paradigm [13]: The slabs used in the recursive partition of the plane are parallel to h_r. The sweep, however, still proceeds in positive y-direction. For every slab σ, maintain the segment $\mu(\sigma)$ among all segments $h_l^{\mathrm{vis}}(x)$ seen so far which maximizes $H_l(x)$ and completely spans σ. When the sweep reaches a new point u, determine the slab σ containing u, decide whether $h_r^{\mathrm{ext}}(u)$ hits $\mu(\sigma)$ and update $\mu(\sigma')$ for all slabs σ' completely spanned by $h_l^{\mathrm{vis}}(u)$.

Given segments $h_l^{\mathrm{vor}}(x)$, $h_r^{\mathrm{vor}}(x)$, and the obstacle edges, it now takes sorting and scanning to extract the vertex and edge sets of $CVD_c(O)$.

[3] Remember that H_l is directed to the left.

References

1. P. K. Agarwal, J. Erickson. Geometric range searching and its relatives. *Advances in Disc. and Comp. Geom.*, pp. 1–56. AMS, 1999.
2. L. Arge, O. Procopiuc, S. Ramaswamy, T. Suel, J. S. Vitter. Theory and practice of I/O-efficient algorithms for multidimensional batched searching problems. *Proc. SODA*, 1998.
3. L. Arge, V. Samoladas, J. S. Vitter. On two-dimensional indexability and optimal range search indexing. *Proc. PODS'99*, 1999.
4. L. Arge. The buffer tree: A new technique for optimal I/O-algorithms. *Proc. WADS*, pp. 334–345, 1995.
5. L. Arge, P. B. Miltersen. On showing lower bounds for external-memory computational geometry problems. In J. Abello and J. S. Vitter (eds.), *External Memory Algorithms and Visualization*. AMS, 1999.
6. L. Arge, D. E. Vengroff, J. S. Vitter. External-memory algorithms for processing line segments in geographic information systems. *Proc. ESA*, pp. 295–310, 1995.
7. L. Arge, J. S. Vitter. Optimal dynamic interval management in external memory. *Proc. FOCS*, pp. 560–569, 1996.
8. S. Arya, G. Das, D. M. Mount, J. S. Salowe, M. Smid. Euclidean spanners: Short, thin, and lanky. *Proc. STOC*, pp. 489–498, 1995.
9. Y.-J. Chiang, M. T. Goodrich, E. F. Grove, R. Tamassia, D. E. Vengroff, J. S. Vitter. External-memory graph algorithms. *Proc. SODA*, 1995.
10. P. B. Callahan, S. R. Kosaraju. A decomposition of multi-dimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Proc. STOC*, pp. 546–556, 1992.
11. K. L. Clarkson. Approximation algorithms for shortest path motion planning. *Proc. STOC*, pp. 56–65, 1987.
12. M. Fischer, T. Lukovszki, M. Ziegler. Geometric searching in walkthrough animations with weak spanners in real time. *Proc. ESA*, pp. 163–174, 1998.
13. M. T. Goodrich, J.-J. Tsay, D. E. Vengroff, J. S. Vitter. External-memory computational geometry. *Proc. FOCS*, 1993.
14. S. Govindarajan, T. Lukovszki, A. Maheshwari, N. Zeh. I/O-efficient well-separated pair decomposition and its applications. *Proc. ESA*, pp. 220–231, 2000.
15. Y. Hassin, D. Peleg. Sparse communication networks and efficient routing in the plane. *Proc. PODC*, 2000.
16. J. M. Keil, C. A. Gutwin. Classes of graphs which approximate the complete Euclidean graph. *Discrete & Computational Geometry*, 7:13–28, 1992.
17. C. Levcopoulos, G. Narasimhan, M. Smid. Efficient algorithms for constructing fault-tolerant geometric spanners. *Proc. STOC*, pp. 186–195, 1998.
18. T. Lukovszki. New results on fault tolerant geometric spanners. *Proc. WADS*, pp. 193–204, 1999.
19. J. Ruppert, R. Seidel. Approximating the d-dimensional complete Euclidean graph. *Proc. CCCG*, pp. 207–210, 1991.
20. D. E. Vengroff, J. S. Vitter. Efficient 3-D range searching in external memory. *Proc. STOC*, 1996.
21. J. S. Vitter. External memory algorithms. *Proc. PODS*, pp. 119–128, 1998.
22. J. S. Vitter, E. A. M. Shriver. Algorithms for parallel memory I: Two-level memories. *Algorithmica*, 12(2–3):110–147, 1994.
23. A. C. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM J. Comp.*, 11:721–736, 1982.