Approximate Nearest Neighbor Queries among Parallel Segments

Ioannis Z. Emiris* Theocharis Malamatos† Elias Tsigaridas‡

Abstract

We develop a data structure for answering efficiently approximate nearest neighbor queries over a set of parallel segments in three dimensions. We connect this problem to approximate nearest neighbor searching under weight constraints and approximate nearest neighbor searching on historical data in any dimension and give efficient solutions for these as well.

1 Introduction

Nearest neighbor searching is a fundamental geometric problem with applications in many areas. For high dimensions there are no known efficient exact solutions and thus approximate solutions to the problem have been studied. Let $d(p, q)$ denote the euclidean distance between points p, q. Given a set P of points in $\mathbb{R}^d$ and a parameter $\varepsilon > 0$, we say that a point p of P is an *ε-approximate nearest neighbor* (ε-NN) to a point q if $d(p, q) \leq (1+\varepsilon)d(q', q)$ where q' is a nearest point to q in P. Arya et al. [3] have shown how to find efficiently an ε-NN to any given query point in constant dimensions and Indyk and Motwani [6] presented efficient methods for high dimensions. See [5] for more references.

An interesting generalization of the problem arises if we replace the point set P with a set of objects O. For this version there are only few results known. When O is a set of disjoint polyhedra in three dimensions Koltun and Sharir [7] presented a data structure of near quadratic size that can answer an ε-NN query in $O(\log(n/\varepsilon))$ time. In three dimensions again, Wang [9] showed how to answer ε-NN queries when O is a set of triangles, segments, and points in a convex position in $O(\log^2 n/\varepsilon^2)$ query time and using $O(n/\varepsilon^2)$ space. In high dimensions, Magen [8] provided an algorithm for a set of k-flats with query time polynomial in d, $\log n$ and $1/\varepsilon$ but super-polynomial space.

The rest of the paper is structured as follows: Sec. 2 presents the problem and we give a first solution in Sec. 2.1. Sec. 2.2 presents an enhanced solution that saves a log factor both in space and time using results in Sec. 3. The latter section is motivated by the *cheap gas station problem*. Given n gas stations (sites) and a car at a position q (query point) we want to find the gas station that is closest or approximately closest to q (since exact distance is not so important) which sells gas for at most w euros. In Sec. 4 we present a data structure for answering time-dependent ε-NN queries. We conclude with a synopsis and on-going work.

2 Methods for parallel segments

Let S be a set of n disjoint segments in $\mathbb{R}^3$. We assume wlog that all segments in S are parallel to the x-axis. For simplicity we also assume that they are in general position. Let P be the set of the $2n$ endpoints of the segments in S. Let q be a query point in $\mathbb{R}^3$, s be a segment of S nearest to q, and p be the point of s nearest to q. Observe that p is either one of the endpoints of s or that the segment qp is perpendicular to s. Let H_q be the plane passing through q that is parallel to the yz plane. Note that if p is interior to s then p is one of the points in $H_q \cap S$. (See Fig. 1)

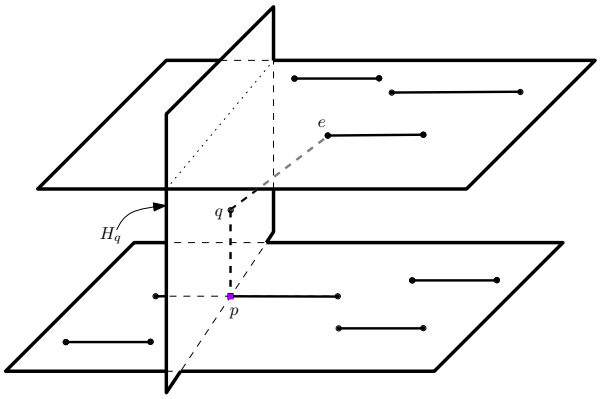

Figure 1: Segments that lie in two parallel planes in $\mathbb{R}^3$ and a query point q. The endpoint nearest to q is e; however, the closest point to q is p.

It follows that to find an ε-NN to q in S it suffices to (a) find an ε-NN to q in P, (b) find an ε-NN to q in the set $H_q \cap S$ and then report whichever of the two is nearest to q. For solving (a) we use an (t, ε)-AVD on set P with $t = O(1/\varepsilon)$ together with the associated data structure [2]. This structure has $O(n)$ space and it returns an ε-NN to any q in $O(\log n + 1/\varepsilon)$ time. For solving (b) we present two methods which are both based on a variation of the well-known interval

*Dept. of Informatics and Telecommunications, National and Kapodistrian University of Athens, Greece.

†Dept. of Computer Science and Technology, University of Peloponnese, Greece.

‡Dept. of Computer Science and Technology, University of Peloponnese, Greece.

tree [4]. The second method is more complicated but leads to a significant improvement.

2.1 First method

The construction of our interval tree T on S proceeds as follows. Let x_m be the median among all x-coordinates of P. Let H_m be the plane passing through point $(x_m, 0, 0)$ and which is parallel to the yz plane. We store x_m at the root of T. We partition the set of segments S into three sets S_ℓ, S_m, and S_r where each set consists of the segments that lie on the left of H_m, that intersect H_m, and that lie on the right of H_m, respectively. We continue the construction of the tree T recursively on S_ℓ and S_r. (If S_ℓ or S_r is the empty set clearly we get a leaf.) The two roots of the trees built on S_ℓ and on S_r become the left and right child of the root of T, respectively. See Figs. 2 and 3 for an example.

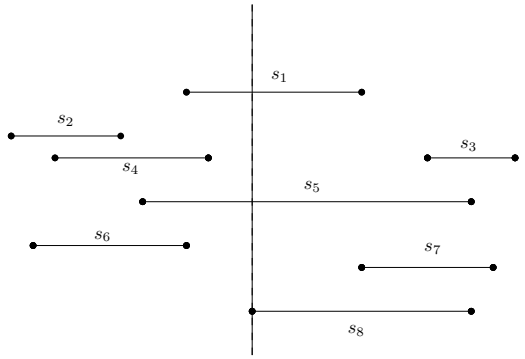

Figure 2: A projection of a set of segments in xz plane. The left endpoint of s_8 is the median of the x-coordinates and the vertical line through it divides the segments to three sets. Segments that are completely on the left, S_l, that are completely on the right, S_r, and that stab the line, S_m.

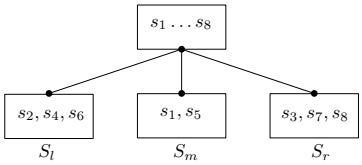

Figure 3: The tree corresponding to the partition of the segments in Fig. 2.

For the set S_m we build an auxiliary data structure T_m which we associate with the root of T. (Structures similar to T_m are built for all internal nodes of T.) T_m is built as follows. H_m cuts naturally each segment in S_m into two pieces. Let C_ℓ be the left pieces of the segments and C_r the right pieces. We build one tree on C_ℓ and one on C_r. We describe the construction of the tree only for C_r since it is symmetric for C_ℓ.

Let x'_m be the median among all x-coordinates of the right endpoints of the segments in C_r. (Note that the x-coordinates of the left endpoints are all equal.) Let H'_m be the plane passing through the point $(x'_m, 0, 0)$ and which is parallel to the yz plane. We store x'_m at the root. The segments of C_r that were not cut by H'_m form the set S'_ℓ. The right pieces of the segments cut by H'_m form the S'_r. The left pieces (which span between planes H_m and H'_m) form the set S'_m. For S'_ℓ, S'_r we continue the construction recursively (unless empty), much like we built our interval tree T above. We refer the reader to Fig. 4 for an example of such a construction. We use the set of segments S'_m to construct a 2D Voronoi diagram for the point set $H'_m \cap S'_m$. Then this Voronoi diagram is combined with a standard point location algorithm [4] to gives us a data structure T_2 for answering optimally 2D nearest neighbor queries over $H'_m \cap S'_m$. T_2 is associated with the root of the tree for C_r. Structures similar to T_2 are also built for all internal nodes of T_m.

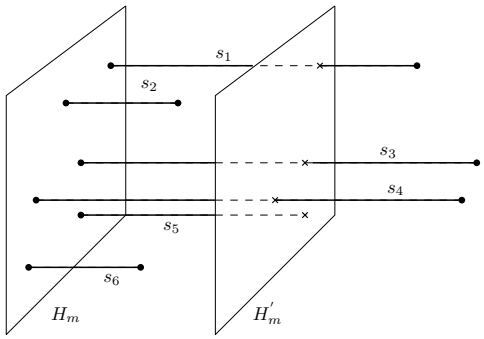

Figure 4: The left endpoint of s_5 is the median of the left x coordinates. The parts of segments s_1, s_2 and s_3 that are on the left of H'_m form the set S'_r. The parts between H_m and H'_m, together with s_5 form S'_m. The set S'_ℓ contains the segments s_2 and s_6.

We compute a bound on the size of T_m, that is, for the augmented trees on C_ℓ and on C_r. Let $n' = |S_m|$. Because we split always at the median, the height of T_m is $O(\log n')$. This implies that one segment of S_m may be cut at most $O(\log n')$ times and thus the total size of T_m is $O(n' \log n')$. Using standard results, it is easy to see that the total size of all structures T_2 associated with the nodes of T_m is also $O(n' \log n')$.

We compute a bound on the size of the main data structure T. Since the set of segments S_m used at each node of T for the auxiliary data structure T_m are disjoint and using the space bound on T_m it follows easily that the total size of T is $O(n \log n)$. Due to the balanced splits T has height $O(\log n)$.

We describe now how to solve (b) that is, given a query $q = (q_x, q_y, q_z)$, how to find an ε-NN to q in the set $H_q \cap S$. (In fact this first method finds an exact nearest neighbor to q). We start at the root of

T and at each node v we follow the child according to the result of the comparison between q_x and x_m, the value stored at v. At each node v we also visit the auxiliary data structure T_m. We similarly follow the path from the root of T_m to the leaf containing q and at each node we use T_2 to find the nearest neighbor to (q_y, q_z) among $H'_m \cap S'_m$. We report as an answer the nearest point to q over all the points returned from all queries to T_2.

Correctness follows from the fact that we only exclude from consideration segments or fragments of segments that do not intersect plane H_q and thus are not needed for (b).

Since the depth of tree T is $O(\log n)$ and for each node of T we visit an auxiliary data structure T_m with depth also at most $O(\log n)$ and at each node of T_m data structure T_2 may have been built for at most n sites, it follows that query time is $O(\log^3 n)$. The total query time is the sum of the time spend on solving (a) and that on (b) and thus we get the following result:

Theorem 1 *Given n parallel segments in 3D we can construct a data structure of $O(n \log n)$ space for finding an ε-NN to any given query point q in $O(\log^3 n + 1/\varepsilon)$ time.*

We will discuss the construction times of the data structures in the final version, however they are all in $O(n \operatorname{poly}(\log n, \frac{1}{\varepsilon}))$.

2.2 Improving space and query time

Next we show a second method that significantly reduces the space and query time in terms of n. The part of the first method that we change is the auxiliary data structure T_m for the sets C_ℓ and C_r. We again describe just the data structure T_r for segments in C_r and an analogous data structure can be built for C_ℓ.

According to (b), our goal for C_r is to find an ε-NN to q in $H_q \cap C_r$. We solve this by reducing our problem to a weight-constrained approximate nearest neighbor searching problem in two dimensions. Specifically each right endpoint (p_x, p_y, p_z) of a segment in C_r is mapped to the point (p_y, p_z) with weight p_x. We denote with P' the 2D weighted point set obtained (see Fig. 5). Let P'_w be the subset of P' containing only the points with weight at least w. Given a query $q = (q_x, q_y, q_z)$, our goal is to find an ε-NN to point $q_2 = (q_y, q_z)$ in P'_{q_x}. Note that this suffices to achieve our first goal.

We apply the Theorem 4 of the next section (weight-constrained ε-NN queries) on point set P' for $d = 2$, $\gamma = 2$ and $q = q_2$ and easily get this lemma:

Lemma 2 *Given q and C_r we can build a data structure T_r of $O(|C_r|)$ size to find an ε-NN to q in $H_q \cap C_r$ in $O(\log |C_r| + 1/\varepsilon^2)$ time.*

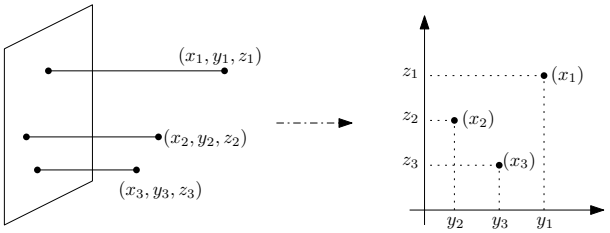

Figure 5: The values of x_i's on the right hand-side play the role of weights, hence the points are considered weighted.

Using the above lemma we obtain the following result: (We omit the analysis which is similar to that of the first method.)

Theorem 3 *Given n parallel segments in 3D we can construct a data structure of $O(n \log(1/\varepsilon))$ space for finding an ε-NN to any given query point q in $O(\log^2 n + \log n/\varepsilon^2)$ time.*

3 Weight-constrained ε-NN queries

Given a set of weighted points in d dimensions, the *weight-constrained ε-approximate nearest neighbor* problem is to find, given a query q and a weight w, an ε-NN to q among the points in P that have weight at least w. Here w is a number that is specified at query time. Note that we allow an approximation error in one parameter (distance) but we require exactness on one other (weight). We can also define the symmetric problem where we search for an ε-NN among the points of P with weight at most w.

Theorem 4 *Let P be a set of n points in $\mathbb{R}^d$, and let $0 < \varepsilon < 1/2$ and $2 \le \gamma \le 1/\varepsilon$ be two real parameters. We can construct a data structure of $O(n\gamma^d \log(1/\varepsilon))$ space that allows us to answer a weight-constrained ε-NN query in time $O(\log(\gamma n) + 1/(\varepsilon\gamma)^d)$.*

(Proof sketch.) We start with some definitions. Throughout this section we use notation from [1]. Let $b(q, r)$ be a ball of radius r centered at point q. Let $b^+(q, r)$ be a ball of radius $(1 + \varepsilon)r$ centered at q and $b^-(q, r)$ be a ball of radius $(1 - \varepsilon)r$ centered at q. (We omit the parameters wherever they are clear from context.) Given P, q and r an *ε-range counting query* returns a number that is between $|P \cap b^-|$ and $|P \cap b^+|$. That is we approximate the range b with some range between b^- and b^+ and count the points in that range. An *ε-range maximum query* returns the weight of a point contained in b^+ that has weight at least as large as the point of maximum weight in b^-. Given q and k an *ε-kth nearest neighbor query* returns a point contained in the annulus formed by the two balls $b^-(q, r_k)$ and $b^+(q, r_k)$ where r_k is the distance to a kth nearest neighbor to q.

We now give a method for solving our problem. This method is an extension of the methods presented for answering ε-approximate range queries and particularly for answering ε-kth nearest neighbor queries [1]. The query input in our case instead of the query point q and the integer k is q and the one-sided constraint on the weights w.

We follow closely [1] (Section 4, improved approach). A difference is that instead of precomputing an $\varepsilon/4$-range counting query for the smallest and largest balls handled by u that are centered at a point inside z we use an $\varepsilon/4$-range maximum query. Let w_s and w_ℓ be the answers (weights) returned for the small ball and the large ball query respectively. At query time, we determine the hypercube z containing q and similarly use the two weights w_s and w_ℓ stored with z to determine if u is responsible for answering our weight-constrained ε-NN query. Here u is responsible for giving the answer if and only if $w_s \leq w \leq w_\ell$. If $w < w_s$ we visit the related node responsible for handling smaller ranges while if $w > w_\ell$ we visit the related node for larger ranges.

We compute the answer to the query by processing the last node u visited as follows. We consider first the case where u is a leaf. Each box $z \in \mathcal{Q}(u)$ is associated with the point having the largest weight in $S \cap z$. The points in $\mathcal{P}(u)$ maintain their original weight. We simply scan the points of $\mathcal{P}(u)$ and the points associated with boxes in $\mathcal{Q}(u)$ and report the point nearest to q among those having weight at least w. A similar approach also works if u is a box cell obtained by shrinking. The last case is if u is a box cell obtained by splitting. Similarly we use a binary search to determine an ε-NN to q among the points with weight at least w. As before instead of the count for a given spherical range with center q, we compute the maximum weight for this range and determine accordingly the next range to check. The space and time analysis in [1] applies to our case too and thus we obtain the theorem above.

4 Querying about the past

We define a *timestamped* operation on a data structure as an operation which carries a label of the time it occurred. An element is *alive* at some moment t if t is between the time of insertion and deletion of the element. We consider the following problem: given a sequence of n timestamped insertions and deletions of d-dimensional points build a data structure which given a query point q and a parameter t finds efficiently an ε-NN of q among the points alive at time t. We call this a *t-moment ε-NN query.*

We tackle the problem using methods from Sec. 2 and 3. Let p be a point that was inserted at time t_s and deleted at time t_f (infinite values are allowed). We use time as an extra dimension and map the point p to the segment with endpoints (t_s, p) and (t_f, p) in $d+1$ dimensions. Note that the points alive at any given moment t correspond to the segments intersecting the hyperplane with equation $x = t$. Hence we can build a similar interval tree as in Sec. 2. To answer queries part (b) is only needed. We extend Lemma 2 in d dimensions and with a simple analysis get the following:

Theorem 5 *Let n timestamped insertions and deletions of points in $\mathbb{R}^d$, and let $0 < \varepsilon < 1/2$ be a real parameter. We can construct a data structure of $O(n \log(1/\varepsilon))$ space that allows us to answer any t-moment ε-NN query in time $O(\log^2 n + \log n/\varepsilon^d)$. Here t is given at query time.*

5 Conclusion

We have shown how to answer efficiently ε-approximate nearest neighbor queries over a set of parallel segments in $\mathbb{R}^3$ and we have presented applications of this result to interesting approximate searching problems in higher dimensions. The result holds also for segments with a fixed number of directions if we allow the query time to grow up by a constant factor. We are currently investigating answering ε-NN queries among fat triangles and polyhedra generated by the above segments.

References

[1] S. Arya, T. Malamatos, and D. M. Mount. Space-time tradeoffs for approximate spherical range counting. In *Proc. 16th Annu. ACM-SIAM Sympos. Discrete Algorithms*, pages 535–544, 2005.

[2] S. Arya, T. Malamatos, and D. M. Mount. Space-time tradeoffs for approximate nearest neighbor searching. *J. Assoc. Comput. Mach.*, 57:1–54, 2009.

[3] S. Arya, D. M. Mount, N. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *J. Assoc. Comput. Mach.*, 45:891–923, 1998.

[4] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, Germany, 3rd edition, 2008.

[5] P. Indyk. Nearest neighbors in high-dimensional spaces. In *in Handbook of Discrete and Computational Geometry*, pages 877–892. CRC Press, 2004.

[6] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. 30th Annu. ACM Sympos. Theory Comput.*, pages 604–613, 1998.

[7] V. Koltun and M. Sharir. Polyhedral Voronoi diagrams of polyhedra in three dimensions. In *SCG '02: Proceedings of the eighteenth annual symposium on Computational geometry*, pages 227–236, 2002.

[8] A. Magen. Dimensionality reductions in ℓ_2 that preserve volumes and distance to affine spaces. *Discrete Comput. Geom.*, 38(1):139–153, 2007.

[9] Y. Wang. Approximating nearest neighbor among triangles in convex position. *Inf. Process. Lett.*, 108(6):379–385, 2008.