Approximate Nearest Neighbor Algorithms for Frechet Distance via Product Metrics

Piotr Indyk
MIT

Categories and Subject Descriptors

F.2.2 [**Theory of Computation**]: Nonnumerical Algorithms and Problems

General Terms

Algorithms

1. INTRODUCTION

The *Nearest Neighbor Search* (NNS) problem is: Given a set P of n points in a metric space X, preprocess P so as to efficiently answer queries for finding the point in P closest to a query point q. NNS and its approximate versions are among the most extensively studied problems in the fields of Computational Geometry and Algorithms, resulting in discovery of many efficient algorithms. In particular, for the case when the metric space X is a low-dimensional Euclidean space l_2^d, it is known how to construct data structure for exact [5, 15] or approximate [4, 13, 12, 9] NNS with query time $(d+\log n)^{O(1)}$. Unfortunately, those data structures require space exponential in d. More recently, several data structures using space (quasi)-polynomial in n and d, and query time sublinear in n, have been discovered for approximate NNS under l_1 and l_2 [14, 12, 11] and l_∞ [10] norms.

While many metrics of interest for nearest neighbor search are norms, quite a few of them are not. A prominent example of the latter is the *Frechet metric*. Unlike the norms, Frechet metric is defined for *sequences* of points (possibly infinite and of different length), not vectors. More specifically, the Frechet distance between two sequences of points is defined as a minimum, over all alignments between the two sequences, of the maximum distance between two corresponding points (see Preliminaries for formal definition). Frechet metric is a natural measure of similarity between sequences of points (e.g., handwritten signatures). It also has been studied in the Computational Geometry literature, e.g., in [2] (see also [3]). Unfortunately, the optimum alignment between any two sequences is not fixed (i.e., it could be different for different sequences), which means that the the NNS problem for Frechet metric is much harder than NNS for normed spaces. In fact, it is not difficult to see that the Frechet metric "contains" l_∞ norm, so NNS for the latter norm can be viewed as a special case of NNS for Frechet metric.

A potential approach for devising an approximate NNS data structure for Frechet metric is to *embed* it into a (low-dimensional) normed space. In other words, we would like to map every sequence into a vector such that the distance between the vectors approximates the distance between the sequences. This approach has been successfully applied for some not-normed metric spaces. In particular, it has been shown [8] that the Hausdorff metric (which measures similarity between *sets* of points) can be embedded with arbitrarily small distortion into l_∞; it was also shown that a variant of the edit distance between sequences of length up to m can be embedded into Hamming space with distortion roughly $O(\log m)$ [7, 16, 6] . Although it is plausible that Frechet metric can be embedded into l_∞ with low dimension and distortion, so far researchers were unable to provide such an embedding.

In this we present a different approach for designing algorithms for not-normed metrics using *product metrics*. A product of metrics $M_1, \ldots M_k$ with distance functions $D_1, \ldots D_k$ is a metric over $M_1 \times \ldots \times M_k$ with distance function D such that $D((p_1, \ldots, p_k), (q_1, \ldots, q_k)) = \max_{i=1}^k D_i(p_i, q_i)$. We show that if we are given a data structure for c-approximate (variant of) nearest neighbor problem in metrics M_i (with $Q(n)$ query time and $S(n)$ space), then we can construct a data structure solving $O(c \log\log n)$-approximate nearest neighbor in the product M of $M_1, \ldots, M_k$. The data structure for M achieves query time roughly $O(Q(n)\log n + k\log n)$ and space $O(kS(n)n^{1+\delta})$, for any constant $\delta > 0$.

In the second step, we show that NNS for Frechet metric can be reduced to NNS over a product of several Frechet metrics over shorter sequences. By applying the above algorithm, we obtain an $O((\log m + \log\log n)^{t-1})$-approximate nearest neighbor data structure for Frechet metric, for any constant $t > 1$ (m is the maximum length of a sequence). The data structure uses space exponential in $tm^{1/t}$ and achieves $(m + \log n)^{O(t)}$ query time (see section 4 for details). For comparison, the only previously known algorithm achieving $(m+\log n)^{O(1)}$ query time used space exponential in m. Thus, although our data structure uses space super-polynomial in n, the bound is much better than the one previously known.

The NNS data structure for product metrics is also of sig-

SoCG'02, June 5-7, 2002, Barcelona, Spain.

nificant interest by itself. Product metrics are of interest in scenarios where one searches for an object satisfying a set of metric constraints, e.g., “Find a person with similar signature and similar face photo”. We can answer such a query by using NNS data structure for a product of individual metrics. Product metrics has been investigated earlier in [1] and [10]. The only result similar to ours, mentioned in [10], proceeded by embedding each metric into l_∞^d with $d = n^{1/c}$ and distortion $O(c)$, and then applying NNS algorithm for l_∞^{dk}. This resulted in an algorithm with either query time polynomial in n, or approximation error $O(\log n)$; none of these bounds can be used to achieve an interesting result for Frechet metric. In addition, the embedding procedure required almost-exact nearest neighbor data structure for individual metrics, while our algorithm works for any approximation error.

Our techniques. Our most technically involved result is the data structure solving NNS for product metrics. This data structure can be viewed as a generalization of the data structure for the l_∞ norm [10], to arbitrary product spaces. The generalization is somewhat involved, since the data structure of [10] used heavily the geometry of the l_∞ space (e.g., existence of plane separators). In contrast, the data structure given in this paper uses only metric properties of the product space. It is interesting to note that the l_∞ data structure guaranteed error of $O(\log\log d)$ when using $dn^{O(1)}$ space, while our data structure guarantees a comparable error of $O(\log\log n)$ while being much more general.

2. PRELIMINARIES

For a metric space $M = (X, D)$, we define the *Frechet metric* $M_F = (X^*, D_F)$ over M is defined over the set of all sequences of points in X. For any two sequences $p = p_1, \ldots p_k$ and $q = q_1, \ldots q_l$, we define

$$D_F(p,q) = \min_s \min_{f,g} \max_{t=1\ldots s} D(p_{f(t)}, q_{g(t)})$$

where the functions $f : \{1\ldots s\} \to \{1\ldots k\}$ and $g : \{1\ldots s\} \to \{1\ldots l\}$ are such that $g(1) = f(1) = 1$, $f(s) = k$, $g(s) = l$. Moreover, f, g are non-decreasing and the difference between any of their two consecutive values of is at most 1.

The above definition is a generalization of the continuous Frechet metric [2] defined for curves to arbitrary sequences of points. In particular, we can express the Frechet distance between two continuous curves as a distance between sequences of uniformly spaced samples taken from the curves, where the number of samples tends to infinity. This approach allows us to use the “discrete” Frechet distance for continuous curves as well. We note that in this case the exponent in the space bound for our data structure would depend linearly on the *complexity* of the curves (e.g, number of segments in a polygonal line), but its dependence on the number of samples would be only logarithmic.

In this paper we focus on solving a decision version of the approximate nearest neighbor problem, called *Priority Point Location in Equal Balls* (or $(1+\epsilon)$-PPLEB). The problem is defined as follows. Assume we are given a set P of n points in a metric space, $r > 0$ and $\epsilon > 0$. Each points has a priority associated with it, which is an integer from $\{1\ldots n\}$. The goal is to create a data structure, that given a query point q:

- if there exists a database point p within distance r from q and priority t, then the data structure returns a point p' within distance $(1+\epsilon)r$ from q with priority value at most t
- if there is no database point within distance $(1+\epsilon)r$ from q, the data structure returns “NONE”

As we will see in the next section, we need the priority version of the (more standard) PLEB data structures for the metrics M_i in order to achieve our results. The reason for that is fairly technical and it is likely that this constraint can be removed. In the meantime, we mention that all known algorithm for approximate PLEB from [14, 12] can be quite easily modified to work for the priority version of the problem.

By the results of [12, 9], the $(\delta + c)$-approximate nearest neighbor problem can be reduced to c-PPLEB (even without priorities), for any constant $\delta > 0$, with only polylogarithmic overhead in space and query time.

Computational assumptions. We use d to denote the “complexity” of each point from an individual metric space M_i, i.e., the amount of storage needed to represent such a point. We assume that computing a distance between two points in M_i can be done in $O(d)$ time.

For simplicity, we assume that $k, m \le n$.

3. PRODUCT METRICS

THEOREM 1. *Assume that we are given a data structure for c-PPLEB in metrics M_i, which achieve $Q(n)$ query time using $S(n)$ space. Then we can construct a data structure solving $O(c \log\log n)$-PPLEB in the product metric $M_1 \times \ldots \times M_k$, with query time $O(Q(n)\log n + kd\log n)$ and space $O(kS(n)n^{1+\delta})$, for any constant $\delta > 0$.*

Proof: For simplicity we focus on the case $\delta = 2$, the modifications needed to achieve any $\delta > 0$ are quite straightforward. Also, by scaling, we assume $r = 1$.

The data structure is based on a search tree. Each node of the tree corresponds to a subset (say S) of P. The root node corresponds to $S = P$. The children of the node are constructed as follows. Let $R = 2(c+1)\log\log n + 1$. For any $p = (p_1, \ldots, p_k) \in P$, let $p_{|i}$ denote the point p_i from M_i. We also extend the projection operator $|i$ to sets of points. The algorithm proceeds by considering the following three cases:

Case 1: For all $i = 1\ldots k$, there exists $p = p(i) \in S$ such that

$$\frac{|B(p_{|i}, R) \cap S_{|i}|}{|S|} \ge 1 - 1/2k$$

Case 2: There exists i such that for all points $p \in S$ we have

$$\frac{|B(p_{|i}, R/2) \cap S_{|i}|}{|S|} \le 1/2$$

Case 3: There exists i and a point $p \in S$ such that

$$\frac{|B(p_{|i}, R/2) \cap S_{|i}|}{|S|} \ge 1/2$$

and

$$\frac{|B(p_{|i}, R) \cap S_{|i}|}{|S|} \le 1 - 1/2k$$

In the following we describe the actions performed by the algorithm for the above 3 cases.

Case 1. We first construct the point $p = (p(1)_{|1}, \ldots, p(k)_{|k})$. Clearly, we have $\frac{|B(p,R) \cap S|}{|S|} \geq 1/2$. Our data structure stores the point p and the point p' in $|B(p,R) \cap S|$ with the smallest priority value. Then it removes the points in $B(p,R)$ from S (creating S'), creates a child node corresponding to S' and recursively constructs the tree for S'.

Case 2. In this case we partition the set S into disjoint collection of subsets. The partitioning criteria are determined entirely by the i-th coordinate of points in S. Therefore, we will show how to partition points in $S_{|i}$ and this will induce partitioning of S, by assuming a one-to-one correspondence between points in these two sets.

The subsets are generated iteratively. We start each iteration step by finding a point $s \in S_{|i}$ and a radius $R(s)$ such that

$$[\frac{|B(s, R(s)+c+1) \cap S_{|i}|}{|S_{|i}|}]^2 \leq \frac{|B(s, R(s)) \cap S_{|i}|}{|S_{|i}|}$$

and

$$\frac{|B(s, R(s)+c+1) \cap S_{|i}|}{|S_{|i}|} \leq 1/2$$

It is not difficult to see that such a radius $R(s)$ exists. We start from $R(s) = 0$. If the above equations are satisfied, we are done. Otherwise, we have a lower bound on $|B(s, c+1) \cap S_{|i}|$. Then we try again with $R(s) = c+1$ etc. It is not difficult to see that if each candidate for $R(s)$ was unsuccessful, then after $\log\log n$ steps we would have $|B(s, R(s))| > n/2$. Which is a contradiction, since $R/2 > (c+1)\log\log n$.

After finding $R(s)$ satisfying the above constraints, we remove the set $S_1 = S_{|i} \cap B(s, R(s))$ from $S_{|i}$. Then we construct $S_2, S_3 \ldots$ and so on, until each point belongs to exactly one set S_j.

The data structure is now constructed as follows. Firstly, we create a c-PPLEB data structure (with radius 1) for points in $S_{|i}$. A point from S_j gets priority j. Moreover, for each set S_j, we create a child which contains all points in $S_{|i} \cap \cup_{s \in S_j} B(s, c+1)$, and apply the procedure recursively on those nodes.

Case 3. Let $p \in S$ be a points satisfying the conditions for Case 3. In this case we find a radius $R' \geq R/2$, such that

$$\frac{|S_{|i} - B(p, R'+2)|}{|S|} \geq 1/2k$$

and

$$[\frac{|S_{|i} - B(p, R')|}{|S|}]^2 < \frac{|S_{|i} - B(p, R'+2)|}{|S|}$$

The existence of such R' follows by an argument as in Case 2.

In this case, the data structure stores the point p. Then we create two sets $S_{|i} \cap B(p_{|i}, R'+2)$ and $S_{|i} - B(p_{|i}, R')$ as children, and apply the construction procedure recursively on those two nodes.

Query procedure. Given a query point $q = (q_1, \ldots, q_k)$, we answer it by traversing the tree from the root to the leaf. In order to select which child to pursue from a given node S, we use the proper procedure depending on the type of the node.

Case 1. Compute the distance from q to p. If this distance is at most $R+1$, we store p' (note that p' is within distance $2(R+1)$ from q) and then recurse on points in $S - B(p, R)$. If the recursive procedure reports p'', we retrieve one of the points p', p'' with the smaller priority value.

If the distance from q to p is greater than $R+1$, then none of the points in $B(p,R)$ can lie within distance 1 from q. Thus we immediately recurse on points in $S - B(p,R)$.

Case 2. Perform one c-PPLEB query with query $q_{|i}$. If the answer is NONE, then report NONE. Otherwise, let s be a point in S such that $s_{|i}$ is within distance c from $q_{|i}$, and let S_j be the set containing s. In this case we proceed to the child containing all points $C_j = S_{|i} \cap \cup_{s \in S_j} B(s, c+1)$.

Note that if there is any point in S within distance 1 from $q_{|i}$, then this point is surely contained in the set C_j. To see this, let s' be such point with the lowest priority value t. The point s returned by PPLEB has priority value $j \leq t$ (this is the only place where we need to use priorities). Thus, when the set S_j was created, all points within distance 1 from $q_{|i}$ were still contained in S (since the first one of them has been removed only after step t). Thus, the set C_j contains all such points.

Case 3. Compute the distance from $q_{|i}$ to $p_{|i}$. If this distance is at most $R'+1$, then proceed to $S_{|i} \cap B(p_{|i}, R'+2)$. Otherwise, proceed to $S_{|i} - B(p_{|i}, R')$. Again, observe that if there is any point in S within distance 1 to q, then this point is contained in the proper child set as well.

This concludes description of the query answering procedure, as well as its correctness. It remains to bound the query time and space used.

Query time. Observe that the number of PPLEB queries performed during each search is $O(\log n)$, since each time we perform such a query, the child of the current node contains at most $1/2$ fraction of the points in the current node. Moreover, the number of times when Case 1 occurs is also $O(\log n)$ by the same argument, which adds $O(dk \log n)$ to the cost. Finally, Case 3 occurs at most $O(k \log n)$ times, adding $O(dk \log n)$ to the total cost.

Space. Observe that we can assume that only cases 2 and 3 occur. We focus only on case 2, the analysis of case 3 is similar.

Assume the set S is partitioned into $S_1, \ldots S_l$. Define $m_j = |S_j|$, $m'_j = |S_{|i} \cap \cup_{s \in S_j} B(s, c+1)|$ and $|S| = m$. We know that $\sum_j m_j = m$ and $m_j/m > (m'_j/m)^2$ for all j.

Let $S(m)$ denotes the number of *nodes* in the tree. Assume (inductively) that the space $S(t) \leq t^2$ for all $t < m$. Since $m_j < m$, we know that

$$S(m) \leq \sum_j S(m'_j) + 1 \leq \sum_j {m'}^2{}_j + 1 < m \sum_j m_j + 1 \leq m^2$$

Thus, the total space used by the tree is at most $O(n) \cdot S(n) \leq O(n^3)$. It is easy to improve the bound to $O(n^{1+\delta})$ for any constant $\delta > 0$ by using more elaborate analysis.

This concludes the proof.

4. FRECHET DISTANCE

In this section we show how to construct an approximate PPLEB for the discrete Frechet distance, where all query and database sequences have length at most m.

THEOREM 2. *For any integer $t > 1$, there is a data structure for $O((\log m + \log\log n)^{t-1})$-PPLEB for Frechet metric over $M = (X, D)$, with $O(m^2|X|)^{tm^{1/t}} n^{2t}$ space and $(m + \log n)^{O(t)}$ query time.*

Proof: We present the algorithm for case $t = 2$, the algorithm for general t follows by recursive application of the same idea.

Assume we are given sequences $p^1 \dots p^n$. For each p^i we generate all possible partitions of p^i into $k = \sqrt{m}$ possibly empty sequences $p^i_1 \dots p^i_k$. Moreover, for each such partition we generate all *semi-partitions* $pp^i_1 \dots pp^i_k$, where each pp^i_j is obtained from p^i_j by attaching (or not) the last symbol of the sequence $pp^i_1 \dots pp^i_{j-1}$ in front of p^i_j.

Observe that if we partition a query q into k *equal length* parts $q_1 \dots q_k$, then the following Claim holds.

CLAIM 1.

$$D_F(p^i, q) = \min_{pp^i_1, \dots, pp^i_k} \max[D_F(pp^i_1, q_1), \dots, D_F(pp^i_k, q_k)]$$

Proof: We prove the above inequality by showing the "$\leq$" and "$\geq$" parts separately. To show the $\leq$ part, we observe that for any functions f_i, g_i, $i = 1 \dots k$, which define the alignments for $D_F(pp^i_j, q_j)$, the concatenations of $f_1, \dots, f_k$ and $g_1, \dots, g_k$ provide an alignment between p^i and q with cost $\max[D_F(pp^i_1, q_1), \dots, D_F(pp^i_k, q_k)]$.

To show the $\geq$ part, consider the functions f, g defining the alignment corresponding to $D_f(p^i, q)$. We will show how to decompose f, g into $f_1 \dots f_k$, $g_1 \dots g_k$ as above, and p^i into $pp^i_1, \dots, pp^i_k$, such that the maximum of distances between pp^i_j and q_j defined by f_j, g_j is at most $D_F(p^i, q)$. Specifically, we will show how to obtain f_1, g_1, pp^i_1, pp^i_2, since the remaining elements can be obtained in the same way. Let t be the smallest index such that $g(t+1) > m/k$. We define pp^i_1 to consist of the first $f(t)$ symbols of p^i. Moreover, we set $s = t$, $f_1(u) = f(u)$ and $g(u) = g(u)$ for $u = 1 \dots s$. In order to define pp^i_2 we consider two cases:

Case 1: $f(t) < f(t+1)$. In this case, we take the first symbol of pp^i_2 to be $(p^i)_{f(t+1)}$ and proceed as above.

Case 2: $f(t) = f(t+1)$. In this case we take the first symbol of pp^i_2 to be $(p^i)_{f(t)}$. Then we proceed as above. $\Box$.

Thus we can consider semi-partitions $pp^i_1, \dots, pp^i_k$ to be points in a product metric space. Then solving the (approximate) nearest neighbor with respect to all semi-partitions of all p^i's is equivalent to solving the (approximate) nearest neighbor with respect to sequences p^i. The exact PPLEB for the individual metrics in the product metric can solved using $|X|^{m/k} = |X|^{\sqrt{m}}$ space (by storing answers to all queries). Thus, by Theorem 1 (with $\delta = 1$) we obtain a c-PPLEB for D_F with parameters:

- $c = O(\log\log(m^{\sqrt{m}} n)) = O(\log m + \log\log n)$
- query time $O(m^{O(1)} \log n)$
- space $O(|X|^{\sqrt{m}} (m^{\sqrt{m}} n)^2)$

In order to reduce the space, we can now use the above algorithm recursively as a c-PPLEB subroutine and apply Theorem 1.

5. CONCLUSIONS AND OPEN PROBLEMS

In this paper we gave an approximate NNS data structure for general product metrics. Building on that result, we constructed a data structure for approximate NNS for (discrete version of) Frechet metric, which use a significantly subexponential space and achieves very fast query time.

The product-metric-based approach to designing approximate NNS algorithms, presented in this paper, seems to be be quite general and powerful. In an upcoming paper, we show how to apply this approach to construct an $O(1)$-approximate data structure for NNS under the Levenstein (or edit) metric over strings of length up to m. The data structure uses space exponential in $m^{1/t} \log n$, for any constant $t > 1$, and achieves $O(m^{O(1)})$ query time. We also obtain a similar result for the discrete Frechet metric. This improves the approximation factor obtained in this paper, while slightly increasing the exponent in the storage bound (by a factor of $\log n$). Both results are using a data structure for NNS under (more general) product metrics which is different than the algorithm given in this paper.

In order to solve our problem, we needed a discrete version of the Frechet metric. Is it possible to devise a data structure for NNS applicable directly to Frechet metric over continuous curves (without sampling) ? We note that this problem is open even for Hausdorff distance, since the embedding results of [8] hold only for finite sets of points. Interestingly enough, the complexity of the best known algorithm [2] for computing Frechet distance between two polygonal curves of complexity n is essentially the same as the running time of a simple dynamic programming algorithm for computing discrete Frechet distance (i.e., roughly quadratic in n). This suggests that the problems could be mutually reducible to each other, perhaps also in the context of NNS.

6. REFERENCES

[1] P.K. Agarwal, A. Gionis, L. Guibas, and R. Motwani. Rank-based similarity searching. *Manuscript*, 1998.

[2] H. Alt and M. Godau. Computing the frechet distance between two polygonal curves. *International Journal of Computational Geometry and Applications*, 5:75–91, 1995.

[3] H. Alt and L. J. Guibas. Discrete geometric shapes: Matching, interpolation, and approximation. In J.-R. Sack and J. Urrutia, editors, *Handbook on Computational Geometry*, pages 121–153. North-Holland, 2000. To appear.

[4] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 573–582, 1994.

[5] K. Clarkson. A randomized algorithm for closest-point queries. *SIAM Journal on Computing*, 17:830–847, 1988.

[6] G. Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, 2002.

[7] G. Cormode, M. Paterson, C. Sahinalp, and U. Vishkin. Communication complexity of document exchange. *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, 2000.

[8] M. Farach-Colton and P. Indyk. Approximate nearest neighbor algorithms for hausdorff metrics via embeddings. *Proceedings of the Symposium on Foundations of Computer Science*, 1999.

[9] S. Har-Peled. A replacement for voronoi diagrams of near linear size. *Proceedings of the Symposium on Foundations of Computer Science*, 2001.

[10] P. Indyk. On approximate nearest neighbors in l_∞ norm. *Journal of Computer and System Sciences, to appear. Preliminary version appeared in Proceedings of the Symposium on Foundations of Computer Science*, 1998.

[11] P. Indyk. Dimensionality reduction techniques for proximity problems. *Proceedings of the Ninth ACM-SIAM Symposium on Discrete Algorithms*, 2000.

[12] P. Indyk and R. Motwani. Approximate nearest neighbor: towards removing the curse of dimensionality. *Proceedings of the Symposium on Theory of Computing*, 1998.

[13] J. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, 1997.

[14] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. *Proceedings of the Thirtieth ACM Symposium on Theory of Computing*, pages 614–623, 1998.

[15] S. Meiser. Point location in arrangements of hyperplanes. *Information and Computation*, 106:286–303, 1993.

[16] S. Muthukrishnan and C. Sahinalp. Approximate sequence nearest neighbors. *Proceedings of the Symposium on Theory of Computing*, 2000.