

Reductions Among High Dimensional Proximity Problems

ASHISH GOEL*

PIOTR INDYK†

KASTURI VARADARAJAN‡

October 23, 2000

Abstract

We present improved running times for a wide range of approximate high dimensional proximity problems. We obtain *subquadratic* running time for each of these problems. These improved running times are obtained by reduction to Nearest Neighbour queries. The problems we consider in this paper are Approximate Diameter, Approximate Furthest Neighbours, Approximate Discrete Center, Approximate Metric Facility Location, Approximate Bottleneck Matching, and Approximate Minimum Weight Matching.

1 Introduction

Proximity problems are a class of geometric problems, loosely described as those which involve the notion of a distance between points in a d -dimensional space. For example, the closest pair problem, furthest pair (or diameter) problem, many variants of clustering (including MST), and nearest neighbor search all belong to this class. If the dimension d is low, these problems enjoy very efficient (exact or approximate) solutions (e.g. see [29, 4, 26, 1] or the textbooks [28, 27]). However, the running time and/or space requirements of these algorithms grow exponentially with the dimension. This is unfortunate, since the high-dimensional versions of the above problems are of major and growing importance to a variety of applications, usually involving similarity search or clustering; some examples are information retrieval, image and video databases, vector quantization, data mining, and pattern recognition. Therefore, a lot of recent research [5, 24, 19, 20, 25, 17, 2, 18] has focused on algorithms with improved (polynomial) dependence on the dimension.

In this paper we consider several basic proximity problems in Euclidean space; see Section 2 for the precise statement of the problems and Table 1 for

the results. (Throughout this paper, we assume that d is $O(\log n)$ because of the Johnson-Lindenstrauss Lemma [22].) We show that all the problems we consider can be reduced to a small number (usually $\tilde{O}(n)$) of calls to a (possibly dynamic) data structure for $(1 + \epsilon)$ -approximate nearest neighbors. By the results of [19, 20] there is a data structure with (randomized) $\tilde{O}(n^{1/(1+\epsilon)})$ time per query/update. In effect, we obtain significantly improved subquadratic-time algorithms for those problem. Moreover, since our results are obtained via black-box reductions, any future improvement in the running time of the $(1 + \epsilon)$ -approximate nearest neighbor data structure will automatically result in improvements for the problems we consider in this paper. Thus, we are able to reduce a large number of high-dimensional geometric problems to a single primitive. This adds to already known reductions for the approximate Minimum Spanning Tree [19, 20, 2] and Bichromatic Closest Pair [8, 9], which give similar results. Thus we believe that this approach will result in further improvements for other high-dimensional geometric problems.

Our techniques. Our techniques are diverse and depend on the actual problems. For the $(1 + \epsilon)$ approximate *diameter* problem, we exploit the fact that if we have a set of points P on a sphere S , then a furthest neighbor of $q \in S$ in P is also a nearest neighbor of q' in P , where q' denotes the antipode of q .¹ Unfortunately, this reduction does not have to preserve approximation. However, if S is the *minimum enclosing ball* of P , we can show that the approximation guarantee is preserved. The (approximate) minimum enclosing ball can be computed in $O(d^{O(1)}n)$ time, for example using ellipsoid algorithm [15]; the dependence on d can be further improved by reducing the dimension to $O(\log n)$ via Johnson-Lindenstrauss Lemma [22]. Thus the running time for the diameter problem is dominated by the time needed to perform $\tilde{O}(n)$ $(1 + \epsilon)$ -

*University of Southern California. Email: agoel@cs.usc.edu .

†MIT. Email: indyk@theory.lcs.mit.edu . This work was done while the author was at Stanford University.

‡University of Iowa. Email: kvaradar@cs.uiowa.edu .

¹A similar observation w.r.t the Hamming cube was earlier used in [18].

Problem	Ref	Approx.	Time	Comments
Diameter	[10]	$\sqrt{3}$	$O(dn)$	$\tilde{O}(n)$ $(1 + \epsilon)$ -NN queries see Section 3 for some evidence of optimality of $\sqrt{2}$ among $\tilde{O}(dn)$ -time algorithms
	[12]	$1 + \epsilon$	$O(dn \log n + n^2)$	
	[2]	$1 + \epsilon$	$\tilde{O}(n^{2-O(\epsilon^2)} + dn)$	
	[18]	$1 + \epsilon$	$\tilde{O}(n^{1+1/(1+\epsilon/6)} + dn)$	
	here	$1 + \epsilon$	$\tilde{O}(n^{1+1/(1+\epsilon)} + dn)$	
	here	$\sqrt{2}$	$\tilde{O}(dn)$	
Furthest neighbor	here	$\sqrt{2}$	$\tilde{O}(dn)$ preprocessing	and $\tilde{O}(1)$ query time
	here	$(1 + \epsilon)$	$\tilde{O}(dn^{1+1/(1+\epsilon)})$ preprocessing	and $\tilde{O}(dn^{1/(1+\epsilon)})$ query time
Discrete center	here	as for diameter	as for diameter	
Facility location	[23]	3	$\tilde{O}(n^2)$	any metric
	[3]	2.4	$\tilde{O}(n^2)$	any metric
	here	$3(1 + \epsilon)^3$	$\tilde{O}(n^{1+1/(1+\epsilon)})$	$\tilde{O}(n)$ $(1 + \epsilon)$ -NN queries
Min matching	[13]	1	$\tilde{O}(n^{2.5})$	any metric
	[14]	2	$\tilde{O}(n^2)$	any metric
	here	$2(1 + O(\epsilon))$	$\tilde{O}(n^{1+1/(1+\epsilon)})$	$\tilde{O}(n)$ $(1 + \epsilon)$ -NN queries
Bottleneck matching	[11]	1	$\tilde{O}(n^{2.5})$	any metric
	here	$2(1 + \epsilon)$	$\tilde{O}(n^{1+1/(1+\epsilon)})$	$\tilde{O}(n)$ $(1 + \epsilon)$ -NN queries

Figure 1: The old and new results. Due to lack of space, the (large number of) references to algorithms for low-dimensional spaces have been omitted.

nearest neighbor queries. These techniques can in fact be generalized to obtain a data-structure for $(1 + \epsilon)$ approximate furthest-neighbour queries. If we are satisfied with $\sqrt{2}$ -approximation, we can use a subset of the above techniques and get $\tilde{O}(dn)$ -time algorithms for the approximate diameter and *discrete center* problem.

For the *bottleneck matching problem*, we use a close relation between spanning forests with even-sized components and perfect matchings. For the *facility location* we use the primal-dual approximation algorithms of Jain and Vazirani [23]. We show that this primal-dual algorithm can be expressed in terms of $\tilde{O}(n)$ calls to a data structure for $(1 + \epsilon)$ -approximate dynamic bichromatic closest pair; this in turn can be reduced to comparable number of $(1 + \epsilon)$ -approximate dynamic nearest neighbor operations using the scheme of Eppstein [8]². Finally, for the *minimum weight matching problem*, we use a fast implementation of the algorithm of Goemans and Williamson [14]³.

Although in this paper we mainly focus on high-dimensional spaces, we mention that the last two results (i.e. facility location and matching) are also of interest if the dimensionality of the space is low (say constant). In particular, they imply nearly linear-time approximation

algorithms for those problems; the algorithms have small $O()$ constants and are simple to implement.

2 Preliminaries

Notation. We use the following notation. For a metric space (X, d) and $r > 0$ we let $B(p, r)$ denote the ball of radius r centered at p , that is the set of points in X within distance r from p . Let $S(p, r)$ denote the sphere of radius r centered at p , that is the set of points at *exactly* distance r from p . Moreover, for any set S of points we use S^δ to denote the set of points within distance δ from S .

Problems. The problems we address are defined as follows. Let $P = \{p_1, \dots, p_{2n}\}$ be a set of $2n$ points in \mathbb{R}^d . A perfect matching of P is a partition of P into n pairs. The *bottleneck cost* $c(M)$ of a perfect matching M is defined to be the distance between the two points that make up the furthest pair in the matching. The (approximate) *bottleneck matching* problem is to find a perfect matching with (approximately) minimal bottleneck cost.

The *facility location* problem is defined as follows. Let (X, d) be a metric and let $f : X \rightarrow \mathcal{R}^+$. The goal is to find a set S of facilities such that the cost

$$\sum_{p \in S} f(p) + \sum_{q \in X} \min_{p \in S} d(q, p)$$

²The reduction in [8] is approximation preserving [9].

³We omit the description of this algorithm in this version due to lack of space.

is minimized.

The *minimum weight matching* problem takes a metric space (X, d) , with $X = n = 2k$ as input. The goal is to divide the $2k$ points into k disjoint pairs (x_i, y_i) such that $\sum_{i=1}^k d(x_i, y_i)$ is minimized.⁴ The remaining problems are defined as follows.

DEFINITION 1. c -Furthest Neighbor Problem (c -FN) Given a set $P = \{p_1, \dots, p_n\}$ of points in \mathbb{R}^d , devise a data structure which, given any $q \in \mathbb{R}^d$, produces a point $p \in P$ such that $d(q, p) \geq 1/c \max_{p' \in P} d(q, p')$.

The following two problems are trivially reducible to c -FN.

DEFINITION 2. Approximate Discrete Center Problem (c -DCP): given an n -point set $P \subset \mathbb{R}^d$, find $s \in P$ such that

$$\max_{p \in P} d(p, s) \leq c \min_{s \in P} \max_{p \in P} d(p, s).$$

DEFINITION 3. Approximate Diameter Problem (c -DP): given an n -point set $P \subset \mathbb{R}^d$, find $p, q \in P$ such that

$$d(p, q) \geq \frac{1}{c} \max_{p, q \in P} d(p, q).$$

Some of our algorithms exploit efficient algorithms for the following problem.

DEFINITION 4. Approximate Minimum Enclosing Ball (c -MEB): given an n -point set $P \subset \mathbb{R}^d$, find $s \in \mathbb{R}^d$ such that

$$\max_{p \in P} d(p, s) \leq c \min_{s \in \mathbb{R}^d} \max_{p \in P} d(p, s).$$

By using the ellipsoid algorithm, this problem can be solved using $\tilde{O}(d^3 n \log 1/\epsilon)$ arithmetic operations (where $c = 1 + \epsilon$); each operation is performed on words of size $O(d \log 1/\epsilon)$.

Most of our algorithms ultimately rely on efficient data structure for the following problem.

DEFINITION 5. c -Nearest Neighbor Problem (c -NN) Given a set $P = \{p_1, \dots, p_n\}$ in \mathbb{R}^d , devise a data structure which, given any $q \in \mathbb{R}^d$, produces a point $p \in P$ such that $d(q, p) \leq c \min_{p' \in P} d(q, p')$.

In [19] it was shown that c -NN can be reduced to the (defined below) (r, c) -PLEB problem. The (binary-search type) reduction is deterministic and incurs only a *polylog*(n) overhead in the running time and storage requirements.

⁴The reductions for the facility location and both the matching problems work for an arbitrary metric space, but since the nearest neighbour results hold only in \mathbb{R}^d , the running times are claimed only for \mathbb{R}^d .

DEFINITION 6. (r, c) -Point Location in Equal Balls ((r, c) -PLEB) Given n radius- r balls centered at $P = \{p_1, \dots, p_n\}$ in a metric (X, d) , devise a data structure which for any query point $q \in X$ does the following:

- if there exists $p \in P$ with $q \in B(p, r)$ then return YES and a point p' such that $q \in B(p', cr)$,
- if $q \notin B(p, cr)$ for all $p \in P$ then return NO,
- if for the point p closest to q we have $r \leq d(q, p) \leq cr$ then return either YES or NO.

We are also interested in the *dynamic* version of the problem, when we are allowed to add/delete elements of P . Indyk and Motwani [19, 20] gave a data structure for this problem whose query and update time (randomized) is $\tilde{O}(n^{1/c})$. A generalization of the above problem is the (approximate) dynamic *chromatic closest pair* (CCP) problem, where the goal is to maintain (under insertions and deletions) a set of *colored* points P (each point can have one of k colors); at any time, the data structure should keep a pair of points (say p, q) of different colors such that $d(p, q)$ is (approximately) minimum among all pairs of points with different colors. It can be observed that the k -chromatic problem can be reduced to the 2-chromatic (or bichromatic) case with $O(\log k)$ overhead in query and update times. Moreover, Eppstein [8, 9] has shown that the c -approximate bichromatic closest pair can be reduced (with polylogarithmic overhead in query and update time) to the c -approximate dynamic nearest neighbor.

3 A $\sqrt{2}$ -approximation for furthest neighbor problems

In this section we develop a data structure for the c -furthest neighbor search problem (c -FN) by using fast algorithms for c' -MEB, where $c' = 1 + 1/L$ and $c = \sqrt{2} + 1/L$, where $L = n^{\Theta(1)}$, with the exponent being an arbitrarily large constant. The preprocessing time of the algorithm is bounded by the running time of c' -MEB; the query time is $O(d^2 \log n)$. Notice, that this immediately implies $\sqrt{2}$ -approximations for the diameter and discrete center problem in c' -MEB time.

The basic idea of our algorithm is as follows. Let P the n -point subset of \mathbb{R}^d . Suppose that we could in fact compute the exact minimum enclosing ball $B(O^*, r^*)$ of P . Then there is a subset $R \subseteq P$ of at most $d + 2$ points such that each point of R lies on the sphere $S(O^*, r^*)$, and the center O^* is in the convex hull of R (see Figure 2). Our data structure consists simply of the set R . For a query point q that is not O^* , (if q is O^* , any point in R is a furthest neighbor of q) we consider the hyperplane h_q passing through O^* and perpendicular to the line through O^* and q . Since O^* is in the convex

hull of R , there must be at least one point in R on the side of h_q that does not contain q . We return any such point as our approximate furthest neighbor of q . It is easy to see that the answer is a $\sqrt{2}$ -approximation, and that the query can be answered in $O(d^2)$ time.

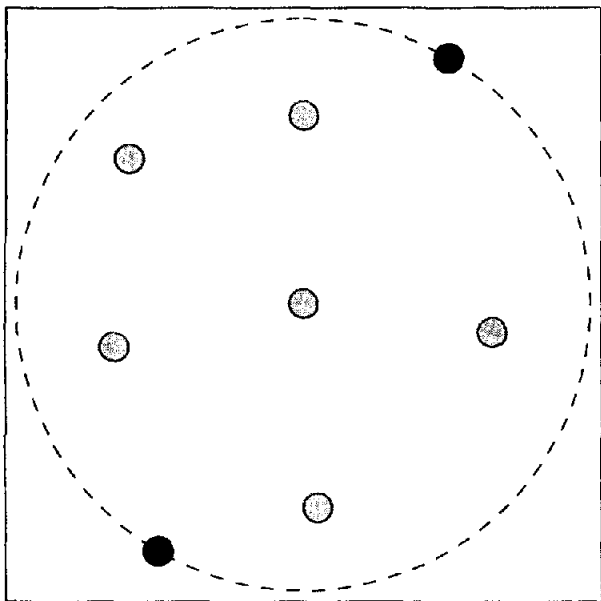


Figure 2: An example set of points and its Minimum Enclosing Ball. The solid points form the set R . One can observe that each hemisphere contains one point from R .

Since we can only afford to compute an approximate minimum enclosing ball $B(O, r)$ for some r sufficiently close to r^* , we modify the algorithm as follows.

- We now choose our subset $R \subseteq P$ of points to be the set of all points of P within $\delta > 0$ of the sphere $S(O, r)$, that is, $R = P \cap S^\delta(O, r)$. We show that this subset satisfies nearly the same properties as in the exact case.
- However, the set R chosen this way may be quite large, in fact of size $\Omega(n)$. So we first apply a small random perturbation to the point set P (before finding the approximate enclosing ball and R) so that with positive probability the set R will have $O(d \log n)$ points.

Perturbation. Assume for simplicity that $[0, 1]^d$ is the bounding box for P . Define a new points set P' by the following random process: for each point $p_i \in P$, choose a random vector v_i such that $|v_i| = \gamma$ and define

$p'_i = p_i + v_i$. Consider now any q from the “ δ -precision cube” $C = [0, 1]^d \cap (\delta\mathcal{Z})^d$ (\mathcal{Z} denotes the set of integers) and $r = |q - p|$ for any $p \in C$ s.t. $r \geq 1/2$. Define $P(q, r) = P \cap S^\delta(q, r)$.

LEMMA 3.1. *There exist constants $c_1, c_2 > 1$ such that if $r - \sqrt{r^2 - \gamma^2} \leq \delta/c_1$ and $\gamma/\delta \geq c_2 n \sqrt{d}$, then there is at least a constant probability that $|P(q, r)| = O(d \log n / \gamma)$ for all q, r as above.*

REMARK 1. *Notice that the first condition is implied by $\gamma^2 \leq \delta r^2 / c_1'$ for some $c_1' \geq 1$. Also, it is possible to satisfy both conditions simultaneously by setting $\delta = \Theta(\frac{1}{dn^2})$ and $\gamma = \Theta(\frac{1}{\sqrt{dn}})$.*

Proof. Take any $p \in P$. Let $S = S(q, r)$ and let $S' = S(p, \gamma)$. Define $A = S^\delta \cap S'$ (see Figure 3 for the illustration). We will first show that $\frac{|A|}{|S'|} \leq 1/n$ where $|A|$ denotes the measure of A (notice that the latter quantity is exactly the probability of $p' \in S^\delta$).

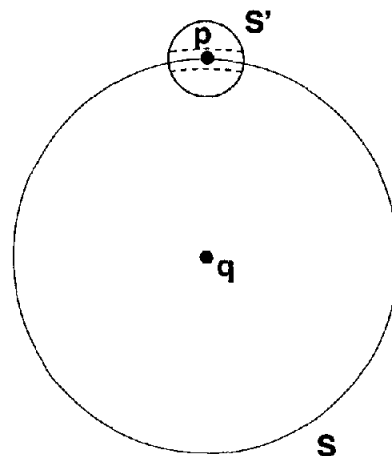


Figure 3: Two spheres: S (with large radius r) and S' (with small radius γ). The region A is bounded using dashed lines.

Let H be a $(d-1)$ -dimensional hyperplane orthogonal to the vector \vec{pq} and tangent to S . It can be verified that (for a suitable c_1) if $r - \sqrt{r^2 - \gamma^2} \leq \delta/c_1$, then $A \subset H^{2\delta}$ (i.e. A is almost “flat”). On the other hand, it is known (e.g., see [21]) that $\frac{|H^{2\delta} \cap S'|}{|S'|} = O(\sqrt{d}\delta/\gamma)$. Therefore, if we choose $\gamma \geq c_2 \delta n \sqrt{d}$ (for a suitable c_2) then the latter quantity becomes smaller than $1/n$.

Now we can upper bound the probability that for fixed pair q and r we have $|P(q, r)| > t$ by

$$\binom{n}{t} / n^t \leq (en/t)^t / n^t = (e/t)^t$$

because each point from P falls into $P(q, r)$ independently with probability $1/n$. Since the number of all

pairs q, r is at most $1/\delta^{2d}$, for $t = cd \log n$ we get that the probability that there exist q, r such that $P(q, r) > t$ is at most $1/\delta^{2d} \cdot \left(\frac{e}{cd \log n}\right)^{cd \log n}$, which is less than $1/2$ for a suitable c .

REMARK 2. Notice that the dependence on n in the above bound can be reduced to $\log n / \log \log n$.

The following “hemisphere lemma” shows that if $B(O, r)$ is an approximate minimum enclosing ball of P , then for any point $q \neq O$, there has to be at least one point in $S^\delta(O, r)$ “close” to the hemisphere of $S(O, r)$ opposite q . Formally, it has to be close to the set $H(q) = \{p \in S(O, r) : (q - O) \cdot (p - O) \leq 0\}$.

LEMMA 3.2. Let $B = B(O, r)$ be a feasible solution for the MEB problem over P such that $r = r^*(1 + \beta)$ where the optimal solution has radius r^* , for $0 \leq \beta \leq 1$. Then for any point $q \in S = S(O, r)$ there exists a point $p \in P$ within distance $a\sqrt{\beta}r$ from $H(q)$, for a constant a .

Proof. We can assume $r^* = 1$. Also, without loss of generality we can assume that $q - O = (1, 0, \dots, 0)$. Let $H = H(q)$. Let $\Delta > \sqrt{\frac{\beta}{1+\beta}}$. By contradiction, assume that $D(P, H) \geq 2\Delta$. Define $O' = (\Delta, 0, \dots, 0)$; we will show that for all points $p \in P$ we have $|p - O'| < \frac{1}{1+\beta}$.

Rotate the space such that the vector $O - O'$ and p lie on the same plane. The point configuration is then depicted as on Figure 4.

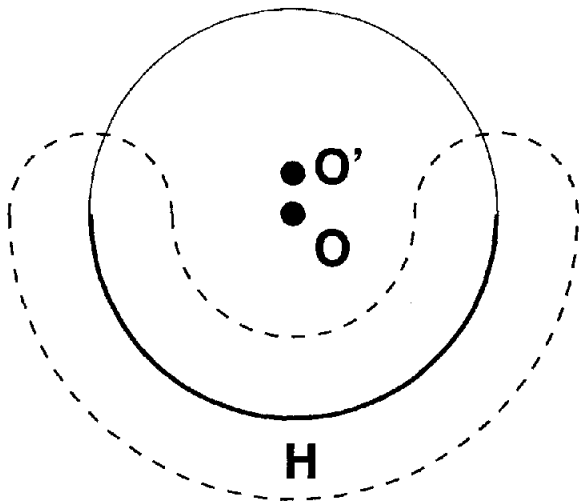


Figure 4: The point configuration after the rotation. All points from P must lie outside of the region surrounded by the dashed line.

It is easy to verify by inspection that in order for a point $p = (x, y)$ to be as far from O' as possible, it must

be that $y \geq \Delta$. However, since $x^2 + y^2 \leq 1$, we conclude that

$$|p - O'| = (y - \Delta)^2 + x^2 \leq 1 - 2y\Delta + \Delta^2 \leq 1 - \Delta^2 < \frac{1}{1 + \beta}$$

which was to be shown.

Final analysis. Our algorithm consists of finding an approximate minimum enclosing ball $B(O, r)$ of P , where $r = r^*(1 + \beta)$ and $\beta = \Theta(\delta^2)$, for $\delta = 1/L$. We then take R to be the set $P \cap S^\delta(O, r)$, the set of points in P lying within δ of the sphere $S(O, r)$. From Lemma 3.1, R has size $O(d \log n)$ with constant probability. The procedure to find R takes $\tilde{O}(d^3 n)$ time. Given any query point q , we simply return the point in R farthest from q as our approximate farthest neighbor.

It follows from Lemmas 3.2 and 3.1 that for any query point q , there is a point $u \in R$ within distance $a\delta r$ from $H(q)$. Let $t = |q - O|$. We know that $|q - u| \geq \sqrt{t^2 + r^2} - a\delta r$, while the furthest point from q (say p) is within distance at most $|q - O| + |p - O| \leq t + r$ from q . Thus

$$\frac{|q - u|}{|q - p|} \geq \frac{\sqrt{t^2 + r^2} - a\delta r}{t + r}.$$

The latter ratio can be easily shown to be at least $(1/\sqrt{2})(1 - 1/n^{\Theta(1)})$, and so u is a $\sqrt{2}(1 + 1/n^{\Theta(1)})$ -furthest neighbor of q .

REMARK 3. We mention that if one is only interested in solving the diameter or discrete center problem (not the furthest neighbor search problem), the algorithms can be considerably simplified. In both cases, we start from finding (approximate) minimum enclosing ball $B = B(O, r)$. For the diameter, we take a point $p \in P$ closest to the boundary of B , find a point $p' \in P$ furthest from p and return (p, p') . For the discrete center, we find a point $p \in P$ closest to O and return it as a center. Since the approximation factor and the analysis remain essentially the same, we omit further description.

A lower bound. It is natural to ask if the approximation factor $\sqrt{2}$ is the best possible, for a furthest neighbor data structure requiring only $O(nT)$ preprocessing and $O(T)$ query time, where $T = \tilde{O}(d^{O(1)})$. Below we show a result which indicates that this indeed could be true. Specifically, we show that if we have a c -approximation (for a constant $c < \sqrt{2}$) for FN, then we could $O(1)$ -approximate nearest neighbor for random point sets from l_2^d (for $d = \Omega(\log n)$) within the same time bounds. The latter (average case) problem has been considered recently in [31]. His algorithm has

$T = n^{\Omega(1)}$; obtaining $T = \tilde{O}(d^{O(1)})$ seems difficult using current techniques.

The reduction works as follows. Let P be a set of points chosen independently and uniformly at random from $[-1, 1]^d$. If $d = \Omega(\log n)$, it is known that for any $p \in P$ the norm $|p|$ is sharply concentrated around the mean (say r). Moreover, for any $p, p' \in P$, the dot product $p \cdot p'$ is close to 0 (i.e. p and p' are “almost” orthogonal), and thus the distance $|p - p'|$ is close to $\sqrt{2}r$. Consider the query point q . We can focus on the case when there exists $p \in P$ such that $|p - q| \leq r/A$, for some large but constant A (otherwise any point from P is a $O(1)$ -approximate nearest neighbor of q). Now we ask a c -FN query with the argument $-q$. Observe that:

- the distance from $-q$ to p is close to $2r$
- the distance from $-q$ to $p' \in P - \{p\}$ is close to $\sqrt{2}r$

Therefore for A large enough the c -FN algorithm will return p .

4 A $(1 + \epsilon)$ -approximation for FN

In this section we show how to answer $(1 + \epsilon)$ -FN queries by performing $\tilde{O}(1)$ $(1 + \epsilon)$ -approximate nearest neighbor queries; the same overhead holds for the preprocessing.

The reduction is as follows. At the beginning, similarly as in the previous section, we define an approximate minimum enclosing ball $B = B(O, r)$ of the given set of points P . Since we can get arbitrarily good approximation at essentially no cost, in the following we will assume for simplicity that B is *exact*; this incurs only a multiplicative error of $1 + 1/n^{\Theta(1)}$, as in the previous section.

Let $\alpha > 0$ (specified later). Let $r_i = r/(1 + \alpha)^i$, let $k = O(1/\alpha)$ be the first integer such that $r_k \leq \sqrt{2} - 1$ and let $B = B(O, r_i)$ for $i = 0 \dots k$ be a sequence of concentric balls around O . We round each point in P to the nearest sphere $S_i = S(O, r_i)$ (notice that in this way we make only αr additive error). For each ball B_i we build an $(1 + \epsilon)$ -approximate nearest neighbor data structure for the set of points on S_i .

Now, for each point $p \in P$ and each ball B_i , we define the “antipode” p^i of p w.r.t. B_i defined as follows. Let p_1 and p_2 be the two points of the intersection of the sphere of B_i with the line passing through p and O . Let h_p denote the hyperplane through O that is perpendicular to the line through p and O . We define p^i to be the one of the points p_1, p_2 which lies on the side of h_p different from the side containing p .

Now, in order to find the furthest neighbor of q , we issue a $(1 + \epsilon)$ -approximate nearest neighbor query with

the point q^i in the data structure for the points on S_i , for all i . Among the points found, we return the one furthest from q .

The complexity of the algorithm is clear from the description. In the following we will show that (modulo the rounding phase) the algorithm indeed computes a $(1 + \epsilon)$ -FN. To this end, consider the actual furthest neighbor p of q . We can assume that $p \in S_i$ and $q \in S_j$. Let $H(q)$ denote the hemisphere of B (or B_j) “opposite” q , that is $H(q) = \{t \in S | (t - O) \cdot (q - O) \leq 0\}$. By an argument as in the previous section we have that point p has to lie in $H(q)$.

Let s be the point returned as the $(1 + \epsilon)$ -approximate nearest neighbor of q^i in the data structure for $P \cap S_i$. From the definition it has to satisfy the property $\frac{|s - q^i|}{|p - q^i|} - 1 \leq \epsilon$. We will show that

$$L = \frac{\frac{|q - p|}{|q - s|} - 1}{\frac{|s - q^i|}{|p - q^i|} - 1} \leq 1$$

which implies $\frac{|q - p|}{|q - s|} \leq 1 + \epsilon$. To this end, observe that we can rotate and scale the space such that O is the origin and the ball B_i has radius 1. Moreover, we can assume that all points q, q^i, p, s lie in a 3-dimensional space. In fact, we can move s to the plane spanned by q, q^i, p such that all distances of interest remain unchanged. Therefore, we can assume that all four points lie on the plane. Let $q^i = 1 = (0, 1)$, $q = (0, -y)$, $y \geq 0$ and $p, s \in S(0, 1)$ lie in the upper left quadrant, so that $p = (\cos \theta, \sin \theta)$ and $s = (\cos \theta', \sin \theta')$, such that $\theta, \theta' \in [\pi/2, \pi]$ and $\theta < \theta'$. We now want to maximize

$$\frac{\frac{|q - p|}{|q - s|} - 1}{\frac{|s - 1|}{|p - 1|} - 1}$$

We will do that in two phases. Firstly, we show that the above expression is maximized for $q = -1 = (0, -1)$. Then we bound

$$L' = \frac{\frac{|-1 - p|}{|-1 - s|} - 1}{\frac{|s - q^i|}{|p - q^i|} - 1}$$

LEMMA 4.1. *For any fixed p and s , the value of $L = L(q)$ is maximized when $q = -1$.*

Proof. Notice that the denominator of L does not depend on q , therefore it is sufficient to maximize $F(y) = \frac{|q - p|}{|q - s|}$, or equivalently, $F(y)^2 - 1$. Then

$$\begin{aligned} F(y)^2 - 1 &= \frac{\cos^2 \theta + (\sin^2 \theta + y)^2}{\cos^2 \theta' + (\sin^2 \theta' + y)^2} - 1 \\ &= \frac{1 + y^2 + 2y \sin \theta}{1 + y^2 + 2y \sin \theta'} - 1 \end{aligned}$$

$$= \frac{2y(\sin\theta - \sin\theta')}{1 + y^2 + 2y\sin\theta'}$$

The latter expression can be easily seen to achieve maximum for $y = 1$, independently of the value of θ and θ' , as long as $\sin\theta - \sin\theta' \geq 0$ and $\sin\theta' \geq 0$.

LEMMA 4.2. $L' \leq 1$.

Proof. We can expand L' into

$$\left(\frac{\sqrt{1 + \sin\theta}}{\sqrt{1 + \sin\theta'}} - 1 \right) / \left(\frac{\sqrt{1 - \sin\theta'}}{\sqrt{1 - \sin\theta}} - 1 \right)$$

Let $\phi = \theta/2 - \pi/4$ and $\phi' = \theta'/2 - \pi/4$; note that $\phi, \phi' \in [\pi/4, \pi/2]$. We use the identities $1 + \sin\theta = 2\cos^2\phi$ and $1 - \sin\theta = 2\sin^2\phi$. In this case, the expression becomes

$$L' = \frac{\frac{\cos\phi}{\cos\phi'} - 1}{\frac{\sin\phi'}{\sin\phi} - 1}$$

In order to show $L' \leq 1$ it is sufficient to show $\cos\phi \sin\phi - \sin\phi' \cos\phi' \leq 0$. However, the LHS of this expression is just $(\sin 2\phi - \sin 2\phi')/2$, which is smaller than 0 since $\phi \leq \phi'$.

Thus, the above procedure returns a $(1 + \epsilon + O(\alpha))$ -approximate furthest neighbor.

THEOREM 4.1. *The static $(c + \delta)$ -approximate furthest neighbor can be solved with the preprocessing time $\tilde{O}(d^{O(1)}n)$ plus the cost of initiating $O(1/\delta)$ data structures for static c -PLEB; the query time is bounded by the cost of $O(1/\delta)$ c -PLEB queries.*

5 Bottleneck matching

Let M^* denote the perfect matching of P with the smallest bottleneck cost, and let c^* be the cost of M^* . In this section, we describe an algorithm that computes a perfect matching whose cost is at most $2(1 + \epsilon)c^*$.

For any real $r \geq 0$, let $G(r)$ denote the graph whose vertex set is P and whose edge set is $\{(p_i, p_j) | p_i \neq p_j, D(p_i, p_j) \leq r\}$. Let r^* be the smallest value of r for which each connected component of $G(r)$ has an even number of vertices. Let G_1, \dots, G_k be the connected components of $G(r^*)$, and for $1 \leq i \leq k$, let T_i be any spanning tree of G_i . Note that the length of any edge in T_i is at most r^* .

LEMMA 5.1. *The cost c^* of the optimal matching is at least r^* .*

Proof. Fix some $r < r^*$. We will show that for any perfect matching M of P , the cost $c(M)$ of M is greater than r .

By definition, there is a connected component G' of $G(r)$ which has an odd number of vertices. Thus for any perfect matching M , there is a pair $(u, v) \in M$ such that u is a vertex of G' and v is not. Clearly $D(u, v) > r$, and therefore $c(M) \geq D(u, v) > r$.

On the other hand, we can get a perfect matching of P whose cost is at most $2r^*$ by applying the algorithm of the following lemma to each T_i .

LEMMA 5.2. *Let T be a tree whose vertex set $V = \{p_1, \dots, p_{2m}\}$ has even cardinality, and let ℓ denote the length of the longest edge in T . We can construct a perfect matching of V whose bottleneck cost is at most 2ℓ . Given T , such a matching can be computed in $O(m)$ time.*

Proof. We make T into a rooted tree by picking an arbitrary vertex as the root. Let u be any internal vertex of T such that all of u 's children are leaves of the tree. It is easy to see that such a vertex exists.

1. If u has two or more children, let v and w be any two children of u . Delete v (resp. w) and the edge (v, u) (resp. (w, u)) incident to it. Let T' be the resulting tree. Recursively compute a perfect matching M' of the vertices in T' . Return $M' \cup \{(v, w)\}$ as the perfect matching for the vertices of T . Note that $D(v, w) \leq D(v, u) + D(u, v) \leq 2\ell$.
2. If u has only one child v , delete v and u from T . In the resulting tree T' , recursively compute a perfect matching M' of its vertices. Return $M' \cup \{(u, v)\}$ as the perfect matching for the vertices of T . Clearly, $D(u, v) \leq \ell$.

It is easy to see that the bottleneck cost of the matching M returned by the above procedure is at most 2ℓ . We now sketch an $O(m)$ time implementation of the procedure. Let us call an internal vertex of a rooted tree T *interesting* if all its children are leaves. Clearly, it suffices to maintain the set of interesting vertices of T as leaves get deleted from T . At each internal vertex u of the tree, we keep a count of all the children of u that are not leaves. These counts can be computed initially in $O(m)$ time by a post-order traversal of T . The interesting vertices are those whose count is zero. When a leaf gets deleted, the count changes at only a constant number of vertices near the leaf. Hence, updating the counts takes $O(1)$ time. Thus the entire procedure can be implemented in $O(m)$ time.

Reduction to nearest-neighbor. We now compute a spanning forest $\{T_1, \dots, T_k\}$ of P with the property that each edge of the forest has length at most

$r^*(1 + \epsilon)$, and each tree in the forest has an even number of vertices. Such a forest can be easily computed by running Kruskal's MST algorithm until each component is even, with $O(n \log n)$ calls to K -chromatic dynamic closest pair. Only, we replace the calls to the exact data structure by calls to a data structure for $(1 + \epsilon)$ -approximate K -chromatic dynamic closest pair. Using the algorithm of the above lemma, we can then compute a matching whose bottleneck cost is $2r^*(1 + \epsilon)$.

THEOREM 5.1. *The $2(1 + \epsilon)$ -approximate bottleneck matching can be solved using $O(n \log n)$ calls to $(1 + \epsilon)$ -CCP.*

6 Facility location

In this section we show how to implement the algorithm of Jain and Vazirani [23] as a sequence of closest pair operations⁵. The original algorithm gives factor 3 approximation to this problem and runs in $O(n^2 \log n)$ time, where n is the number of vertices⁶. Although there are algorithms which achieve lower approximation factor, their result is interesting because of its low time complexity.

We first reduce the bit precision needed to represent the distances to $O(\log n)$, in the following way. Firstly, we compute a lower bound L for the optimal facility location cost C^* , such that $L \leq C^* \leq aLn$, for a constant a . To this end, observe that the cost C' of a minimum spanning forest with k components is a lower bound for C^* ; it is also easy to see $C^* \leq nC'$. We compute an a -approximate spanning forest using the approximate MST techniques; if we set $a > c = 1 + \epsilon$ then this computation can be done in time negligible compared with the running time of the rest of this procedure. We can then set $L = C'/a$.

After that, we collapse all pairs of vertices whose distance is less than $\delta \frac{L}{n}$ for $\delta \approx \epsilon$, forming disjoint clusters; clearly, this operation changes the value of C^* by at most $\pm \delta C^*$. This clustering can be achieved using $\tilde{O}(n)$ calls to $O(1)$ -PLEB, along with a simple Union-Find data structure. We also open all facilities which have cost at most $\delta L/n$. Again, this can increase the cost of the solution by at most δL . Thus, as a result of the above transformation, the cost of the solution can go up by at most a factor of $1 + O(\delta)$. Finally, we set the value of all edges with distance greater than aLn to aLn ; note that the resulting distance matrix is still a metric. We do the same for the facilities which cost

⁵We give here only a brief description of their algorithm and its proof of correctness; the reader is advised to read this section after or concurrently to reading of their paper.

⁶In fact their running time is $O(n_c n_f \log(n_c n_f))$, where n_c and n_f are the number of cities and facilities, respectively.

more aLn . At the of this procedure we can represent each distance using only $O(\log n)$ bits of precision.

Now we proceed with the main algorithm of Jain and Vazirani. The algorithm is based on the primal-dual approach (see [14] for the background information on this method). The basic idea is to formulate the problem as a 0-1 integer program, relax it to a linear program, and apply primal-dual ascent to obtaining approximate solution. The 0-1 programming formulation of facility location problem is as follows. We use two kinds of variables: x_{ij} and y_i . The event $y_i = 1$ is interpreted as that the facility i is open; the event $x_{ij} = 1$ is interpreted as that the j th point (city) is using the i th facility. With this in mind, one can formulate the problem as follows.

$$\begin{aligned} &\text{minimize} && \sum_{i \in P_f, j \in P_c} D(i, j)x_{ij} + \sum_{i \in P_f} f(i)y_i \\ &\text{such that} && \\ &&& \forall j \in P_c : \sum_{i \in P_f} x_{ij} \geq 1 \\ &&& \forall i \in P_f, \forall j \in P_c : y_i - x_{ij} \geq 0 \\ &&& \forall i \in P_f, \forall j \in P_c : x_{ij} \in \{0, 1\} \\ &&& \forall i \in P_f : y_i \in \{0, 1\} \end{aligned}$$

After the relaxation, the program becomes:

$$\begin{aligned} &\text{minimize} && \sum_{i \in P_f, j \in P_c} D(i, j)x_{ij} + \sum_{i \in P_f} f(i)y_i \\ &\text{such that} && \\ &&& \forall j \in P_c : \sum_{i \in P_f} x_{ij} \geq 1 \\ &&& \forall i \in P_f, \forall j \in P_c : y_i - x_{ij} \geq 0 \\ &&& \forall i \in P_f, \forall j \in P_c : x_{ij} \geq 0 \\ &&& \forall i \in P_f : y_i \geq 0 \end{aligned}$$

The primal-dual method operates mainly on the dual formulation of the linear program, which is as follows.

$$\begin{aligned} &\text{maximize} && \sum_{j \in P_c} \alpha_j \\ &&& \forall i \in P_f, \forall j \in P_c : \alpha_j \leq \beta_{i,j} + D(i, j) \\ &&& \forall i \in P_f : \sum_j \beta_{i,j} \leq f(i) \\ &&& \forall i \in P_f, \forall j \in P_c : \beta_{i,j} \geq 0 \\ &&& \forall j \in P_c : \alpha_j \geq 0 \end{aligned}$$

The algorithm tries to maximize the dual objective while maintaining dual feasibility. It is done by growing

balls around each active city (initially all cities are active) at a uniform rate; α_j represents the radius of the ball around j . There are two types of events. The first type of event occurs when a ball hits a facility (i.e. $\alpha_j = D(i, j)$ for some pair (i, j)); to maintain dual feasibility, $\beta_{i,j}$ must also be increased now as α_j grows. The second type of event occurs when $\sum_j \beta_{k,j} = f(k)$ for some facility k . Now none of the $\beta_{k,j}$'s can increase, and therefore we have to stop growing all α_j such that $\alpha_j \geq D(k, j)$ - the corresponding cities become inactive and the corresponding edges are declared *tight*. The facility which causes an event of the second type is marked as being temporarily open. If an event of the first type results in a ball hitting a temporarily open facility then the corresponding edge is declared tight and the ball stops growing.

The number of events of the first type can be as large as $\Omega(n^2)$ and hence it might seem impossible to obtain a sublinear (in the size of the metric space) implementation of the algorithm of Jain and Vazirani. We get around this problem by dividing events of the first type into two categories - essential and non-essential. Events which result in a ball hitting an already open facility are declared essential - there are only $O(n)$ of these events. The remaining events are used merely to help in detecting events of the second type and are not used in the subsequent phase. We call these events inessential.

We now have to explain how we detect events of the second type and essential events of the first type. Notice that all distances are larger than $\delta L/n$ (by our preprocessing phase) and hence no events can happen till all the balls grow to a radius of $r_0 = L$. During each ball growing step, the radius of each active ball will be increased to the next discrete value.

Define a PLEB_k to be a c -approximate PLEB data structure used to locate points within balls of radius $r_0 \cdot c^k$. The essential events of the first type are easy to detect - after the k -th ball growing step, add all the temporarily open facilities to a PLEB_k structure. Then, for each active city, add the city to the data structure, see if there is a facility that is close enough, and then delete the city from the data structure. If a close enough facility was found, then this city will be marked inactive and the corresponding edge declared tight.

Detecting events of the second type is somewhat harder. Let X_k denote a PLEB_k data structure defined on all cities which were active during the k -th ball growing step. For facility i , let $N_k(i)$ denote the number of cities in X_k which are within a distance r_k from facility i .

LEMMA 6.1. For any unopened facility i , $\sum_j \beta_{ij} =$

$\sum_{k=1}^K (r_k - r_{k-1}) N_k(i)$ after K ball growing steps.

Proof. Let $\gamma_{i,j,k} = 1$ iff each of the following is true

1. City j is active after the k -th ball growing step
2. Facility i is unopened after the k -th ball growing step
3. $D(i, j) < r_k$

and $\gamma_{i,j,k} = 0$ otherwise. Now, after K ball growing steps, for any unopened facility i ,

$$\begin{aligned} \sum_j \beta_{ij} &= \sum_j \max\{\alpha_j - D(i, j), 0\} \\ &= \sum_j \sum_{k=1}^K (r_k - r_{k-1}) \gamma_{ijk} \\ &= \sum_{k=1}^K (r_k - r_{k-1}) \sum_j \gamma_{ijk} \\ &= \sum_{k=1}^K (r_k - r_{k-1}) N_k(i) \end{aligned}$$

Since k takes $\tilde{O}(1)$ values, we now try to estimate $N_k(i)$ in time polynomially less than n . To this end notice that $N_k(i)$ is just the number of cities within distance r_k from i . This can be estimated using standard approximate counting techniques (by reduction to approximate PLEB), see [20] for the details. Thus the total running time of phase 1 is bounded by $\tilde{O}(n)$ c -PLEB operations.

The second phase of the algorithm by Jain and Vazirani is relatively straightforward. Let F' denote the set of temporary open facilities, T denote a graph defined by set of tight edges and let H be the square of T . The facilities in F' are ordered in the increasing order of the time they have been declared open; let $p_1 \dots p_t$ denote this sequence. Then the algorithm builds the final set of facilities, by doing the following step for $i = 1 \dots t$: if no neighbor of p_i in H has been yet added to F , p_i is added to F ; otherwise, if such a neighbor exists, p_i is *not added* to F . We can again implement it efficiently using the c -PLEB data structure.

It is not difficult to see that the algorithm performs at most $\tilde{O}(n)$ calls to an c -PLEB data structure, and thus its running time is bounded accordingly. It remains to bound the approximation ratio, which we defer to the final version of this paper.

THEOREM 6.1. Our implementation of the algorithm of Jain and Vazirani performs $\tilde{O}(n)$ calls to c -PLEB and produces a $3(1 + \delta)c^3$ -approximate solution, for any constant $\delta > 0$.

Acknowledgment: We would like to thank C. Sinan Güntürk for the analytic proof of Lemma 4.1.

References

- [1] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, and A. Wu, "An optimal algorithm for approximate nearest neighbor searching", *SODA'94*, pp. 573-582.
- [2] A. Borodin, R. Ostrovsky and Y. Rabani "Sub-quadratic Approximation Algorithms For clustering Problems in High Dimensional Spaces", *STOC'99*.
- [3] M. Charikar, S. Guha, "Improved combinatorial algorithms for facility location and k -median problems", *FOCS'99*.
- [4] K. Clarkson. "A randomized algorithm for closest-point queries", *SIAM Journal on Computing*, 17(1988), pp. 830-847.
- [5] E. Cohen, D. Lewis, "Approximating matrix multiplication for pattern recognition tasks", *SODA'97*, pp. 682-691.
- [6] F. Chudak. Improved approximation algorithms for uncapacitated facility location. *Integer Programming and Combinatorial Optimization*, pages 180-194, 1998.
- [7] F. Chudak and D.B. Shmoys. Improved approximation algorithms for the capacitated facility location problem. *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 875-876, 1999.
- [8] D. Eppstein, "Dynamic Euclidean minimum spanning trees and extrema of binary functions", *Disc. Comp. Geom.* 13, pp. 111-122, 1995.
- [9] D. Eppstein, personal communication, 1999.
- [10] O. Egecioglu, B. Kalantari, "Approximating the diameter of a set of points in the Euclidean space", *IPL* (32) 4, pp. 205, 1989.
- [11] S. Even, O. Kariv, "An $O(n^{2.5})$ algorithm for maximum matching in general graphs", *FOCS'75*, pp. 100-112.
- [12] D. Finocchiaro, M. Pellegrini, "On computing the diameter of a point set in high dimensional Euclidean space", *ESA'99*.
- [13] H. Gabow, R. Tarjan, "Faster scaling algorithms for general graph matching problems".
- [14] M.X. Goemans and D.P. Williamson, "The Primal-Dual Method for Approximation Algorithms and its Application to Network Design Problems", in *Approximation Algorithms*, D. Hochbaum, Ed., 1997.
- [15] M. Grotschel, L. Lovasz, and A. Schrijver, "Geometric algorithms and combinatorial optimization", Springer-Verlag, 1987.
- [16] S. Guha and S. Khuller. Greedy strikes back: Improved facility location algorithms. *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 649-657, 1998.
- [17] P. Indyk, "On Approximate Nearest Neighbors in Non-Euclidean Spaces", *FOCS'98*, pp. 148-155.
- [18] P. Indyk, "Dimensionality Reduction Techniques for Proximity Problems", accepted to the 11th Symposium on Discrete Algorithms, 2000.
- [19] P. Indyk, R. Motwani, "Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality", *STOC'98*, pp. 604-613.
- [20] P. Indyk, "High-dimensional Computational Geometry", Ph.D. thesis, Stanford, 2000.
- [21] P. Indyk, R. Motwani, P. Raghavan, S. Vempala, "Locality-preserving hashing in multidimensional spaces", *STOC'97*, pp. 618-625.
- [22] W.B. Johnson and J. Lindenstrauss, "Extensions of Lipschitz mapping into Hilbert space", *Contemporary Mathematics*, 26(1984), pp. 189-206.
- [23] K. Jain, V. V. Vazirani, "Primal-Dual Approximation Algorithms for Metric Facility Location and k -Median Problems", *FOCS'99*.
- [24] J. Kleinberg, "Two Algorithms for Nearest Neighbor Search in High Dimensions", *STOC'97*, pp. 599-608.
- [25] E. Kushilevitz, R. Ostrovsky, and Y. Rabani, "Efficient search for approximate nearest neighbor in high dimensional spaces", *STOC'98*, pp. 614-623.
- [26] S. Meiser. "Point location in arrangements of hyperplanes", *Information and Computation*, 106(1993):286-303.
- [27] K. Mulmuley, "Computational geometry", Prentice Hall, 1993.
- [28] F. Preparata, I. Shamos, "Computational geometry", 1985.
- [29] I. Shamos, J. Bentley, "Divide-and-Conquer in Multi-dimensional Space", *STOC'76*, pp. 220-230.
- [30] D.B. Shmoys, Tardos.E., and K. Aardal. Approximation algorithms for facility location problems. *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 265-274, 1997.
- [31] P. Yianilios, "Locally Lifting the Curse of Dimensionality for Nearest Neighbor Search", *SODA'00*.