# Reductions Among High Dimensional Proximity Problems

Ashish Goel[*] Piotr Indyk[†] Kasturi Varadarajan[‡]

April 28, 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 Line Center, Approximate Metric Facility Location, Approximate Bottleneck Matching, and Approximate Minimum Weight Matching.

[*]University of Southern California. Email: agoel@cs.usc.edu .
[†]Stanford University. Email: indyk@cs.stanford.edu .
[‡]University of Iowa. Email: kvaradar@cs.uiowa.edu .

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

## 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; 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 $(1+\varepsilon)$ 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 $\tilde{O}(n)$ $(1+\epsilon)$-nearest neighbor queries. These techniques can in fact be generalized to obtain a data-structure for $(1+\varepsilon)$ 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. The algorithm for the approximate *line center* problem computes the (approximate) diameter pair (say $p$ and $q$) and returns the line passing through $p, q$; this results in roughly 2-approximate solution.

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 give a fast implementation of the algorithm of Goemans and Williamson [14]; our main tool is a new data structure for multichromatic nearest neighbors.

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.

**The paper.** The rest of the paper is organized as follows. In Section 2 we introduce the basic notation and the formal statements of the problems. In Section 3 we describe a fast $\sqrt{2}$-approximate furthest neighbor data structure, and in Section 4 we describe our $(1+\epsilon)$-approximate diameter algorithm. The techniques in Section 4 in fact yield a $(1+\epsilon)$-approximate furthest neighbour data structure. Note that a data structure for $c$-approximate furthest neighbours immediately yields $c$-approximate algorithms for the diameter and discrete center problems. Section 5 is devoted to bottleneck matching, Section 6 to the facility location problem, and Section 7 to minimum weight matching. We omit the 2-approximation proof for the line center problem in this version of the paper.

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

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

## 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 $S^\delta$ to denote the set of points within distance $\delta$ from $S$.

**Problems.** The problems we address are defined as follows. Let $P = \{p_1, \ldots, p_{2n}\}$ be a set of $2n$ points in $\Re^d$. 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 $f : X \to \mathcal{R}^+$. The goal is to find a set $S$ of facilities such that the cost

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

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.[3] The remaining problems are defined as follows.

**Definition 1 ($c$-Furthest Neighbor Search Problem ($c$-FNS))** *Given a set $P = \{p_1, \ldots, p_n\}$ of points in $\Re^d$, devise a data structure which, given any $q \in \Re^d$, 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 $c$-FNS.

**Definition 2 Approximate Discrete Center Problem ($c$-DCP):** *given an $n$-point set $P \subset \Re^d$, 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 $P \subset \Re^d$, 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 Continous Center Problem ($c$-CCP):** *given an $n$-point set $P \subset \Re^d$, find $s \in \Re^d$ such that*

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

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 $O(d \log 1/\epsilon)$.

The *line center* problem is similar to the Continous Center Problem, but the center $s$ is a *line* instead of a point.

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

[3]The reductions for the facility location and both the matching problems work for an arbitarary metric space, but since the nearest neigbour results hold only in $\Re^d$, the running times are claimed only for $\Re^d$.

**Definition 5 ($c$-Nearest Neighbor Search Problem ($c$-NNS))** *Given a set $P = \{p_1, \ldots, p_n\}$ in $\Re^d$, devise a data structure which, given any $q \in \Re^d$, produces a point $p \in P$ such that $d(q,p) \le c \min_{p' \in P} d(q,p')$.*

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 $k$-chromatic closest pair* ($k$-DCCP) 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 queryand 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 An approximation for Furthest Neighbor Problems

In this section we develop a data structure for the $c$-furthest neighbor search problem ($c$-FNS), where $c = \sqrt{2} + 1/n^{O(1)}$, by using fast algorithms for $c'$-CCP. The preprocessing time of the algorithm is bounded by the running time of $c'$-CCP; 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 $c'$-CCP time.

The basic idea of our algorithm is as follows. Let $P$ be the $n$-point subset of $\Re^d$. 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 $R$. 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 neighbour 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 neighbour 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.

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 $\Omega(n)$. So we first apply a small random perturbation to the point set $P$ (before finding the approximate enclosing ball and the set $R$) 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 minimum 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 $p'_i = p_i + v_i$. We define the "$\delta$-precision cube" $C = [0,1]^d \cap (\delta \mathcal{Z})^d$($\mathcal{Z}$ denotes the set of integers). Let $q \in C$, and $r$ a real number such that $r = |q - q'|$ for some $q' \in C$ s.t. $r \ge 1/2$. Define $P(q,r) = P' \cap S^\delta(q,r)$.

**Lemma 1** *There exist constants $c_1, c_2 > 1$ such that if $r - \sqrt{r^2 - \gamma^2} \le \delta / c_1$ and $\gamma/\delta \ge c_2 n \sqrt{d}$, 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 coresponds to $\gamma^2 \leq \delta r / c_1'$ and it is possible to satisfy both conditions simultanously.*

**Proof:** Take any $p \in P$. Let $S = S(q, r)$ and let $S' = S(p, \gamma)$. Define $A = S^\delta \cap S'$. 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 $p' \in S^\delta$).

Let $H$ be a $d-1$-dimensional hyperplane orthogonal to the vector $\overrightarrow{pq}$ and tangent to $S$. It can be verified that (for a suitable $c_1$) if $r - \sqrt{r^2 - \gamma^2} \leq c_1 \delta$, then $A \subset H^{2\delta}$ (i.e. $A$ is almost "flat"). On the other hand, it can be easily shown (cf. [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

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

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 $P(q, r) \geq t$ 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 2** *Let $B = B(O, r)$ be a solution to CCP problem over $P$ such that $r = r^*(1 + \beta)$ where the optimal solution has radius $r^*$. Then for any point $q \in S = S(O, r)$ there exists a point $p \in P$ within distance $O(\sqrt{\beta} r)$ from $H(q)$.*

**Proof:** For simplicity, we prove the theorem for $\beta = 0$. We can assume $r^* = 1$. By contradiction, assume that the statement in the theorem is false. In this case for each $p$ we have either $|p - O| < r^*$ or $(p - O) \cdot (q - O) > 0$. Without loss of generality we can assume that $q - O = (1, 0, \ldots 0)$, in this case the latter condition becomes $p_{|1} > O_{|1}$. It is easy to see that this implies existence of $\epsilon > 0$ such that for each $p \in P$ we have $d(O + (\epsilon, 0, \ldots 0), p) < r^*$, contradicting the assumption that $r^*$ is optimal. □

**Final analysis.** Our algorithm first applies the perturbation to $P$. Abusing notation, we denote by $P$ the perturbed set. We then find an approximate minimum enclosing ball $B(O, r)$ of $P$, where $r = r^*(1 + \delta/\sqrt{d})$, and $\delta = 1/n^{\Theta(1)}$. We "round" $O$ to the nearest point on the $\delta$-precision cube $C$, and after increasing $r$ by $2\delta$, we "round" it so that it is a multiple of $\delta$. Note that $r$ now corresponds to the distance between two points on the grid $C$, $B(O, r)$ is still an enclosing ball for $P$, and $r \leq r^*(1 + 4\delta)$. (The last inequality uses the fact that by our bounding box assumption $r^* \geq 1/2$.) We then take $R$ to be the set $P \cap S^\delta(O, r)$, the set of points in $P$ lying within $\delta$ of the sphere $S(O, r)$. From Lemma 1, $R$ has size $O(d \log n)$ with constant probability. (The probability can be amplified by repetitions.) 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 neighbour.

It follows from Lemma 2 that for any query point $q$, there is a point $u \in R$ within distance $r/L$ from $H(q)$, where $1/L = \sqrt{4\delta}$. Let $t = |q - O|$. We know that $|q-u| \geq \sqrt{t^2 + r^2} - r/L$, while the furthest point from $q$ (say $p$) is within distance at most $|q - O| + |p - O| \leq t + r$ from $q$. Thus

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

Since $1/L = \sqrt{4\delta}$, the latter ratio an 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 neighbour 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. Specifially, we show that if we have a $c$-approximation (for a constant $c < \sqrt{2}$) for FNS, 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 $c$-FNS 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 $c$-FNS algorithm will return $p$.

## 4 $(1+\epsilon)$-approximation for Diameter and FNS

In this section we show how to answer $(1+\epsilon)$-FNS queries by performing $\tilde{O}(1)$ $(1+\epsilon)$-NNS queries; the preprocessing time is $\tilde{O}(dn^{1+1/(1+\epsilon)})$. This in particular gives a bound of $\tilde{O}(dn^{1+1/(1+\epsilon)})$ for the $(1+\varepsilon)$-diameter problem. For simplicity, we describe the method as applied directly to the diameter problem.

The reduction is as follows. At the beginning, similarily as in the previous section, we compute 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.*)

Let $\alpha > 0$ (specified later). Let $r_i = r/(1+\alpha)^i$, let $k = O(1/\alpha)$ be the first integer such that $r_k \leq (\sqrt{2}-1)r$ and let $B = B(O, r_i)$ for $i = 0 \ldots k$ be a sequence of concentric balls around $O$. We throw away the points in the interior of the ball $B(O, r_k)$; since the diameter is at least $\sqrt{2}r$, the discarded points don't affect the diameter. We round each remaining point in $P$ to the nearest sphere $S_i = S(O, r_i)$ (Notice that this introduces an additive error of at most $\alpha r$ in the diameter). For each ball $B_i$ we build an $(1+\epsilon)$-approximate nearest neighbor data structure of [20] for the set of points on $S_i$. This completes the preprocessing phase, which takes $\tilde{O}(dn^{1+1/(1+\epsilon)})$ time.

In the query phase, for each point $p \in P$ and each sphere $S_i$, we compute the "antipode" $p'$ of $p$ w.r.t. $S_i$ defined as follows. Let $p_1$ and $p_2$ be the two points of the intersection of the sphere $S_i$ 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 $p'$ to be the one of the points $p_1, p_2$ which lies on the side of $h_p$ different from the side containing $p$. Then we issue a $(1+\epsilon)$-approximate nearest neighbor query with the point $p'$ in the data structure for the points on $S_i$. If the query returned the point $q$, we take $(p, q)$ as a candidate diametric pair. After collecting all $\tilde{O}(n)$ pairs obtained in this way we return the furthest one.

The complexity of the algorithm is clear from the description. In the following we will show that the algorithm indeed computes a $(1+\epsilon)$-approximate diameter for the rounded set. (We will take care of the $\alpha r$ rounding error later). To this end, consider the actual diameter pair $p, q$. We can assume that $p \in S_i$ and $q \in S_j$, $i \geq j$, so $p \in B = B_j$ (otherwise we swap $p$ and $q$). Let $H(p)$ denote the hemisphere of $B_j$ "opposite" $p$, that is $H(p) = \{t \in S_j | (t-O).(q-O) \leq 0\}$. The point $q$ has to lie in $H(p)$, since otherwise $|p-q| < \sqrt{2}r$ while we know the diameter is at least $\sqrt{2}r$.

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

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

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

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

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

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

**Lemma 3** *For any fixed $q$ and $s$, the value of $L = L(p)$ is maximised when $p = -1$.*

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

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

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

**Lemma 4** $L' \leq 1$.

**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 is smaller than 0 since $\phi \leq \phi'$. $\Box$

Thus, the above procedure returns a $(1 + \epsilon + c\alpha)$-approximate diameter for some $c > 1$. To remove the factor $O(\alpha)$, we run a $(1 + \epsilon')$-approximate NNS (instead of the $(1 + \epsilon)$-approximate one), where $\epsilon' = \epsilon - c\alpha$ while $\alpha = \frac{1}{c \log n}$. The running time is still as stated since $1/\alpha = \tilde{O}(1)$ and $n^{1/(1+\epsilon-1/\log n)} = O(n^{1/(1+\epsilon)})$.

## 5 Approximate 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 $2(1 + \varepsilon)c^*$.

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 $G_1, \ldots, G_k$ 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** *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 $c(M) \geq d(u, v) > r$. $\Box$

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

**Proof:** We make $T$ into a rooted tree by picking an arbitatary 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 $2\ell$. 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. $\square$

**Reduction to nearest-neighbour.** We now compute a spanning forest $\{T_1, \ldots, T_k\}$ of $P$ with the property that each edge of of the forest has length at most $r^*(1+\varepsilon)$, 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 $2r^*(1+\varepsilon)$. The running time is $\tilde{O}(dn^{1+1/(1+\epsilon)})$.

## 6 Metric Facility Location

We are given a metric space $(V, d)$ with $|V| = n$. We are also given a function $f : V \rightarrow \Re^+$ which maps each vertex in the metric space to a poisitive integer. For any $v \in V$ and any $S \subseteq V$, define $d(v, S)$ to be $\min_{u \in S} d(v, u)$ and define $f(S)$ to be $\sum_{u \in S} f(u)$. The Metric Facility Location problem is to find $S \subseteq V$ that minimizes its cost $C = f(S) + \sum_{v \in V} d(v, S)$. Let $S^*$ represent the minimizing set, and $C^*$ the corresponding cost.

The vertice in $V$ can be thought of as cities; $f(v)$ can be thought of as the cost of opening a facility in city $v$. The goal is to open a set of facilities to minimize the sum of the facility access costs and the facility opening costs.

The first constant factor approximation algorithm for this problem was obtained by Shmoys, Tardos, and Aardal [30]. After several improvements [16, 6], the current best known approximation factor is $1 + 2/e$ [7]. Jain and Vazirani [23] recently gave a factor 3 approximation to this problem which runs in $O(n^2 \log n)$ time, where $n$ is the number of vertices[4]. Their result is interesting

[4]Infact their running time is $O(n_c n_f \log(n_c n_f))$, where $n_c$ and $n_f$ are the number of citieis and facilities, respectively.

because of its low time complexity.

We use Nearest Neighbour techniques to obtain an even faster implementation of the algorithm of Jain and Vazirani. Our main result is an algorithm whose running time is $\tilde{O}(n^{1+\frac{1}{1+\epsilon}})$ and which produces a $3(1+O(\delta))(1+O(\epsilon))$-approximation for points in $\Re^d$. The $\tilde{O}$ notation hides terms which are logarithmic in $n$ or polynomial in $1/\delta$ and $1/\epsilon$. Here $\delta > 0$ and $\epsilon > 0$ are arbitrary constants. Our implementation uses the $\epsilon$-approximate dynamic PLEB (Point Location in Equal Balls) data structure [19, 20] and the dynamic bichromatic NN data structure by Eppstein [8, 9].

The main ideas behind our implementation are outlined in Appendix A. We now state the following theorem without proof.

**Theorem 1** *Our implementation of the algorithm of Jain and Vazirani's algorithm takes time* $\tilde{O}(n^{1+\frac{1}{1+\epsilon}})$ *(or* $\tilde{O}(n)$ *queries to an* $(1+\epsilon)$*-approximate NN oracle) and produces a* $3(1+O(\delta))(1+O(\epsilon))$*-approximation for point in* $\Re^d$.

## 7 Matching Using the Primal-dual Algorithm of Goemans and Williamson

The Goemans-Williamson matching algorithm can be described as follows. Initially, each vertex is put in a component of its own. Components with an odd number of vertices are said to be active. Those with an even number of vertices are said to be inactive. There is a ball around each vertex (initially of radius zero). The algorithm repeatedly grows the balls around all vertices in all active components at a uniform rate till two balls collide. The corresponding components are then merged – the edge joining the two vertices whose balls merged is added to the solution. This process conitnues till all components are inactive (i.e. of even size). Goemans and Williamson show that the resuling solution (a forest with only even components) has weight at most twice that of the optimal matching. It is trivial to then take each tree in this forest and convert it locally to a matching without increasing the cost of the solution.

The running time of this algorithm can be improved to $\tilde{O}(n^{1+\frac{1}{1+\epsilon}})$ for $\Re^d$ – the approximation guarantee now becomes $2+O(\epsilon)$. Our algorithm is sketched in Appendix B, but we describe here the main data structure used to achieve this reduction in running time. The data structure is an approximate Multichromatic NN data structure which consists of a set of points $X$ in a metric space $d$ with $|X| \leq n$, a mapping $C$ from $X$ to a set of colours $\{1,\dots,n\}$.

The data structure supports adding/deleting points to $X$, and the following query operation: *Given a query point* $q$*, if there is a point* $x$ *in* $X$ *such that* $d(q,x) \leq r$ *and* $C(x) \neq C(q)$*, then return a point* $y$ *in* $X$ *such that* $d(q,y) \leq r(1+\epsilon$ *and* $C(y) \neq C(q)$.

We now outline how this data structure can be created so that it uses $\tilde{O}(n^{1/(1+\epsilon)})$ time per operation – the $\tilde{O}$ notation hides terms polynomial in $1/\epsilon$ and $\log n$. We use $2\lfloor 1+\log n\rfloor$ Approximate NN data structures, referred to as $N_i^b$ where $b \in \{0,1\}$ and $0 \leq i \leq \lfloor \log n \rfloor$. $N_i^b$ contains all points $x$ such that the binary representation of $C(x)$ has $b$ in the $i$-th bit position. Clearly, insertions and deletions from the multichromatic NN data structure can be done in time $\tilde{O}(n^{1/(1+\epsilon)})$ since there must be exactly $\lfloor 1+\log n\rfloor = \tilde{O}(1)$ insertions/deletions into the the NN data structures. The search routine is also simple to implement – let $q_i$ represent the $i$-th bit in the binary representation of $C(q)$. Then the result $y$ must belong to one of the structures $N_i^{1-q_i}$, and we can make $\tilde{O}(1)$ calls to the underlying data structures $N_i^b$ to locate this point.

**Theorem 2** *The algorithm described in appendix B achieves an approximation of* $2+O(\epsilon)$ *in time* $\tilde{O}(n^{1+\frac{1}{1+\epsilon}})$ *(or* $\tilde{O}(n)$ *queries to an* $(1+\epsilon)$*-approximate NN oracle).*

## A Metric facility location

We are given a metric space $(V, d)$ with $|V| = n$. We are also given a function $f : V \to \Re^+$ which maps each vertex in the metric space to a poisitive integer. For any $v \in V$ and any $S \subseteq V$, define $d(v, S)$ to be $\min_{u \in S} d(v, u)$ and define $f(S)$ to be $\sum_{u \in S} f(u)$. The Metric Facility Location problem is to find $S \subseteq V$ that minimizes its cost $C = f(S) + \sum_{v \in V} d(v, S)$. Let $S^*$ represent the minimizing set, and $C^*$ the corresponding cost.

The vertice in $V$ can be thought of as cities; $f(v)$ can be thought of as the cost of opening a facility in city $v$. The goal is to open a set of facilities to minimize the sum of the facility access costs and the facility opening costs.

The first constant factor approximation algorithm for this problem was obtained by Shmoys, Tardos, and Aardal [30]. After several improvements [16, 6], the current best known approximation factor is $1 + 2/e$ [7]. Jain and Vazirani [23] recently gave a factor 3 approximation to this problem which runs in $O(n^2 \log n)$ time, where $n$ is the number of vertices[5]. Their result is interesting because of its low time complexity.

We use Nearest Neighbour techniques to obtain an even faster implementation of the algorithm of Jain and Vazirani. Our main result is an algorithm whose running time is $\tilde{O}(n^{2-\frac{\epsilon}{1+\epsilon}})$ and which produces a $3(1+O(\delta))(1+O(\epsilon))$-approximation for points in $\Re^d$. The $\tilde{O}$ notation hides terms which are logarithmic in $n$ or polynomial in $1/\delta$ and $1/\epsilon$. Here $\delta > 0$ and $\epsilon > 0$ are arbitrary constants. Our implementation uses an $\epsilon$-approximate dynamic PLEB (Point Location in Equal Balls) data structure [19, 20].

We will not present details of this implementation, but will briefly mention the main ideas. We first compute a lower bound $L$ such that $L \leq C^* \leq Ln^c$ for some fixed constant $c$. Such a bound can be computed in time $\tilde{O}(n)$. We then identify disjoint clusters of vertices such that there are no intercluster edges smaller than $\delta L/n^2$, and each cluster has a spanning tree with no edges larger than $\delta L/(4n)$. This clustering can be achieved in time $\tilde{O}(n)$ using the PLEB data structure along with a simple Union-Find data structure. We choose an arbitrary vertex from each cluster as the representative of the cluster and delete all non-representative vertices from our metric space. Each deleted vertex will use the same facility as its representative. Clearly, this can increase the cost of the solution by at most $\delta L/2$. We also open all facilities which have cost at most $\delta L/2n$. Again, this can increase the cost of the solution by at most $\delta L/2$. Thus, as a result of the above transformation, the cost of the solution can go up by at most a factor of $1 + \delta$.

The algorithm of Jain and Vazirani works in two phases.

### Phase 1, and our implementation

In the first phase, they define non-negative dual variables $\alpha_i$ and $\beta_{i,j}$ where $i$ ranges over facilities, and $j$ over cities. The dual constraints are $\sum_j \beta_{i,j} \leq f(i)$ for any facility $i$, and $\alpha_j < \beta_{i,j} + d(i, j)$ for any (facility,city) pair $(i, j)$. The objective is to maximize $\sum_j \alpha_j$. They try to maximize the dual objective while maintaining dual feasibility. They do this by growing balls around each active city (initially all cities ar active) at a uniform rate; $\alpha_j$ 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 mainitain dual feasibility, $\beta_{i,j}$ must also be increased now as $\alpha_j$ 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

[5]Infact their running time is $O(n_c n_f \log(n_c n_f))$, where $n_c$ and $n_f$ are the number of citieis and facilities, respectively.

increase, and therefore we have to stop growing all $\alpha_j$ 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 temprarily 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 $\delta L/n^2$ (by our preprocessing phase) and hence no events can happen till all the balls grow to a radius of $r_0 = \delta L/n^4$. We discretize distances[6] such that the only distances we consider are $r_k = r_0.(1+\epsilon)^k$ for $k \geq 0$; notice that $k$ can be at most $O(\log n)$. During each ball growing step, the radius of each active ball will be increased to the next discrete value. A $(k, \epsilon)-$PLEB is a $(1+\epsilon)$-approximate PLEB data structure used to locate points within balls of radius $r_0.(1+\epsilon)^k$. 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 $(k, \epsilon)$-PLEB 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. The same can be accomplished using the dynamic bichromatic nearest neighbour data structure by Eppstein [8, 9].

Detecting events of the second type is much harder. Let $X_k$ denote a $(k, \epsilon)$-PLEB 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 7** *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. $d_{ij} < r_k$

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

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

[6]After this discreitzation, the distances may not form a metric, but that does not affect the correctness of our implementation

□

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 $i$. It is well known that one can dynamically estimate $N_k(i)$ up to a factor of $(1 \pm \alpha)$ for any $\alpha > 0$ in time needed to perform approximate PLEB queries times poly$(\log n, 1/\alpha)$. The basic idea here is to check (approximately) if $N_k(i) > t$ by checking if there is *any* city within distance $r_k$ from $i$ which belongs to a random sample $S_t$ of cities of size $n/t$. This can be done by choosing $S_t$ in advance and building a PLEB data structure for it. Then the value of $N_k(i) > t$ can be estimated by performing binary search over $t$.

The time for each PLEB operation is $n^{1/(1+\epsilon)}$. Therefore, the running time of phase 1 is $\tilde{O}(n^{1+1/(1+\epsilon)})$ which is subquadratic.

The second phase of the algorithm by Jain and Vazirani is relatively straightforward, and can again be implemented using the PLEB data structure. Theorem 1 follows from the above description and from the corresponding theorem by Jain and Vazirani; we omit the proof from this version. The basic idea is that after the preprocessing phase, the algorithm of Jain and Vazirani can be forced to have the same execution sequence as our algorithm by modifying each edge weight by at most $(1 + O(\epsilon))$

## B Approximate matching in sub-quadratic time

The algorithm hinges on an approximate multichromatic NN which is described in Section 7. We give a sketch of the algorithm below.

### Preprocessing

First compute a 3-approximate bottleneck matching using techniques from Section 5. Let $C_B$ be the cost of the longest edge in this matching. Clearly, $C_B/3$ is a lower bound and $n \cdot C_B$ an upper bound on the cost of the minimum weight matching. Let $G'$ be the subgraph induced on $G$ by all edges of length less than $L = \delta.C_B/(6n^2)$. We find the connected components in $G'$ using an approximate-PLEB data structure. All even components of $G'$ are removed from $G$. We arbitrarily choose one vertex from each odd component and delete all the remaining vertices in the component from $G$.

We deleted an even number of vertices from each component. We separately determine an arbitrary matching for the deleted vertices of each component of $G'$ and all these matchings form an initial solution. Let $G''$ be the resulting graph. The matching algorithm described in the next section will use $G''$ as the input graph and the solution found will be appended to the initial solution found during the above preprocessing steps.

**Lemma 8** *The total running time of the preprocessing step is $\tilde{O}(n^{1+\frac{1}{1+\delta}})$. Also, a $(1+\epsilon)$ approximate solution to the minimum matching in the $G''$ can be appended to the initial solution found in the preprocessing step to obtain a $(1+\epsilon+\delta)$-approximate solution to the minimum weight matching in $G$.*

**Proof:** Omitted. □

### The matching algorithm

Since we are guaranteed that there are no edges smaller than $L$ in the graph, we can start with balls of size $L/2$. Since the cost of the matching produced by the Goemans and Willamson algorithm can also be bounded by the radius of the largest ball produced, we are guaranteed that no ball grows larger than $\frac{12n^3L}{\delta}$. We say that a vertex belongs to class $i$ if the size of its ball lies in $[L \cdot (1+\epsilon)^{i-1}, L.(1+\epsilon)^i)$. Let $r_i = L \cdot (1+\epsilon)^{i-1}$. Clearly, there can be at most $\beta = \log_{1+\epsilon} \frac{24n^3}{\delta} = O(\frac{\log n/\delta}{\epsilon})$ distinct classes. Define $\Delta = \gamma \cdot \log_{1+\epsilon}(1/\epsilon)$, where $\gamma$ is a large enough constant. Intuitively, when a ball grows, it can only collide with balls whose class is at most $\Delta$ larger. Also the number of classes is $\tilde{O}(1)$.

Initially each vertex is assigned a different colour. For each class $i$ we maintain the following data structures.

1. An approximate multichromatic NN data structure ACTIVE$_i$ that contains all balls in class $i$ that the algorithm has currently marked as active

2. An approximate multichromatic NN data structure INACTIVE$_i$ that contains all balls in class $i$ that the algorithm has currently marked as inactive

   *Note: The algorithm does not guarantee that balls will be correctly marked. In fact balls of the same colour may be marked as active in some classes and inactive in others. However the algorithm will maintain the invariant that all the balls of the same colour in the same class must belong to the same NN data structure (i.e. either all in ACTIVE$_i$ or all in INACTIVE$_i$)*

3. ACTIVE-ACCURATE$_i$, ACTIVE-INACCURATE$_i$, INACTIVE-ACCURATE$_i$, and INACTIVE-INACCURATE$_i$. The structure ACTIVE-ACCURATE$_i$ stores all the colours which have a ball that belongs to class $i$, are marked as ACTIVE in class $i$ and are indeed active. The data structures ACTIVE-INACCURATE, INACTIVE-ACCURATE, and INACTIVE-INACCURATE are similarly defined. Each operation (search, insert, delete) takes $O(\log n) = \tilde{O}(1)$ time.

4. For each colour $i$, a list of the balls that belong to this class and have colour $i$. Each operation (search, insert, delete) takes $O(\log n) = \tilde{O}(1)$ time.

We repeatedly do the following.

1. Find an active ball which is in the smallest class that contains an active ball. Recall that balls may be marked inaccurately so this amounts to finding the smallest class $i$ where either ACTIVE-ACCURATE$_i$ or INACTIVE-INACCURATE$_i$ is non-empty. This step takes $\tilde{O}(1)$ time.

2. For all classes $j$ where $j \leq i+\Delta$ we correctly mark all the colours that are incorrectly marked. This involves deleting the incorrectly marked balls from one of ACTIVE$_i$/INACTIVE$_i$ and adding them to the other.

3. Grow all active balls[7]by a radius $r_i\epsilon$. Balls whose radius increases beyond the upper limit for their class get "promoted" to the new class. The total time for all these operations throughout the algorithm is $\tilde{O}(n^{1+\frac{1}{1+\epsilon}})$

4. For each active ball in classes $j$ where $j \leq i + \Delta$, find collisions with any of the balls of a different colour in classes $k$ where $k \leq i + \Delta$. These steps take $\tilde{O}(n^{1+\frac{1}{1+\epsilon}})$ time during the course of the algorithm.

[7]For balls which belong to a class larger than $i + \Delta$ we can use a randomized sampling scheme to ensure that no ball has its radius increased more than $\tilde{O}(1)$ times throughout the course of the algorithm.

5. Each time a collision is found, the two colours are merged (the colour with less vertices is changed to be the same as the colour with larger number of vertices). If the collision is between two active balls, the new colour is marked inactive; else it is marked active. All the data structures are updated to reflect this. Since the total number of colour changes for all balls is $O(n \log n)$, the total time for all these steps is $\tilde{O}(n^{1+\frac{1}{1+\epsilon}})$.

All the steps in part (2) take time at most $\tilde{O}(n^{1+\frac{1}{1+\epsilon}})$; the proof is omitted from this version.

This completes our description of the sub-quadratic time approximation algorithm for matching based on the primal dual algorithm of Goemans and Williamson.

## 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 "Subquadratic Approximation Algorithms For clustering Problems in High Dimensional Spaces", STOC'99, to appear.

[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.

to appear.

[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, R. Motwani, a draft of the final version of the above. Available at http://theory.stanford.edu/~indyk/nndraft.ps

[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 Lipshitz 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, to appear.

[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 Multidimensional 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.