# The Euclidean *k*-Supplier Problem

Viswanath Nagarajan[1], Baruch Schieber[1], and Hadas Shachnai[2⋆]

[1] IBM T.J. Watson Research Center, Yorktown Heights, NY 10598
[2] Computer Science Department, Technion, Haifa 32000, Israel

**Abstract.** In the *k-supplier* problem, we are given a set of clients $C$ and set of facilities $F$ located in a metric $(C \cup F, d)$, along with a bound $k$. The goal is to open a subset of $k$ facilities so as to minimize the maximum distance of a client to an open facility, i.e., $\min_{S \subseteq F:|S|=k} \max_{v \in C} d(v, S)$, where $d(v, S) = \min_{u \in S} d(v, u)$ is the minimum distance of client $v$ to any facility in $S$.

We present a $1 + \sqrt{3} \approx 2.73$ approximation algorithm for the $k$-supplier problem in Euclidean metrics. This improves the 3-approximation algorithm of Hochbaum and Shmoys [9] which also holds for general metrics (where it is known to be tight). By a result of Feder and Greene [5], it is NP-hard to approximate Euclidean $k$-supplier to better than a factor of $\sqrt{7} \approx 2.65$, even in dimension two. Our algorithm is based on a relation to the *edge cover* problem.

We also present a nearly linear $O(n \cdot \log^2 n)$ time algorithm for Euclidean $k$-supplier in constant dimensions that achieves an approximation ratio of $\approx 2.965$, where $n = |C \cup F|$. The previously known nearly linear $O(n \log k)$ time algorithm for this problem by Feder and Greene [5] achieves a 3-approximation.

## 1 Introduction

Location problems are an important class of combinatorial optimization problems that arise in a number of applications, e.g., choosing sites for opening plants, placing servers in a network, and clustering data. Moreover, the underlying distance function in many cases is Euclidean ($\ell_2$ distance). In this paper, we study a basic location problem on Euclidean metrics.

The *Euclidean k-supplier* problem consists of $n$ points in $p$-dimensional space, that are partitioned into a client set $C$ and a set of facilities $F$. Additionally, we are given a bound $k \leq |F|$. The objective is to open a set $S \subseteq F$ of $k$ facilities that minimizes the maximum distance of a client to its closest facility. The $k$-supplier problem is a generalization of the *k-center* problem, where the client and facility sets are identical.

On general metrics, the $k$-supplier problem admits a 3-approximation algorithm [9]. There is a better 2-approximation algorithm for $k$-center, due to Hochbaum and Shmoys [8] and Gonzalez [6]. Moreover, these bounds are best

⋆ Work partially supported by the Israel Science Foundation (grant number 1574/10), and by funding for DIMACS visitors.

possible assuming $P \neq NP$. On Euclidean metrics, Feder and Greene [5] showed that it is NP-hard to approximate $k$-supplier better than $\sqrt{7}$ and $k$-center better than $\sqrt{3}$. Still, even on 2-dimensional Euclidean metrics, the best known approximation ratios remain 3 for $k$-supplier and 2 for $k$-center. In this paper, we provide the following improvement for Euclidean $k$-supplier:

**Theorem 1.** *There is a $(1+\sqrt{3})$-approximation algorithm for the Euclidean $k$-supplier problem in any dimension.*

It is worth noting that it remains NP-hard to approximate the $k$-supplier problem better than 3 if we use $\ell_1$ or $\ell_\infty$ distances, even in 2-dimensional space [5]. Thus, our algorithms make heavy use of the Euclidean metric properties.

In many applications, such as clustering, the size of the input data may be very large. In such settings it is particularly useful to have fast (possibly linear time) algorithms. Geometry plays a crucial role here, and many optimization problems have been shown to admit much faster approximation algorithms in geometric settings than in general metrics, for example TSP [1], $k$-median [7, 10], or matching [15, 14, 1]. These papers consider the setting of low constant dimension, which is also relevant in practice; the running time is typically exponential in the dimension. For the Euclidean $k$-supplier problem in constant dimension, [5] gave a nearly linear $O(n \log k)$ time 3-approximation algorithm; whereas the best running time in general metrics is quadratic $O(nk)$ [9, 6]. Extending some ideas from Theorem 1, we obtain the following nearly linear time algorithm for Euclidean $k$-supplier having an approximation ratio better than 3.

**Theorem 2.** *There is an $O(n \cdot \log^2 n)$ time algorithm for Euclidean $k$-supplier in constant dimensions that achieves an approximation ratio $\approx 2.965$.*

It is unclear if our algorithm from Theorem 1 admits a near-linear time implementation: the best running time that we obtain for constant dimensions is $O(n^{1.5} \log n)$. Both our algorithms extend easily to the *weighted* $k$-supplier problem, where facilities have weights $\{w_f : f \in F\}$, and the goal is to open a set of facilities having total weight at most $k$.

**Our Techniques and Outline.** The $(1+\sqrt{3})$-approximation algorithm (Theorem 1) is based on a relation to the *minimum edge cover* problem, and is very simple. Recall, the edge cover problem [13] involves computing a subset of edges in an undirected graph so that every vertex is incident to some chosen edge; this problem is equivalent to maximum matching.[3] The entire algorithm is:

> "Guess" the value of $\mathsf{opt}$. $P$ is a maximal subset of clients $C$ whose pairwise distance is more than $\sqrt{3} \cdot \mathsf{opt}$. Graph $G$ on vertices $P$ contains an edge for each pair $u, v \in P$ of clients that are both within distance $\mathsf{opt}$ from the same facility. Compute the minimum edge cover in $G$ and output the corresponding facilities.

[3] In any $n$-vertex graph without isolated vertices, it is easy to see that the minimum edge cover has size $n$ minus the maximum matching.

The key property (which relies on the Euclidean metric) is that any facility can "cover" (within distance $\mathsf{opt}$) at most two clients of $P$, which leads to a correspondence between $k$-supplier solutions and edge-covers in $G$. The main difference from [9, 8, 6] is that our algorithm uses information on *pairs* of clients that can be covered by a single facility.

To implement the algorithm we apply the fastest known algorithm for edge-cover, due to Micali and Vazirani [11] that runs in time $O(E_G\sqrt{V_G})$, which in our setting is $O(n^{1.5})$. However, in $p$-dimensional space, the algorithm to construct graph $G$ takes $O(p\,n^2)$ time in general,[4] which results in an overall runtime of $O(p\,n^2)$. These results appear in Section 2.

When the *dimension is constant*, which is often the most interesting setting for optimization problems in Euclidean space, we show that a much better runtime can be achieved. Here, we can make use of good *approximate nearest neighbor* (ANN) data structures and algorithms [2, 4, 3]. These results state that with $O(n \log n)$ pre-processing time, one can answer $(1+\epsilon)$-approximate nearest-neighbor queries in $O(\log n)$ time each; where $\epsilon > 0$ is any constant. This immediately gives us an $O(n \log n)$ time algorithm to construct $G$, and hence an $\tilde{O}(n^{1.5})$ time implementation of Theorem 1 at the loss of a $1+\epsilon$ factor. Also, in the special case of dimension two, we can show that $G$ is *planar* (see Section 2), so we can use the faster $O(n^{1.17})$ time planar matching algorithm due to Mucha and Sankowski [12] and obtain an $\tilde{O}(n^{1.17})$ time implementation of Theorem 1. However, there are no additional properties of $G$ that we are able to use. In particular:

- Any degree 3 planar graph can be obtained as graph $G$ for some instance of 2-D Euclidean $k$-supplier, and the fastest known matching algorithm even on this family of graphs still runs in $O(n^{1.17})$ time [12]. Indeed, there is a linear time reduction [12] from matching on general planar graphs to degree 3 planar graphs.
- Even in 3-D, the graph $G$ does not necessarily exclude any fixed minor.[5] So, for higher constant dimensions, we need a general edge cover algorithm [11].

In Theorem 2 we provide a different algorithm (building on ideas from Theorem 1) that achieves near-linear running time, but a somewhat worse approximation ratio of 2.965: which is still better than the previous best bound of 3. The main idea here is to reduce to an edge cover problem on (a special class of) *cactus graphs*. Since (weighted) edge cover (and matching) on cactus graphs can be solved in linear time, the overall running time is dominated by the procedure to construct this edge-cover instance. Although the graph construction procedure here is more complicated, we show that it can also be implemented in $\tilde{O}(n)$ time using ANN algorithms [2]. The details appear in Section 3.

*Remark:* Our ideas do not extend directly to give a better than 2-approximation for the Euclidean $k$-center problem, which remains an interesting open question.

[4] The factor of $p$ appears because, given a pair of points in $p$-dimension it takes $O(p)$ time to even compute the Euclidean distance between them.

[5] Recall that a graph is planar iff it does not contain $K_5$ nor $K_{3,3}$ as a minor.

## 2 The $(1+\sqrt{3})$-Approximation Algorithm

For any instance of the $k$-supplier problem, it is clear that its optimal value is one of the $|F| \cdot |C|$ distances between clients and facilities. As with most bottleneck optimization problems [9], our Algorithm 2.1 uses a procedure that takes as input an additional parameter $L$ (which is one of the possible objective values) and outputs one of the following:

1. a certificate showing that the optimal $k$-supplier value is more than $L$, or
2. a $k$-supplier solution of value at most $\alpha \cdot L$.

Above, $\alpha = 1 + \sqrt{3}$ is the approximation ratio. The final algorithm uses binary search to find the best value of $L$. We now prove the correctness of this algorithm.

**Algorithm 2.1** Algorithm for Euclidean $k$-supplier

1: pick a maximal subset $P \subseteq C$ of clients such that each pairwise distance in $P$ is more than $\sqrt{3} \cdot L$.
2: construct graph $G = (P, E)$ with vertex set $P$ and edge set $E = E_1 \cup E_2$

$$E_1 = \{(u, v) \ : \ u, v \in P, \exists f \in F \text{ with } d(u, f) \leq L \text{ and } d(v, f) \leq L\}. \quad (1)$$

$$E_2 = \{(u, u) \ : \ u \in P, \exists f \in F \text{ with } d(u, f) \leq L \text{ and } \forall v \in P, (u, v) \notin E_1\}. \quad (2)$$

3: compute the *minimum edge cover* $\Gamma \subseteq E$ in $G$.
4: **if** $|\Gamma| > k$ **then**
5: the optimal value is larger than $L$.
6: **else**
7: output the facilities corresponding to $\Gamma$ as solution.

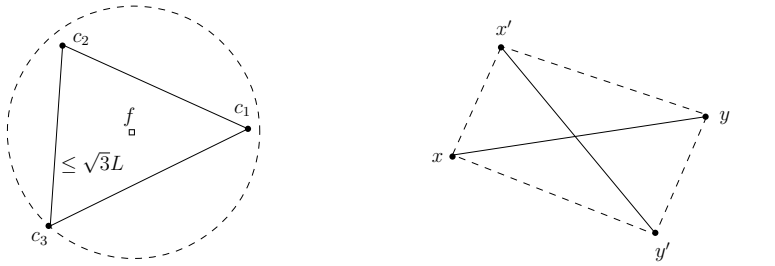
We start with a key property that makes use of Euclidean distances.

**Lemma 1.** *For any facility $f \in F$, the number of clients in $P$ that are within distance $L$ from $f$ is at most two.*

*Proof.* To obtain a contradiction suppose that there is a facility $f$ with three clients $c_1, c_2, c_3 \in P$ having $d(f, c_i) \leq L$ for $i \in \{1, 2, 3\}$. Since all vertices are in Euclidean space, in the plane containing $c_1, c_2$ and $c_3$, there is a circle of radius $L$ that has $\{c_i\}_{i=1}^3$ in its interior. See Figure 1 (A). Hence, the minimum pairwise distance in $\{c_i\}_{i=1}^3$ is at most $\sqrt{3} \cdot L$. This contradicts the fact every pairwise distance between vertices in $P$ is greater than $\sqrt{3} \cdot L$. The lemma now follows. ■

This lemma provides a one-to-one correspondence between the edges $E$ defined in (1)-(2) and facilities $H = \{f \in F : \exists u \in P \text{ with } d(u, f) \leq L\}$. Note that facilities in $H$ that are within distance $L$ of exactly one client in $P$ give rise to self loops in $E$. Clearly, if the optimal $k$-supplier value is at most $L$ then there is a set $H'$ of at most $k$ facilities in $H$ so that each client in $P$ lies within distance $L$ of some facility of $H'$. In other words,

**Claim 3** *If the optimal $k$-supplier value is at most $L$ then graph $G$ contains an edge cover of size at most $k$.*




Clients $c_1, c_2, c_3$ within distance $L$ from facility $f$

Possible edge crossing between $(x, y)$ and $(x', y')$

Fig. 1: Examples for (A) Lemma 1 and (B) Lemma 3.

Recall that an *edge cover* in an undirected graph is a subset of edges where each vertex of the graph is incident to at least one edge in this subset. The minimum size edge cover of a graph can be computed in polynomial time using algorithms for *maximum matching*, see, e.g., [13]. By Claim 3, if the minimum edge cover $\Gamma$ is larger than $k$ then we have a certificate for the optimal $k$-supplier value being more than $L$. This justifies Step 5. On the other hand, if the minimum edge cover $\Gamma$ has size at most $k$ then (in Step 7) we output the corresponding facilities (from $H$) as the solution.

**Lemma 2.** *If the algorithm reaches Step 7 then $\Gamma$ corresponds to a $k$-supplier solution of value at most $(1 + \sqrt{3})L$.*

*Proof.* To reduce notation, we use $\Gamma$ to denote both the edge cover in $G$ and its corresponding facilities from $H$. Since $\Gamma$ is an edge cover in $G$, each client $u \in P$ is within distance $L$ of some facility in $\Gamma$.

$$\max_{u \in P} d(u, \Gamma) \quad \leq \quad L. \tag{3}$$

Now, since $P \subseteq C$ is a maximal subset satisfying the condition in Step 1, for each client $v \in C$ there is some $u \in P$ with $d(u, v) \leq \sqrt{3}L$. Using (3) and triangle inequality, it follows that $\max_{v \in C} d(v, \Gamma) \leq (\sqrt{3} + 1)L$. ∎

Finally, we perform a binary search over the parameter $L$ to determine the smallest value for which there is a solution. This proves Theorem 1.

**Weighted supplier Problem.** Our algorithm extends easily to the weighted supplier problem, where facilities have weights $\{w_f : f \in F\}$ and the objective is to open facilities of total weight at most $k$ that minimizes the maximum distance to any client. In defining edges in the graph $G$ (Equation (1)-(2)) we also include weights of the respective facilities. Then we find a *minimum weight* edge cover $\Gamma$, which can also be done in polynomial time [13].

### 2.1 Running Time

We use $n = |F| + |C|$ to denote the total number of vertices. For arbitrary dimension $p \geq 2$, the running time can be naïvely bounded by $O(p\, n^2)$. This

running time is dominated by the time it takes to construct the graph $G$. The edge cover problem can be solved via a matching algorithm [11, 13] that runs in time $O(E(G)\sqrt{V(G)}) = O(n^{3/2})$ since in our setting $V(G) \leq |C|$ and $E(G) \leq |F|$.

When dimension $p$ is constant, we provide a better running time implementation. There are two main parts in our algorithm: constructing the graph $G$ and solving the edge cover problem on $G$. A naïve implementation of the first step results in an $O(n^2)$ running time. We show below that the runtime can be significantly improved, while incurring a small loss in the approximation ratio.

**Constructing graph G.** The main component here is a fast data structure for *approximate nearest neighbor search* from Arya et al. [2].

**Theorem 4 ([2]).** *Consider a set of $n$ points in $\mathbb{R}^p$. Given any $\epsilon > 0$, there is a constant $c \leq p\lceil 1 + 6p/\epsilon \rceil^p$ such that it is possible to construct a data structure in $O(pn \log n)$ time and $O(pn)$ space, with the following properties:*

- *For any "query point" $q \in \mathbb{R}^p$ and integer $\ell \geq 1$, a sequence of $\ell$ $(1+\epsilon)$-approximate nearest neighbors of $q$ can be computed in $O((c+\ell p)\log n)$ time.*
- *Point insertion and deletion can be supported in $O(\log n)$ time per update.*

We will maintain such a data structures $\mathcal{P}$ for clients. First, we implement the step of finding a maximal "net" $P \subseteq C$ in Algorithm 2.2.

**Algorithm 2.2** Algorithm for computing vertices $P$ of $G$

1: initialize $P \leftarrow \emptyset$ and $\mathcal{P} \leftarrow \emptyset$.
2: **for** $v \in C$ **do**
3: $\quad v' \leftarrow$ approximate nearest neighbor of $v$ in $\mathcal{P}$ (or NIL if $\mathcal{P} = \emptyset$).
4: $\quad$ **if** $d(v, v') > \sqrt{3}(1+\epsilon)L$ or $v'$=NIL **then**
5: $\quad\quad P \leftarrow P \cup \{v\}$ and insert $v$ into $\mathcal{P}$.
6: output $P$.

Since we use $(1+\epsilon)$-approximate distances, the condition in Step 4. ensures that every pairwise distance in the final set $P$ is at least $\sqrt{3}L$. Moreover, for each $u \in C \setminus P$, there is some $v \in P$ satisfying $d(u, v) \leq \sqrt{3}(1+\epsilon)L$. By Theorem 4, the time taken for each insertion and nearest-neighbor query in $\mathcal{P}$ is $O(\log n)$; so the total running time of this Algorithm 2.2 is $O(n \log n)$.

Next, Algorithm 2.3 shows how to compute the edge set $E$ in (1)-(2).

Since all pairwise distances in $P$ are larger than $\sqrt{3}L$, Lemma 1 implies that each facility $f \in F$ is within distance $L$ of at most two clients in $P$. This is the reason we only look at the *two* approximate nearest neighbors ($u$ and $v$) of $f$. Again, the condition for adding edges ensures that there is an edge in $E$ for every facility in the set $H = \{f \in F : \exists u \in P \text{ with } d(u, f) \leq L\}$; since we use approximate distances, there might be more edges in $E$. By Theorem 4, the

**Algorithm 2.3** Algorithm for computing edges $E$ of $G$

```
construct data structure P containing points P, and initialize E ← ∅.
for f ∈ F do
    u ← approximate nearest neighbor of f in P.
    v ← approximate second nearest neighbor of f in P.
    if d(u, f) ≤ (1 + ε)L and d(v, f) > (1 + ε)L then
        set E ← E ∪ {(u, u)}.
    if d(u, f) ≤ (1 + ε)L and d(v, f) ≤ (1 + ε)L then
        set E ← E ∪ {(u, v)}.
output E.
```

time for each 2-nearest neighbors query is $O(c \log n)$. Thus, the total time is $O(cn \log n)$, which is $O(n \log n)$ for any constant dimension $p$.

**Computing Edge-Cover on G.** Finding a minimum size edge cover is equivalent to finding a maximum cardinality matching on $G$. The fastest algorithm for matching on general graphs takes $O(E(G)\sqrt{V(G)})$ time [11]. This results in an $O(n^{3/2})$ running time in our setting, since we only deal with sparse graphs.

We can obtain a better running time in $p = 2$ dimensions, by using the following additional property of the graph $G$.

**Lemma 3.** *If dimension is $p = 2$ and $\epsilon \leq 0.2$, the graph $G$ is planar.*

*Proof.* Consider the natural drawing of graph $G$ in the plane: each vertex in $P$ is a point, and each edge $(u, v) \in E$ is represented by the line segment connecting $u$ and $v$. To obtain a contradiction suppose that there is some crossing, say between edges $(x, y)$ and $(x', y')$, see also Figure 1 (B).

Notice that the distance between the end-points of any edge in $E$ is at most $2(1 + \epsilon)L$, and the distance between any pair of points in $P$ is at least $\sqrt{3}L$. Hence (setting $\epsilon \leq 0.2$), for any edge $(u, v)$ and vertex $w \in P$, the angle $uwv$ is strictly less than $90^o$.

Using this observation on edge $(x, y)$ and points $x'$ and $y'$, we obtain that the angles $xx'y$ and $xy'y$ are both strictly smaller than $90^o$. Similarly, for edge $(x', y')$ and points $x$ and $y$, angles $x'xy'$ and $x'yy'$ are also strictly smaller than than $90^o$. This contradicts with the fact that the sum of interior angles of quadrilateral $xx'yy'$ must equal $360^o$. ∎

Based on this lemma, we can use the faster $O(n^{\omega/2})$ time randomized algorithm for matching on planar graphs, due to Mucha and Sankowski [12]. Here, $\omega < 2.38$ is the matrix multiplication exponent. Thus, we have shown:

**Theorem 5.** *For any constant $0 < \epsilon < 0.2$, there is a $(1 + \epsilon)(1 + \sqrt{3})$ factor approximation algorithm for Euclidean $k$-supplier that runs in time: $O(n^{1.5} \log n)$ for any constant dimension $p$, and $O(n^{1.17} \log n)$ for $p = 2$ dimensions.*

The additional $\log n$ factor is because we need to perform binary search over the parameter $L$. We note that for larger dimension $p \geq 3$, the graph $G$ does

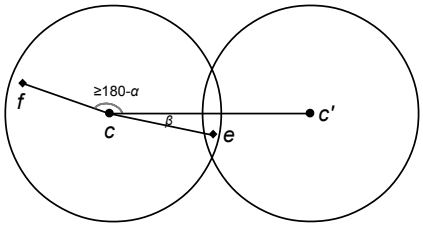
not necessarily have such nice properties. In particular, even in 3-dimensions $G$ does not exclude any fixed minor. So it is unclear whether the faster matching algorithms on graphs with excluded minors [16] can be used in our setting.

## 3 Nearly Linear Time 2.965-Approximation Algorithm

In this section, we give an $O(n \log^2 n)$ time approximation algorithm for Euclidean $k$-supplier in fixed dimensions. The approximation ratio we obtain is 2.965 which is worse than the $1 + \sqrt{3} \approx 2.73$ bound from the previous section. We do not know a near-linear time implementation achieving that stronger bound. The algorithm here uses some ideas from the previous reduction to an edge-cover problem. But to achieve near-linear running time, we do not want to solve a general matching problem (even on planar graphs). Instead, we show that using additional Euclidean properties one can reduce to an edge-cover problem on (a special case of) *cactus* graphs. This approach gives a nearly linear time algorithm, since matching on cactus graphs can be solved in linear time.

Let $0 < \rho < 1$ be some constant and $0^o < \alpha, \beta < 30^o$ be angles, the values of which will be set later. To reduce notation, throughout this section, we normalize distances so that the parameter $L = 1$ (guess of the optimal value). For any point $v$, we denote the *ball* of radius one centered at $v$ by $B(v)$. Two clients $c$ and $c'$ are said to *intersect* if **(i)** $d(c, c') \leq 2$ and **(ii)** there is some facility $f \in B(c) \cap B(c')$; if in addition, $d(c, c') > 2 \cos \beta$ then we call it a *fringe intersection*. Note that when $c$ and $c'$ have a fringe intersection, for any point $v \in B(c) \cap B(c')$ the angles $\angle vcc'$ and $\angle vc'c$ are at most $\beta$.

Given a client $c$ and facility $f \in B(c)$, we say that another client $c'$ is in *antipodal position* with respect to (abbreviated w.r.t.) $\langle f, c \rangle$ if the angle $fcc'$ is more than $180^o - \alpha$; if in addition, $c$ and $c'$ have a fringe intersection, we say that $c'$ has a fringe antipode intersection with $\langle f, c \rangle$. See figure to the right.




As in the previous section, the algorithm here builds a graph $G$ on clients as vertices and facilities as edges. However, this procedure is more complex, since we want the resulting graph to have simpler structure: so that the edge cover problem on $G$ can be solved in linear time.

The graph $G$ is constructed iteratively, where each iteration adds a new component $H$ as follows. We initialize $H$ with an arbitrary pair $c_1, c_2$ of clients that have a fringe intersection, say with facility $f_0 \in B(c_1) \cap B(c_2)$; so $H$ has vertices $V(H) = \{c_1, c_2\}$ and an edge $(c_1, c_2)$ that is labeled $f_0$. (If there is no such pair, we pick an arbitrary client $c_0$ and set $H = \{c_0\}$ to be a singleton component.) Throughout the iteration, we maintain (at most) two *endpoint clients* $x$ and $y$ along with facilities $f \in B(x)$ and $g \in B(y)$; the role of these will become clear shortly. We will also refer to the tuples $\langle x, f \rangle$ and $\langle y, g \rangle$ as endpoints. Initially, set $x \leftarrow c_1$, $f \leftarrow f_0$, $y \leftarrow c_2$ and $g \leftarrow f_0$.

We repeatedly add to component $H$ a new client $c$ satisfying the following:

- $c$ has a fringe antipode intersection with either $\langle x, f\rangle$ or $\langle y, g\rangle$.
- If $x \neq y$ and $c$ intersects both, then $c$ must be fringe antipode w.r.t. both $\langle x, f\rangle$ and $\langle y, g\rangle$.
- $c$ does not intersect any client in $V(H) \setminus \{x, y\}$.

For a client $c$ that satisfies these three conditions and is added to $H$, we distinguish the following two cases:

Case 1: Client $c$ intersects exactly one of $\{x, y\}$, say $x$ (the other case is identical). Let $f' \in B(x) \cap B(c)$ denote the facility in the (fringe) intersection of $x$ and $c$. Add vertex $c$ to $V(H)$ and an edge $(x, c)$ labeled $f'$. Also set $x \leftarrow c$ and $f \leftarrow f'$.

Case 2: Client $c$ intersects both $x$ and $y$. Let $f_1 \in B(x) \cap B(c)$ and $f_2 \in B(y) \cap B(c)$ denote the facilities in the (fringe) intersections of $x$ and $c$ and of $y$ and $c$, respectively. In this case, add vertex $c$ to $V(H)$, and edges $(c, x)$ labeled $f_1$ and $(c, y)$ labeled $f_2$. Set $x \leftarrow c$, $f \leftarrow f_1$, $y \leftarrow c$ and $g \leftarrow f_2$.

The construction of component $H$ ends when there are no new clients that can be added. At this point, we remove all clients that intersect with any client in $V(H)$ (these will be covered by a subset of facilities in $E(H)$), and iterate building the next component of $G$. Finally, we output an *edge cover* of $G$ as the solution to the $k$-supplier problem.

Next, we prove some useful properties of the graph $G$.

**Lemma 4.** *Each component $H$ is a cactus, where its simple cycles are linearly ordered. Hence, the edge-cover problem on $G$ is solvable in linear time.*

*Proof.* It is easy to show by induction that $H$ is a cactus with linearly ordered simple cycles. In each step, $H$ grows by a new vertex $c$ and (i) one edge from $c$ to $x$ (after which $x \leftarrow c$), or (ii) two edges, from $c$ to $x$ and $y$ (after which $x, y \leftarrow c$).

A linear time algorithm for (weighted) edge-cover (and weighted matching) on cactus graphs can be obtained via a dynamic program. Here we just state an algorithm for the unweighted case of linearly ordered cycles. Such a graph $G$ is given by a sequence $\langle v_1, v_2, \ldots, v_r\rangle$ of vertices, disjoint cycles $C_1, \ldots, C_{r-1}$ and a path $C_r$ containing $v_r$. The cycles $C_1, \ldots, C_{r-1}$ and path $C_r$ are vertex-disjoint except at the $v_i$s: for each $j \in [r-1]$, $C_j \cap \{v_i\}_{i=1}^r = \{v_j, v_{j+1}\}$, and $C_r \cap \{v_i\}_{i=1}^r = \{v_r\}$.

For any $i \in [r]$, let $T[i, 0]$ denote the minimum edge cover in the graph $G_i := C_i \cup C_{i+1} \cdots \cup C_r$; and $T[i, 1]$ the minimum edge cover for graph $G_i$ when vertex $v_i$ is not required to be covered. The base cases $T[r, 0]$ and $T[r, 1]$ can be easily computed by considering all minimal edge covers of path $C_r$. We can write a recurrence for $T[i, *]$ as follows. Let $e_{i+1}$ and $f_{i+1}$ denote the two edges incident to $v_{i+1}$ in the cycle $C_i$. Define the following minimal edge covers in $C_i$ (each is unique subject to its condition).

- $\Gamma_i^1$ (resp. $\Gamma_i^2$) contains neither $e_{i+1}$ nor $f_{i+1}$, and covers vertices $C_i \setminus v_{i+1}$ (resp. $C_i \setminus \{v_i, v_{i+1}\}$).

- $\Gamma_i^3$ (resp. $\Gamma_i^4$) contains $e_{i+1}$ but not $f_{i+1}$, and covers vertices $C_i$ (resp. $C_i \setminus \{v_i\}$).
- $\Gamma_i^5$ (resp. $\Gamma_i^6$) contains $f_{i+1}$ but not $e_{i+1}$, and covers vertices $C_i$ (resp. $C_i \setminus \{v_i\}$).
- $\Gamma_i^7$ (resp. $\Gamma_i^8$) contains both $f_{i+1}$ and $e_{i+1}$ and thus not the other edge incident to $e_{i+1}$, and covers vertices $C_i$ (resp. $C_i \setminus \{v_i\}$).

Then we have for all $r \in [r-1]$,

$$T[i,0] := \min\left\{T[i+1,0] + \Gamma_i^1,\ T[i+1,1] + \min\left\{\Gamma_i^3, \Gamma_i^5, \Gamma_i^7\right\}\right\}$$

$$T[i,1] := \min\left\{T[i+1,0] + \Gamma_i^2,\ T[i+1,1] + \min\left\{\Gamma_i^4, \Gamma_i^6, \Gamma_i^8\right\}\right\}$$

Clearly this dynamic program can be solved in linear time. ■

**Claim 6** *If the optimal k-supplier value is at most* 1*, then graph G has an edge cover of size k.*

*Proof.* We show that each facility can cover at most two clients in $V(G)$, and that there is an edge in $G$ between *every* pair of clients in $V(G)$ that can be covered by a single facility. This would imply the claim.

Note that if a pair of vertices in $V(G)$ intersect then this intersection is fringe intersection, i.e., any such pairwise distance is at least $2\cos\beta \geq \sqrt{3}$. Hence, by Lemma 1, each facility can cover (within distance one) at most two clients of $V(G)$. Moreover, by the construction of each component $H$, the edge set $E(H)$ contains *all* intersections between pairs of vertices in $V(H)$. Also, clients in different components of $G$ do not intersect. This is because we remove all clients intersecting with $V(H)$ after constructing component $H$. ■

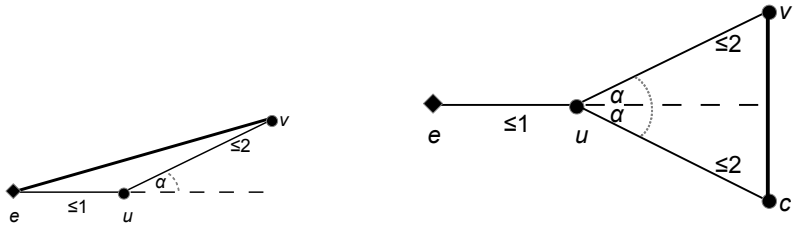
We now prove that this algorithm achieves an approximation ratio $3-\rho$. Below we consider a particular component $H$. The variables $x, f, y, g$ will denote their values at the end of $H$'s construction (unless specified otherwise).

**Claim 7** *For any client $u \in V(H)$ and edge (facility) $e = (u, u') \in E(H)$ such that $\langle e, u\rangle \notin \{\langle f, x\rangle, \langle g, y\rangle\}$, and client $v \in C$ that intersects $u$, either $d(e,v) \leq 3-\rho$ or $d(v, V(H)) \leq 2-\rho$.*

*Proof.* The Claim holds trivially for $v \in V(H)$. Consider clients $u \in V(H)$ and $v \in C \setminus V(H)$ as stated. If $d(u,v) \leq 2\cos\beta$ then, clearly, $d(v, V(H)) \leq d(v,u) \leq 2\cos\beta$. Else, if $v$ is *not* in antipodal position w.r.t. $\langle e, u\rangle$, then by the cosine rule, $d(e,v) \leq \sqrt{1^2 + 2^2 + 2\cdot 2\cos\alpha}$ (see Figure 2a)

Below, we assume that $2\cos\beta \leq d(u,v) \leq 2$ and $u$ is in antipode position w.r.t. $\langle e, u\rangle$. Since $\langle e, u\rangle \notin \{\langle f, x\rangle, \langle g, y\rangle\}$, one of the following two cases must be true.

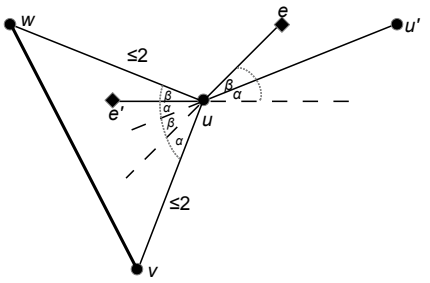
Case 1: At some earlier iteration, $\langle e, u\rangle$ was an endpoint and some client $c$ was added due to a fringe antipode intersection w.r.t. $\langle e, u\rangle$. In this case we will bound $d(v, V(H)) \leq d(v,c)$. Since both $v$ and $c$ are in antipode position w.r.t. $\langle e, u\rangle$, the angle $\angle vuc$ is at most $2\alpha$. Again by the cosine rule, $d(v,c) \leq \sqrt{2^2 + 2^2 - 2\cdot 2\cdot 2\cos 2\alpha}$ (see Figure 2b).




(a) $v$ is not antipodal w.r.t. $\langle e, u\rangle$. (b) $\langle e, u\rangle$ was an endpoint and $c$ was added.

Fig. 2: Cases from Claim 7.

Case 2: At some earlier time, $\langle e', u\rangle$ was an endpoint where $e' = (w, u) \neq e$, and $e = (u, u')$ was added due to $u'$ having a fringe antipode intersection w.r.t. $\langle e', u\rangle$. In this case, we will bound $d(v, V(H)) \leq d(v, w)$; recall $w \in V(H)$ is the earlier occurring vertex of $e'$. See also the figure on the right.




Since $u'$ is antipodal w.r.t. $\langle e', u\rangle$, the angle $\angle e'uu'$ is between $180^o - \alpha$ and $180^o$. Moreover, $u'$ has a fringe intersection with $u$ and $e \in B(u) \cap B(u')$, so the angle $\angle euu'$ is at most $\beta$. Hence $\angle eue'$ is between $180^o - (\alpha + \beta)$ and $180^o$. Again, $\angle euv$ is between $180^o - \alpha$ and $180^o$, since $v$ is in antipodal position with $\langle e, u\rangle$. So $\angle vue'$ is at most $2\alpha + \beta$. Finally, $\angle e'uw$ is at most $\beta$ since $u$ and $w$ have a fringe intersection with $e' \in B(w) \cap B(u)$.
Thus we have $\angle vuw \leq 2\alpha + 2\beta$. By the cosine rule,

$$d(v, w) \leq \sqrt{2^2 + 2^2 - 2 \cdot 2 \cdot 2\cos(2\alpha + 2\beta)}.$$

We need to choose $\rho$, $\alpha$, and $\beta$ so that the following three constraints hold:

1. $2\cos\beta \leq 2 - \rho$
2. $\sqrt{5 + 4\cos\alpha} \leq 3 - \rho$
3. $\sqrt{8 - 8\cos(2\alpha + 2\beta)} \leq 2 - \rho$

Setting $\rho < 0.035$, $\alpha \approx 18.59^o$, and $\beta \approx 10.73^o$ satisfies all three constraints. Thus, the claim holds in all the above cases. ∎

**Lemma 5.** *Any edge cover $\Gamma$ of $G$ corresponds to a $k$-supplier solution of value at most $3 - \rho$.*

*Proof.* Since $\Gamma$ is an edge cover of $G$, it covers clients $V(G)$ within distance one. It is clear that each client in $C$ intersects with some client in $V(G)$. It suffices to show that for each component $H$ and client $v \in C \setminus V(H)$ that intersects some $u \in V(H)$, the distance $d(v, \Gamma) \leq 3 - \rho$.

Let $e \in \Gamma \cap B(u)$ be the edge (facility) in the edge cover $\Gamma$ that is incident to vertex $u \in V(H)$. If $\langle e, u\rangle \notin \{\langle f, x\rangle, \langle g, y\rangle\}$ (at the end of constructing component $H$) then by Claim 7, either $d(e, v) \leq 3 - \rho$ or $d(v, V(H)) \leq 2 - \rho$. So, $d(v, \Gamma) \leq \min\{d(e, v), 1 + d(v, V(H))\} \leq 3 - \rho$.

Now, suppose $\langle e, u\rangle = \langle f, x\rangle$ when the construction of $H$ is complete (the other case of $\langle g, y\rangle$ is identical). We consider the following cases:

Case 1: Client $v$ does not have a fringe antipode intersection w.r.t. $\langle f, x\rangle$. Then, as in the initial cases of Claim 7, $d(v, \Gamma) \leq d(v, f) \leq 3 - \rho$.

Case 2: Client $v$ intersects with some client $u' \in V(H) \setminus \{x, y\}$. Then applying Claim 7 to $u'$ and edge $e' \in \Gamma \cap B(u')$ yields $d(v, \Gamma) \leq 3 - \rho$.

Case 3: If none of the above two cases hold, then we must have $y \neq x$ and $v$ has a non antipode intersection w.r.t. $\langle g, y\rangle$: otherwise, $v$ would have been added to $H$ as a new client. Let $e' \in \Gamma \cap B(y)$, and consider two sub-cases:

- If $e' = g$, then since $v$ has a *non antipodal* intersection w.r.t. $\langle g, y\rangle$, $d(v, \Gamma) \leq d(v, e') = d(v, g) \leq 3 - \rho$.
- If $e' \neq g$, then Claim 7 applies since $e' \in E(H) \setminus \{f, g\}$ and yields $d(v, \Gamma) \leq 3 - \rho$.

In all the cases, we have shown $d(v, \Gamma) \leq 3 - \rho$ which proves the lemma. ■

In the appendix we present details of implementing this algorithm in near-linear time and complete the proof of Theorem 2.

## References


1. S. Arora. Nearly linear time approximation schemes for Euclidean TSP and other geometric problems. In *FOCS*, pages 554–563, 1997.
2. S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *J. ACM*, 45(6):891–923, 1998.
3. T. M. Chan. Approximate nearest neighbor queries revisited. *Discrete & Computational Geometry*, 20(3):359–373, 1998.
4. K. L. Clarkson. An algorithm for approximate closest-point queries. In *Symposium on Computational Geometry (SoCG)*, pages 160–164, 1994.
5. T. Feder and D. H. Greene. Optimal algorithms for approximate clustering. In *STOC*, pages 434–444, 1988.
6. T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.*, 38:293–306, 1985.
7. S. Har-Peled and S. Mazumdar. On coresets for k-means and k-median clustering. In *STOC*, pages 291–300, 2004.
8. D. S. Hochbaum and D. B. Shmoys. A best possible heuristic for the $k$-center problem. *Mathematics of Operations Research*, 10(2):180–184, 1985.
9. D. S. Hochbaum and D. B. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *J. ACM*, 33(3):533–550, 1986.
10. S. G. Kolliopoulos and S. Rao. A nearly linear-time approximation scheme for the Euclidean k-median problem. *SIAM J. Comput.*, 37(3):757–782, 2007.
11. S. Micali and V. V. Vazirani. An $O(\sqrt{V}E)$ Algorithm for Finding Maximum Matching in General Graphs. In *FOCS*, pages 17–27, 1980.

12. M. Mucha and P. Sankowski. Maximum matchings in planar graphs via gaussian elimination. *Algorithmica*, 45(1):3–20, 2006.
13. A. Schrijver. *Combinatorial optimization*. Springer, New York, NY, USA, 2003.
14. P. M. Vaidya. Approximate minimum weight matching on points in k-dimensional space. *Algorithmica*, 4(4):569–583, 1989.
15. P. M. Vaidya. Geometry helps in matching. *SIAM J. Comput.*, 18(6):1201–1225, 1989.
16. R. Yuster and U. Zwick. Maximum matching in graphs with an excluded minor. In *SODA*, pages 108–117, 2007.

## A Implementation Details from Section 3

We will maintain two approximate nearest neighbor (ANN) data structures using Theorem 4: $\mathcal{C}$ containing clients $C$, and $\mathcal{F}$ containing facilities $F$. Our algorithm ensures the property that each client (resp. facility) is obtained at most once as the result of some ANN query in $\mathcal{C}$ (resp. $\mathcal{F}$). Furthermore, the number of steps associated with each client/facility will be constant.

First we define some simple subroutines that use the ANN algorithm [2].

- "Remove-Intersection" (Algorithm A.1) takes as input client $v$, and removes all clients in $\mathcal{C}$ intersecting with it. In particular, it removes all facilities $f$ in $\mathcal{F}$ within distance 1 of $v$, and all clients in $\mathcal{C}$ within distance 1 of $f$. More points may be removed since we only have access to $(1+\epsilon)$-nearest neighbors; this affects the final approximation ratio only by a $1+\epsilon$ factor.
- "Fringe" (Algorithm A.2) takes input point $v$ and ANN data structure $\mathcal{D}$, and outputs a point in $\mathcal{D}$ at distance between $2\cos\beta - 1$ and $1+\epsilon$, i.e. in the fringe of ball $B(v)$. In the process, it also removes all points within distance $2\cos\beta - 1$ of $v$.
- "Next" (Algorithm A.3) takes input client $c$ and facility $f$, and finds a new client $c'$ and facility $f'$ so that (i) $c$ and $c'$ have a fringe intersection with $f' \in B(c) \cap B(c')$ and (ii) $c'$ is antipodal w.r.t. $\langle f, c\rangle$. Again, this procedure deletes the queried facilities/clients that do not satisfy this criteria.

The algorithm for constructing graph $G$ appears as Algorithm A.4. It can be verified directly that it achieves the construction of component $H$ (and the final graph $G$) as desired. To keep the implementation simple, it differs slightly from our description above: when we remove clients in $C$ that intersect with clients in $V(H)$, we may do it also when there is no facility within the intersection. Also, when a new client is connected by an edge to endpoint $x$ and this client is in antipodal position w.r.t. to $\langle g, y\rangle$ we remove other clients which have fringe antipode intersection w.r.t. $\langle g, y\rangle$. It is easy to see that the removed clients are within distance $3-\rho$ from one of the facilities in the edge cover.

After constructing $G$, we apply the linear time edge-cover algorithm on cactus graphs (recall Lemma 4) to obtain the $k$-supplier solution. Finally, we perform binary search over the parameter $L$, that incurs an additional log-factor in the running time. We also lose an additional $1+\epsilon$ factor due to the ANN algorithm. This completes the proof of Theorem 2.

**Algorithm A.1** Remove-Intersection$(v, \mathcal{C}, \mathcal{F})$

set $f \leftarrow$ approximate nearest neighbor of $v$ in $\mathcal{F}$.
**while** $d(v, f) \leq 1 + \epsilon$ **do**
    set $w \leftarrow$ approximate nearest neighbor of $f$ in $\mathcal{C}$.
    **while** $d(f, w) \leq 1 + \epsilon$ **do**
        remove $w$ from $\mathcal{C}$.
        set $w \leftarrow$ approximate nearest neighbor of $f$ in $\mathcal{C}$.
    remove $f$ from $\mathcal{F}$.
    set $f \leftarrow$ approximate nearest neighbor of $v$ in $\mathcal{F}$.

**Algorithm A.2** Fringe$(v, \mathcal{D})$

set $w \leftarrow$ approximate nearest neighbor of $v$ in $\mathcal{D}$.
**while** $d(v, w) \leq 2\cos\beta - 1$ **do**
    remove $w$ from $\mathcal{D}$.
    set $w \leftarrow$ approximate nearest neighbor of $v$ in $\mathcal{D}$.
**if** $d(v, w) > 1 + \epsilon$ **then**
    return NIL.
**else**
    return $w$.

**Algorithm A.3** Next$(f, c, \mathcal{C}, \mathcal{F})$

set $f' \leftarrow$ Fringe$(c, \mathcal{F})$.
**while** $f' \neq$ NIL **do**
    $c' \leftarrow$ Fringe$(f', \mathcal{C})$.
    **while** $(c' \neq$ NIL$)$ and
    $(d(c, c') \notin [2\cos\beta, 2]$ or $(f \neq$ NIL and $c'$ not antipodal w.r.t. $\langle f, c\rangle))$ **do**
        remove $c'$ from $\mathcal{C}$.
        set $c' \leftarrow$ Fringe$(f', \mathcal{C})$.
    **if** $c' \neq$ NIL **then**
        return $\langle f', c'\rangle$.
    **else**
        remove $f'$ from $\mathcal{F}$.
        set $f' \leftarrow$ Fringe$(c, \mathcal{F})$.
return $\langle$NIL, NIL$\rangle$.

**Algorithm A.4** Nearly Linear Algorithm for Euclidean $k$-supplier

construct data structure $\mathcal{C}$ containing points $C$, and $\mathcal{F}$ containing $F$.
**while** $\mathcal{C} \neq \emptyset$ **do**
  pick any $x \in \mathcal{C}$.
  $\langle f, y\rangle \leftarrow$ Next(NIL, $x, \mathcal{C}, \mathcal{F}$).
  **if** $f =$ NIL **then**
    add singleton component $\{x\}$ to $G$, and go to step 2.
      *//computing $f \in F$ that corresponds to the self loop $(x, x)$ is simple*
  set $g \leftarrow f$, component $H$ has $V(H) = \{x, y\}$ and edge $(x, y)$ labeled $f$.
    *//endpoints $\langle x, f\rangle$ and $\langle y, f\rangle$ get initialized.*
  **while** $|V(H)|$ increases **do**
    $\langle f', c'\rangle \leftarrow$ Next$(f, x, \mathcal{C}, \mathcal{F})$.
    **if** $f' =$ NIL **then**
      $\langle g', b'\rangle \leftarrow$ Next$(g, y, \mathcal{C}, \mathcal{F})$.  *//endpoint $\langle x, f\rangle$ does not extend.*
      run Remove-Intersection$(y, \mathcal{C}, \mathcal{F})$.
      **if** $g' \neq$ NIL **then**
        add $b'$ to $V(H)$ and edge $(y, b')$ labeled $g'$.
          *//new client $b'$ intersects only $y$.*
        $y \leftarrow b'$ and $g \leftarrow g'$.
    **else if** $x \neq y$ and
      $d(c', y) \leq 2\cos\beta$ or ($d(c', y) \leq 2$ but $c'$ not antipode w.r.t. $\langle g, y\rangle$) **then**
      remove $c'$ from $\mathcal{C}$, and go to step 9.
        *//$c'$ has (potential) intersection with $y$ that is not fringe antipode.*
    **else**
      add $c'$ to $V(H)$ and edge $(x, c')$ labeled $f'$.  *//new client $c'$ intersects $x$.*
      run Remove-Intersection$(x, \mathcal{C}, \mathcal{F})$.
      $f \leftarrow f'$ and $x \leftarrow c'$.
      **if** $x \neq y$ and $d(c', y) \leq 2$ **then**
        $g'' \leftarrow$ Fringe$(y, \mathcal{F})$.
          *//$c'$ has (potential) fringe antipode intersection with $y$.*
        **while** $g'' \neq$ NIL and $d(g'', c') > 1$ **do**
          remove $g''$ from $\mathcal{F}$.
          $g'' \leftarrow$ Fringe$(y, \mathcal{F})$.
        run Remove-Intersection$(y, \mathcal{C}, \mathcal{F})$.
        **if** $g'' \neq$ NIL **then**
          add edge $(y, c')$ labeled $g''$.  *//new client $c'$ intersects both $x$ and $y$.*
          $y \leftarrow c'$ and $g \leftarrow g''$.
  add component $H$ to $G$.