Mathematics of Operations Research

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

The Euclidean k-Supplier Problem

Viswanath Nagarajan, Baruch Schieber, Hadas Shachnai

To cite this article:
Viswanath Nagarajan, Baruch Schieber, Hadas Shachnai (2020) The Euclidean k-Supplier Problem. Mathematics of Operations Research 45(1):1-14. https://doi.org/10.1287/moor.2018.0953

http://pubsonline.informs.org/journal/moor

MATHEMATICS OF OPERATIONS RESEARCH
Vol. 45, No. 1, February 2020, pp. 1–14
ISSN 0364-765X (print), ISSN 1526-5471 (online)

The Euclidean *k*-Supplier Problem

Viswanath Nagarajan,[a] Baruch Schieber,[b] Hadas Shachnai[c]

[a] Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, Michigan 48109; [b] IBM T.J. Watson Research Center, Yorktown Heights, New York 10598; [c] Computer Science Department, Technion, Haifa, Israel 3200003

Contact: viswa@umich.edu, http://orcid.org/0000-0002-9514-5581 (VN); sbar@us.ibm.com (BS); hadas@cs.technion.ac.il (HS)

Received: June 19, 2017
Revised: February 11, 2018
Accepted: June 10, 2018
Published Online in Articles in Advance: August 2, 2019

MSC2000 Subject Classification: 68W25, 90C27
OR/MS Subject Classification: Primary: networks/graphs: heuristics; secondary: analysis of algorithms: suboptimal algorithms

https://doi.org/10.1287/moor.20180953

Abstract. The *k*-supplier problem is a fundamental location problem that involves opening *k* facilities to minimize the maximum distance of any client to an open facility. We consider the *k*-supplier problem in Euclidean metrics (of arbitrary dimension) and present an algorithm with approximation ratio $1+\sqrt{3}<2.74$. This improves upon the previously known 3-approximation algorithm, which also holds for general metrics. Our result is almost best possible as the Euclidean *k*-supplier problem is NP-hard to approximate better than a factor of $\sqrt{7}>2.64$. We also present a nearly linear time algorithm for the Euclidean *k*-supplier in constant dimensions that achieves an approximation ratio better than three.

Funding: H. Shachnai's work was partially supported by the Israel Science Foundation [Grant 1574/10] and by funding for Center for Discrete Mathematics and Theoretical Computer Science DIMACS visitors. V. Nagarajan's work was partially supported by the National Science Foundation [Grant CAREER CCF-1750127].

Keywords: approximation algorithms • facility location

1. Introduction

Location problems are an important class of combinatorial optimization problems that arise in a number of applications, for example, choosing facility sites in a supply chain (Daskin et al. [9]), placing servers in a telecommunication network (Baev et al. [4]), and clustering data (Kanungo et al. [15]). Moreover, the underlying distance function in many cases is Euclidean (ℓ_2 distance). In this paper, we study a basic location problem on Euclidean metrics.

The *k*-supplier problem consists of *n* points that are partitioned into a client set *C* and a facility set *F*. The points are located in a metric with distance function $d:(C\cup F)\times(C\cup F)\to\mathbb{R}_+$. The distances satisfy two properties: symmetry (i.e., $d(v,u)=d(u,v)$ for all u,v) and triangle inequality (i.e., $d(u,v)+d(v,w)\geq d(u,w)$ for all u,v,w). 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 open facility. Formally,

$$\min_{S\subseteq F:|S|=k}\ \max_{v\in C} d(v,S),$$

where $d(v,S)=\min_{u\in S} d(v,u)$ is the minimum distance of client v to any facility in S.

In the *Euclidean k*-supplier problem, the points $C\cup F$ are in *p*-dimensional space, and the distance function is given by Euclidean (ℓ_2) distances. That is, for any pair $x,y\in\mathbb{R}^p$ of points, the distance between them is $d(x,y)=\sqrt{\sum_{i=1}^{p}(x_i-y_i)^2}$. The dimension p can be arbitrary.

The *k*-supplier problem is well understood on general metrics. There is a 3-approximation algorithm, which is also the best possible (assuming $P\neq NP$) (Hochbaum and Shmoys [14]). The *k*-center problem is the special case of *k*-supplier when the client and facility sets are identical. For *k*-center, there is a better 2-approximation algorithm (Gonzalez [11], Hochbaum and Shmoys [13]), which is also the best possible (assuming $P\neq NP$). Because of the fundamental nature of these problems, many variants and generalizations have also been studied, for example, Panigrahy and Vishwanathan [21], Khuller and Sussmann [16], Cygan et al. [8], An et al. [1], and Chen et al. [6].

Both *k*-supplier and *k*-center have also been studied on Euclidean metrics. In this setting, it is NP-hard to approximate *k*-supplier better than $\sqrt{7}$ and *k*-center better than $\sqrt{3}$ (Feder and Greene [10]). Still, even on two-dimensional Euclidean metrics, the best-known approximation ratios have remained three for *k*-supplier and two for *k*-center.

Our first main result is 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 p with running time* $O(pn^2)$.

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; see, for example, the traveling salesman problem (Arora [2]), k-median (Har-Peled and Mazumdar [12], Kolliopoulos and Rao [17]), or matching Vaidya [25, 24], and Arora [2]. All these papers consider the setting of low constant dimension, which is also very relevant in practice. The running time of these algorithms is typically exponential in the dimension but (near) linear in the input size.

For the Euclidean k-supplier problem in constant dimension, Feder and Greene [10] gave a nearly linear $O(n \log k)$-time 3-approximation algorithm. In contrast, the best running time to obtain a 3-approximation algorithm in general metrics is quadratic $O(nk)$ (Gonzalez [11], Hochbaum and Shmoys [14]). Our second main result is a nearly linear time algorithm for Euclidean k-supplier having an approximation ratio better than three.

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* ≈ 2.965.

Both of our algorithms extend easily to the *knapsack* k-supplier problem, in which facilities have weights $\{w_f : f \in F\}$, and the goal is to open a set of facilities having total weight at most k. We note that handling facility weights in other related location problems, such as k-median, has required a lot of additional effort (Krishnaswamy et al. [18], Swamy [23]).

It is worth noting that it remains NP-hard to approximate the k-supplier problem better than three if we use ℓ_1 or ℓ_∞ distances even in two-dimensional space (Feder and Greene [10]). Thus, our algorithms need to rely heavily on Euclidean metric properties.

1.1. Our Techniques and Outline

The $(1+\sqrt{3})$-approximation algorithm (Theorem 1) is based on a relation to the *minimum edge-cover* problem. Recall that the edge-cover problem (Schrijver [22]) involves computing a subset of edges in an undirected graph so that every vertex is incident to some chosen edge. It is well known that this problem is equivalent to maximum cardinality matching.

The algorithm first performs a standard step of "guessing" the optimal value opt. Then it constructs a maximal "net" $P \subseteq C$ subset of clients whose pairwise distance is more than $\sqrt{3} \cdot \mathsf{opt}$. Next, the algorithm finds a solution $S \subseteq F$ for only the clients in P; the distances to other clients can then be bounded by triangle inequality (this is also a standard approach). The new aspect in our algorithm is the way solution S is found, which involves computing the minimum edge cover in an auxiliary graph with vertices P and an edge for each pair $u, v \in P$ of clients that are both within distance opt from *some* facility.

The key property (which relies on the Euclidean metric) is that any facility can "cover" (within distance 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 prior algorithms (Gonzalez [11], Hochbaum and Shmoys [14, 13]) is that our algorithm uses information on *pairs* of clients that can be covered by a single facility.

This algorithm can be implemented in $O(pn^2)$ time. We can solve the edge-cover instance using the fastest known matching algorithm (Micali and Vazirani [19]), which runs in time $O(|E_G|\sqrt{|V_G|})$, where E_G and V_G are the sets of edges and vertices in the auxiliary graph. In our setting, this leads to an $O(n^{1.5})$ runtime. However, the algorithm to construct the auxiliary graph takes $O(pn^2)$ time,[1] which results in an overall runtime of $O(pn^2)$. These results appear in Section 2.

To obtain the near-linear running time in Theorem 2 (for constant dimension p) we need some more ideas. First, we make use of *approximate nearest neighbor* (ANN) data structures and algorithms (Clarkson [7], Arya et al. [3], Chan [5]) in constructing the auxiliary graph. Given a set $\mathcal{D}$ of n points and any new point q, the nearest-neighbor query involves finding $\arg\min_{x \in \mathcal{D}} d(q, x)$. An approximate nearest-neighbor query settles for an approximate minimizer. The ANN results (Clarkson [7], Arya et al. [3], Chan [5]) imply that 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 the auxiliary graph and, hence, an $\tilde{O}(n^{1.5})$-time implementation of Theorem 1 (in constant dimensions) at the loss of an additional $1+\epsilon$ factor in the approximation ratio (see Section 2.1).

The second idea behind Theorem 2 is to bypass the need for a general matching algorithm. We note that the auxiliary graph G can be quite complex even in constant dimensions: see the end of Section 2 for a more detailed discussion. Using some ideas from Theorem 1, we show that the k-supplier problem can be reduced to

edge cover on *cactus graphs*. This results in a worse approximation ratio of 2.965 (compared with Theorem 1). Note however that this ratio is still better than the previous best bound of three. Crucially, edge cover (and matching) on cactus graphs can be solved in linear time. So the overall running time is now 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 (Arya et al. [3]). The details appear in Section 3.

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

For any instance of the k-supplier problem, it is clear that the optimal value is one of the $|F|\cdot|C|$ distances between clients and facilities. As with most bottleneck-optimization problems (Hochbaum and Shmoys [14]), our algorithm 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;
(2) a k-supplier solution of value at most $\alpha\cdot L$.

As such, $\alpha = 1+\sqrt{3}$ is the approximation ratio. The final algorithm then uses binary search to find the smallest value of L for which a solution is found.

The algorithm is described formally as Algorithm 1 and consists of three main steps. In the first step, we select a maximal subset P of clients in which each pairwise distance is more than $\sqrt{3}L$. In the second step, we construct a graph with vertex-set P in which each edge corresponds to some facility that is within distance L of either a pair of clients (in which case the edge connects these two clients) or a single client (in which case the edge is a self-loop). Finally, the third step computes the minimum edge cover on graph G. See Figure 1 for an example. If the edge-cover value is larger than k, then we obtain a proof that the optimal k-supplier value is more than L, and otherwise, we obtain the desired approximate solution. We now prove the correctness of this algorithm. 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 some facility $f\in F$ with three clients $c_1,c_2,c_3\in P$ having $d(f,c_i)\le L$ for $i\in\{1,2,3\}$. Consider now the plane containing c_1,c_2 and c_3 (which need not contain f). By taking the projection f' of f onto this plane, we obtain a circle centered at f' of radius at most L that has c_1, c_2, and c_3 in its interior (see Figure 2(a)). Hence, the minimum pairwise distance in $\{c_i\}_{i=1}^3$ is at most $\sqrt{3}\cdot L$. This contradicts the fact that every pairwise distance between vertices in P is greater than $\sqrt{3}\cdot L$, which holds by the construction of P (see Equation (1)). The lemma now follows. □

Algorithm 1 (Algorithm for Euclidean k-Supplier)

(1) Pick a maximal subset $P\subseteq C$ such that each pairwise distance in P is more than $\sqrt{3}\cdot L$, that is,

$$\min_{u,v\in P,\, u\neq v} d(u,v) > \sqrt{3}\cdot L. \tag{1}$$

(2) Construct graph $G=(P,E)$ with vertex set P and edge set $E=E_1\cup E_2$, where

$$E_1=\{(u,u): u\in P,\ \exists f\in F \text{ with } d(u,f)\le L \text{ and } d(v,f)>L \text{ for all } v\in P\setminus\{u\}\}; \tag{2}$$

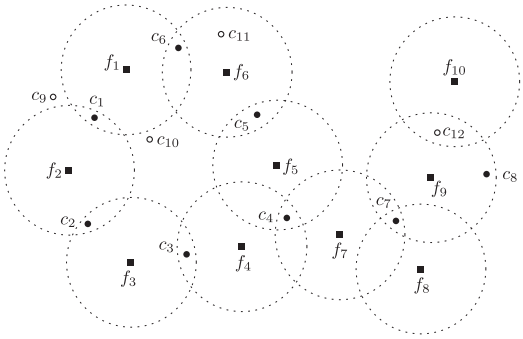
$$E_2=\{(u,v): u,v\in P,\ \exists f\in F \text{ with } d(u,f)\le L \text{ and } d(v,f)\le L\}. \tag{3}$$

(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 Γ as solution.
(8) **End if**

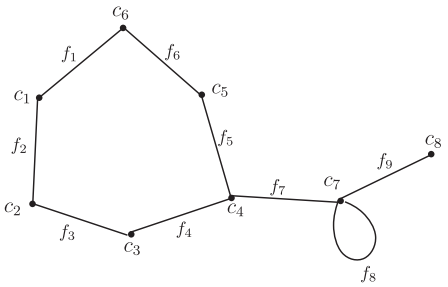
Lemma 1 implies that the facilities F can be partitioned into three sets: for each $j\in\{0,1,2\}$ subset F_j consists of the facilities $f\in F$ with exactly j clients in P within distance L from f. Let $F'=F_1\cup F_2$. Note that $F'=\{f\in F:\exists u\in P \text{ with } d(u,f)\le L\}$. By definition of the edges E_1 in Equation (2), it follows that there is a one-to-one correspondence between the edges E_1 and facilities F_1. Similarly, by Equation (3), there is a one-to-one correspondence between edges E_2 and facilities F_2. Clearly, if the optimal k-supplier value is at most L, then there is a subset $F''\subseteq F'$ of at most k facilities such that each client in P lies within distance L of some facility of F''. In other words,

Claim 1. *If the optimal k-supplier value is at most L, then graph $G=(P,E)$ in step 2 contains an edge cover of size at most k.*

Figure 1. An instance of k-supplier (left) and its graph G (right) as used in Algorithm 1.



There are 12 clients $\{c_i\}_{i=1}^{12}$ and 10 facilities $\{f_j\}_{j=1}^{10}$
The radius L balls around each facility are also shown



The set P consists of clients $\{c_i\}_{i=1}^{8}$
Each edge is labeled by its corresponding facility

Recall that an edge cover in an undirected graph is a subset of edges in which 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, for example, Schrijver [22]. By Claim 1, if the minimum edge cover Γ 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 Γ has size at most k, then (in step 7) we output the corresponding facilities (from F') as the solution.

We can now prove the performance guarantee:

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

Proof. To reduce notation, we use Γ to denote both the edge cover in G and its corresponding facilities from F'. Because Γ is an edge cover in G, each client $u \in P$ is within distance L of some facility in Γ. That is,

$$\max_{u \in P} d(u, \Gamma) \quad \leq \quad L. \tag{4}$$

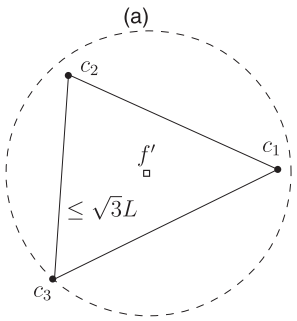
Now, because $P \subseteq C$ is a maximal subset satisfying the condition in step 1, for each client $v \in C \setminus P$ there is some $u \in P$ with $d(u,v) \leq \sqrt{3}L$. Using Equation (4) and triangle inequality, it follows that $\max_{v \in C} d(v, \Gamma) \leq (\sqrt{3}+1)L$. □

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

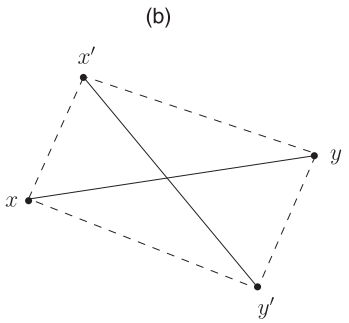
2.1. Knapsack *k*-Supplier Problem

Our algorithm extends easily to the knapsack k-supplier problem, in which 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 (Equations (2) and (3)), we also include weights of the respective facilities. Then we find a *minimum weight* edge cover Γ, which can also be done in polynomial time (Schrijver [22]).

Figure 2. Examples for (a) Lemma 1 and (b) Lemma 4.



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



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

2.2. 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 $G = (V_G, E_G)$. The edge-cover problem can be solved via a matching algorithm (Micali and Vazirani [19], Schrijver [22]) that runs in time $O(|E_G|\sqrt{|V_G|})$, where $|E_G|$ is the number of edges and $|V_G|$ is the number of vertices in G. We have $|V_G| \leq |C|$ as each vertex is a client and $|E_G| \leq |F|$ as each edge corresponds to a facility. So the runtime for finding the edge cover is $O(n^{3/2})$. This proves the runtime in Theorem 1.

In the rest of this section, we focus on the case in which the dimension p is constant. We obtain implementations with better running times while incurring a small loss in the approximation ratio. In particular, for any constant $\epsilon > 0$, our algorithm will output one of the following:

(1) a certificate showing that the optimal k-supplier value is more than L;
(2) a k-supplier solution of value at most $(1+\epsilon)^2(1+\sqrt{3})\cdot L$.

Recall that there are two main parts in our algorithm: constructing the graph G and solving the edge-cover problem on G. We discuss the implementation details for each of these parts separately.

In constructing graph G, a naïve implementation of this step results in an $O(n^2)$ running time as described. We now show that the runtime can be significantly improved. The main component here is a fast data structure for approximate nearest neighbor search.

Theorem 3 (Arya et al. [3]). *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 ℓ many $(1+\epsilon)$ approximate nearest neighbors of q can be computed in $O((c+\ell p)\log n)$ time.*
- *Any new point can be inserted in $O(\log n)$ time.*
- *Any point can be deleted in $O(\log n)$ time.*

We maintain such a data structure $\mathcal{P}$ for the clients. First, we implement the step of finding a maximal "net" $P \subseteq C$ in Algorithm 2.

Algorithm 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** $d(v, v') > \sqrt{3}(1+\epsilon)^2 L$ **or** $v' =$ NIL**, then**
(5) $\quad\quad P \leftarrow P \cup \{v\}$ and insert v into $\mathcal{P}$.
(6) $\quad$ **End if**
(7) **End for**
(8) output P.

Because we use $(1+\epsilon)$ approximate distances, the condition in step 4 ensures that every pairwise distance in the final set P is more than $(1+\epsilon)\sqrt{3}L$. Moreover, for each $u \in C \setminus P$, there is some $v \in P$ satisfying $d(u,v) \leq \sqrt{3}(1+\epsilon)^2 L$. By Theorem 3, the time taken for each insertion and nearest-neighbor query in $\mathcal{P}$ is $O(\log n)$, so the total running time of Algorithm 2 is $O(n\log n)$.

Algorithm 3 (Algorithm for Computing Edges E of G)

(1) Construct data structure $\mathcal{P}$ containing points P, and initialize $E \leftarrow \emptyset$.
(2) **For** $f \in F$**, do**
(3) $\quad u \leftarrow$ approximate nearest neighbor of f in $\mathcal{P}$.
(4) $\quad v \leftarrow$ approximate second nearest neighbor of f in $\mathcal{P}$.
(5) $\quad$ **If** $d(u,f) \leq (1+\epsilon)L$ **and** $d(v,f) > (1+\epsilon)L$**, then**
(6) $\quad\quad$ set $E \leftarrow E \cup \{(u,u)\}$ and label edge (u,u) by facility f.
(7) $\quad$ **End if**
(8) $\quad$ **If** $d(u,f) \leq (1+\epsilon)L$ **and** $d(v,f) \leq (1+\epsilon)L$**, then**
(9) $\quad\quad$ set $E \leftarrow E \cup \{(u,v)\}$ and label edge (u,v) by facility f.
(10) $\quad$ **End if**
(11) **End for**
(12) Output E.

Algorithm 3 shows how to compute the edge set E. Because all pairwise distances in P are larger than $(1+\epsilon)\sqrt{3}L$, Lemma 1 implies that each facility $f \in F$ is within distance $(1+\epsilon)L$ 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. The condition for adding edges ensures that there is an edge in E for every facility in the set $F' = \{f \in F : \exists u \in P \text{ with } d(u, f) \le L\}$; this is because the approximate nearest neighbor of any facility $f \in F'$ has distance at most $(1+\epsilon)L$ from f. Moreover, if there is any facility $f \in F$ and client $u \in C$ with $d(f, u) \le L$, then the edge in E labeled f must be incident to u. Therefore, Claim 1 continues to hold. Note that there may be additional edges in E corresponding to facilities $f \in F \setminus F'$ with nearest neighbor in P at distance between L and $(1+\epsilon)L$. Finally, note that the following weaker version of Lemma 2 holds.

Lemma 3. *If graph G has an edge cover Γ with $|\Gamma| \le k$, then Γ corresponds to a k-supplier solution of value at most $(1+\epsilon)^2(1+\sqrt{3})L$.*

Proof. We only sketch the proof as it is very similar to Lemma 2. Because Γ is an edge cover in G, each client $u \in P$ is within distance $(1+\epsilon)L$ of Γ. Note that each client $v \in C \setminus P$ has some $u \in P$ with $d(u, v) \le (1+\epsilon)^2\sqrt{3}L$. So each $v \in C \setminus P$ is within distance $(1+\epsilon)^2\sqrt{3}L + (1+\epsilon)L$ of Γ. □

By Theorem 3, the time for each two nearest neighbors query is $O(c \log n)$. Thus, the total time in Algorithm 3 is $O(cn \log n)$, which is $O(n \log n)$ for any constant dimension p.

For computing edge cover on $\mathbf{G}$, as noted in the beginning of Section 2.1, using the fastest algorithm (Micali and Vazirani [19]) for matching on general graphs results in an $O(n^{3/2})$ running time.

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

Lemma 4. *If dimension is $p = 2$ and $\epsilon \le 0.2$, 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 2(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 $\epsilon \le 0.2$), for any edge (u, v) and vertex $w \in P$, the angle uwv is strictly less than 90°. 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°. Similarly, for edge (x', y') and points x and y, angles $x'xy'$ and $x'yy'$ are also strictly smaller than 90°. This contradicts with the fact that the sum of interior angles of quadrilateral $xx'yy'$ must equal 360°. □

Based on this lemma, we can use the faster $O(n^{\omega/2})$-time randomized algorithm for matching on planar graphs because of Mucha and Sankowski [20]. Here, $\omega < 2.38$ is the matrix multiplication exponent. Thus, we have shown the following:

Theorem 4. *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 $O(n^{1.5} \log n)$ time for any constant dimension p and $O(n^{1.17} \log n)$ time for $p = 2$ dimensions.*

The additional $\log n$ factor comes from the binary search that we perform over the parameter L.

Remark 1. We are not aware of any additional properties of G that are useful in improving the running time of Algorithm 1. In particular,

- Any degree three planar graph can be obtained as graph G for some instance of two-dimensional Euclidean k-supplier, and the fastest known matching algorithm even on this family of graphs still runs in $O(n^{1.17})$ time (Mucha and Sankowski [20]). Indeed, there is a linear time reduction (Mucha and Sankowski [20]) from matching on general planar graphs to degree three planar graphs.
- Even in three dimensions, there are instances of k-supplier in which the graph G (for certain values of L) does not exclude any fixed minor. So, for higher constant dimensions, we cannot rely on matching algorithms on excluded-minor graphs and need to use a general matching algorithm.

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 $1+\sqrt{3}$ bound from the previous section but still better than the previous best three-approximation algorithm (Feder and Greene [10]). The algorithm here uses some ideas from the previous reduction to an edge-cover problem. However, to achieve near-linear running time, we can not solve a general edge-cover problem. Instead, we show that, using additional Euclidean properties, one can reduce to an edge-cover problem on a special class of cactus graphs. This approach gives a nearly linear time algorithm because edge cover on cactus graphs can be solved in linear time.

In Section 3.1, we present the details of our algorithm and its approximation ratio without running time details. Then, in Section 3.2, we present the implementation details using approximate nearest neighbors (Theorem 3) and establish the running time bound.

3.1. Algorithmic Details

As in Section 2, the algorithm here also uses the approach of guessing a bound L on the optimal value and verifying it approximately. To reduce notation, throughout this section, we scale distances so that this parameter $L = 1$. We then show that the algorithm either finds a solution of value at most 2.965 or proves that the optimal value is more than one.

Let $0 < \rho < 1$ be some constant and $0° < \alpha, \beta < 30°$ be angles, the values of which will be set later. For any point v, we denote the *ball* of radius one centered at v by $B(v) := \{u \in C \cup F : d(v,u) \le 1\}$. We may assume (without loss of generality) that $B(v) \cap F \neq \emptyset$ for all clients $v \in C$: otherwise, we would immediately obtain a proof that the optimal value is more than one.

As in the previous section, the algorithm here builds a graph G with vertices corresponding to clients and edges corresponding to facilities. However, this procedure is more complex because we want the resulting graph to have a much simpler structure (so that the edge-cover problem on G can be solved in linear time). We now start with some useful definitions.

Definition 1 (Interference). Two clients $c, c' \in C$ are said to *interfere* if

- the distance between c and c' is at most two, that is, $d(c,c') \le 2$;
- there is some facility in $B(c) \cap B(c')$, that is, $F \cap B(c) \cap B(c') \neq \emptyset$.

We emphasize that clients c and c' interfere only if there is some *facility* in the intersection $B(c) \cap B(c')$ of their balls. So we can have clients c, c' with $B(c) \cap B(c') \neq \emptyset$ that do not interfere.

Definition 2 (Fringe Interference). Two clients $c, c' \in C$ are said to have a *fringe interference* if they interfere and $d(c,c') > 2\cos\beta$.

Note that when c and c' have a fringe interference, for any point $v \in B(c) \cap B(c')$ the angles $\angle vcc'$ and $\angle vc'c$ are at most β. See Figure 3(a) for an example.

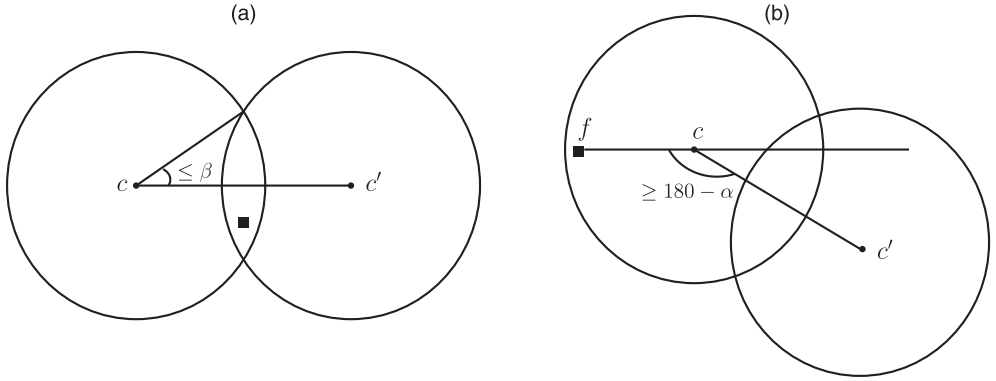
Definition 3 (Antipode). Given a client c and facility $f \in B(c)$, we say that another client c' is in *antipodal position* with respect to (w.r.t.) $\langle f, c\rangle$ if the angle fcc' is more than $180° - \alpha$. If, in addition, c and c' have a fringe interference, we say that c' has a *fringe antipode interference* with $\langle f, c\rangle$.

See Figure 3(b) for an example.

3.1.1. Constructing Graph *G*.

The graph G is constructed iteratively; 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 interference, 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 pair of clients with a fringe interference, then we pick an arbitrary client c_0, set $V(H) = \{c_0\}$ and $E(H) = \{(c_0, c_0)\}$ (i.e., H is a singleton component with a self-loop), remove all clients interfering with c_0, and continue with the next iteration.[2]

Figure 3. Examples of fringe interference and antipode.



c and c' have a fringe interference

c' is in antipodal position w.r.t. $\langle f, c\rangle$

Component H is now constructed incrementally, adding one vertex in each step. We maintain two "end 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 also refer to the tuples $\langle x, f\rangle$ and $\langle y, g\rangle$ as *end points*. Initially, we set $x \leftarrow c_1$, $f \leftarrow f_0$ and $y \leftarrow c_2$, $g \leftarrow f_0$.

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

P1. c does not interfere with any client in $V(H)\setminus\{x, y\}$.

P2. c has a fringe antipode interference with either $\langle x, f\rangle$ or $\langle y, g\rangle$.

P3. If $x \neq y$ and c interferes with both x and y, then c must be fringe antipode w.r.t. both $\langle x, f\rangle$ and $\langle y, g\rangle$.

For any client c that satisfies these three conditions, we update H as follows:

Case 1. Suppose $x \neq y$ and client c interferes with both x and y. Let $f' \in B(x) \cap B(c)$ denote a facility in the fringe interference of x and c. Similarly, let $g' \in B(y) \cap B(c)$ denote a facility in the fringe interference of y and c. We now add vertex c to $V(H)$ and edges (c, x) labeled f' and (c, y) labeled g'. Also update $x \leftarrow c$, $f \leftarrow f'$, $y \leftarrow c$, and $g \leftarrow g'$.

Case 2. Suppose $x \neq y$ and client c interferes with exactly one of $\{x, y\}$, say x (the other case of y is identical). Let $f' \in B(x) \cap B(c)$ denote a facility in the fringe interference of x and c. Now, add vertex c to $V(H)$ and an edge (x, c) labeled f'. Also update $x \leftarrow c$ and $f \leftarrow f'$.

Case 3. Suppose $x = y$ and client c has a fringe antipode interference with $\langle x, f\rangle$ (the other case of $\langle y, g\rangle$ is identical). Then, we perform the same steps as in case 2.

The construction of component H ends when there is no new client c (satisfying these conditions) that can be added. Let $E(H)$ denote the edges in component H. At this point, we remove all clients that interfere with any client in $V(H)$: such clients will eventually be covered by a subset of facilities in $E(H)$. Then we iterate by building the next component of G.

3.1.2. Solving Edge Cover on *G*.

Next, we prove a useful property of the graph G.

Definition 4 (Linear Cactus). A graph H is called a *linear cactus* if it contains distinct "portal" vertices $P = \{v_1, \cdots v_r\}$, cycles $C_1, \cdots, C_{r-1}$, and path C_r (possibly empty) such that

- $H = \cup_{i=1}^{r} C_i$ is the union of these cycles and path;
- $\{C_i\}_{i=1}^{r}$ are vertex-disjoint except at the portals, that is, $C_i \cap C_j \subseteq P$ for any $1 \le i \neq j \le r$;
- for each $i \in [r-1]$, $C_i \cap P = \{v_i, v_{i+1}\}$, and if $C_r \neq \emptyset$, then $C_r \cap P = \{v_r\}$.

If the path $C_r \neq \emptyset$, then we refer to H as an *open* cactus, and the two end points of path C_r are called the end points of H. If path $C_r = \emptyset$, then we refer to H as a *closed* cactus; in this case, vertex v_r is the only end point of H. See Figure 4 for an example.

Lemma 5. *Each nonsingleton component H in the algorithm of Section 3.1.1 is a linear cactus.*

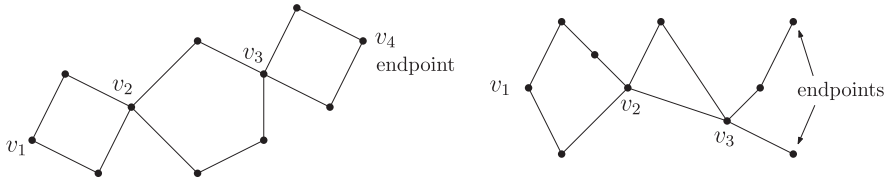
Proof. We proceed by induction on the number of vertices $|V(H)|$. We show that, at any point in its construction, H is a linear cactus with end points x and y. It is possible to have $x = y$, which corresponds to a closed cactus: in this case, we also show that the degree of x is two.

The base case corresponds to $V(H) = \{c_1, c_2\}$, $E(H) = \{(c_1, c_2)\}$, $x = c_1$, and $y = c_2$, which is a linear cactus with $r = 1$. (The case $H = \{c_0\}$ is trivial.)

For the inductive step, we consider the two types of updates.

- Case 1: H gets a new vertex c and two new edges (c, x) and (c, y). In this case, we must have $x \neq y$. Here, H changes from an open cactus to a closed cactus with end point c (which is also the new value of x and y). Note also that the degree of c is two as needed.
- Case 2: $x \neq y$ and H gets a new vertex c and one new edge (c, x). Here, H remains an open cactus: its end points are y and c (which is the new value of x).

Figure 4. Examples of linear cactus.



Closed linear cactus ($r = 4$) Open linear cactus ($r = 3$)

- Case 3: $x = y$ and H gets a new vertex c and one new edge (c, x). By the inductive hypothesis, we know that x has degree two before the addition of edge (c, x). So after adding edge (c, x) graph H is still a linear cactus. Note that H changes to an open cactus with end points y and c (which is the new value of x).

Thus, in all cases, H is a linear cactus with the desired properties. □

Theorem 5. *There is a linear time algorithm for edge cover on linear cactus graphs.*

Proof. Consider a linear cactus as in Definition 4. We provide a simple dynamic program to compute the minimum edge cover. For any $i \in [r]$, let $T[i, 0]$ denote the minimum edge cover in the graph $H_i := C_i \cup C_{i+1} \cdots \cup C_r$ and $T[i, 1]$ the minimum edge cover for graph H_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: this requires $O(|C_r|)$ time.

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 of which can be found in $O(|C_i|)$ time.

- Γ_i^1 (Γ_i^2) contains neither e_{i+1} nor f_{i+1} and covers vertices $C_i \setminus v_{i+1}$ ($C_i \setminus \{v_i, v_{i+1}\}$).
- Γ_i^3 (Γ_i^4) contains e_{i+1} but not f_{i+1} and covers vertices C_i ($C_i \setminus \{v_i\}$).
- Γ_i^5 (Γ_i^6) contains f_{i+1} but not e_{i+1} and covers vertices C_i ($C_i \setminus \{v_i\}$).
- Γ_i^7 (Γ_i^8) contains both f_{i+1} and e_{i+1} and covers vertices C_i ($C_i \setminus \{v_i\}$).

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

$$T[i, 0] := \min \{T[i+1, 0] + \Gamma_i^1,\ T[i+1, 1] + \min \{\Gamma_i^3, \Gamma_i^5, \Gamma_i^7\}\}$$
$$T[i, 1] := \min \{T[i+1, 0] + \Gamma_i^2,\ T[i+1, 1] + \min \{\Gamma_i^4, \Gamma_i^6, \Gamma_i^8\}\}.$$

Clearly, this dynamic program can be solved in linear time. □

We note that this dynamic program easily extends to *weighted* edge cover as well.

3.1.3. Proving the Approximation Ratio. We start with a relation between k-supplier solutions and edge covers in G.

Claim 2. *If the optimal k-supplier value is at most one, graph G has an edge cover of size k.*

Proof. We say that a facility *covers* a client if the distance between them is at most one. Note that if the same facility covers two clients c, c' then c and c' interfere.

We first show that each facility covers at most two clients in $V(G)$. Note that clients in different components of G do not interfere. This is because we remove all clients interfering with $V(H)$ after constructing each component H. So each facility covers clients in at most one component. Now consider the clients in any component H. By the choice of each new new client c, it is clear that all of c's interferences with the current set of vertices $V(H)$ are fringe interferences. Hence, any interference between clients in the final component H is a fringe interference. So, if a facility covers some subset $S \subseteq V(H)$ of clients, then all pairwise distances in S are at least $2 \cos \beta \geq \sqrt{3}$ because $\beta \leq 30°$. And, by Lemma 1, we must have $|S| \leq 2$. So each facility covers at most two clients in $V(G)$. Moreover, there is an edge in graph G between every pair of clients in $V(G)$ that can be covered by a single facility.

Let $F' \subseteq F$ denote the facilities that cover at least one client in $V(G)$. Define (multi)graph $\overline{G}$ on vertices $V(G)$ with an edge corresponding to *each* facility $f \in F'$: if f covers only one client c, then the edge is a self-loop (c, c), and if f covers two clients c, c', then the edge is (c, c'). If the optimal k-supplier value is at most one, then it is clear that $\overline{G}$ has an edge cover of size k.

Clearly, the graph G constructed in our algorithm is a subgraph of $\overline{G}$. For every pair c, c' of distinct clients that are connected by at least one edge in $\overline{G}$, the edge $(c, c') \in E(G)$. Moreover, if c is a singleton connected component in $\overline{G}$ with at least one self-loop (c, c), then we also have $(c, c) \in E(G)$. This implies that G also has an edge cover of size k. □

We now prove that this algorithm achieves an approximation ratio $3 - \rho$. We consider a particular component H. The variables x, f, y, g denote their values at the end of H's construction (unless specified otherwise). To keep notation simple, for any edge $e \in E(H)$, we use e to also refer to the facility labeling it.

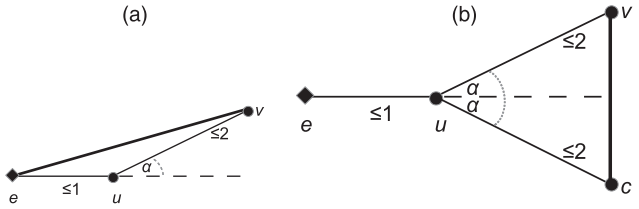
Claim 3. *For any client $u \in V(H)$ and edge (facility) $e = (u, u') \in E(H)$ such that $\langle e, u \rangle \notin \{\langle f, x \rangle, \langle g, y \rangle\}$ and client $v \in C$ that interferes with u, either $d(e, v) \leq 3 - \rho$ or $d(v, V(H)) \leq 2 - \rho$.*

Proof. The claim holds trivially for $v \in V(H)$. So we assume $v \in C \setminus V(H)$. We consider various possible cases.

(1) u and v do not have a fringe interference. Then $d(u, v) \leq 2 \cos \beta$ and

$$d(v, V(H)) \leq d(v, u) \leq 2 \cos \beta. \tag{5}$$

Figure 5. Cases from Claim 3.



(2) u and v have a fringe interference, and v is not in antipodal position w.r.t. $\langle e, u\rangle$. Then, by the cosine rule (see Figure 5(a)),

$$d(e, v) \le \sqrt{1^2 + 2^2 + 2 \cdot 2 \cos \alpha}. \tag{6}$$

(3) u and v have a fringe interference and v is in antipodal position w.r.t. $\langle e, u\rangle$.

- u' was added to H before u. In this case, $\langle e, u\rangle$ became an end point immediately after u was added to H. By the assumption in the claim, $\langle e, u\rangle$ is *not* an end point in the final component H. So it must be that some client c was added to H because of a fringe antipode interference w.r.t. $\langle e, u\rangle$. We now use $d(v, V(H)) \le d(v, c)$. Because both v and c are in antipode position w.r.t. $\langle e, u\rangle$, the angle $\angle vuc$ is at most 2α; see Figure 5(b). Again by the cosine rule,

$$d(v, V(H)) \le d(v, c) \le \sqrt{2^2 + 2^2 - 2 \cdot 2 \cdot 2 \cos 2\alpha}. \tag{7}$$

- u was added to H before u'. Consider the time when u' is added to H. Then, we must have had an end point $\langle e', u\rangle$ where $e' \ne e$ such that u' had a fringe antipode interference with $\langle e', u\rangle$. Let edge $e' = (w, u)$; note that $w \in V(H)$ must have been added before u. In this case, we use $d(v, V(H)) \le d(v, w)$. See also Figure 6. Because u' is antipodal w.r.t. $\langle e', u\rangle$, the angle $\angle e'uu'$ is between $180° - \alpha$ and $180°$. Moreover, u' has a fringe interference with u and $e \in B(u) \cap B(u')$, so the angle $\angle euu'$ is at most β. Hence $\angle eue'$ is between $180° - (\alpha + \beta)$ and $180°$. As v is in antipodal position with $\langle e, u\rangle$, we have $\angle euv$ between $180° - \alpha$ and $180°$. So $\angle vue'$ is at most $2\alpha + \beta$. Finally, $\angle e'uw$ is at most β because u and w have a fringe interference with $e' \in B(w) \cap B(u)$. Thus, we have $\angle vuw \le 2\alpha + 2\beta$. By the cosine rule,

$$d(v, V(H)) \le d(v, w) \le \sqrt{2^2 + 2^2 - 2 \cdot 2 \cdot 2 \cos(2\alpha + 2\beta)}. \tag{8}$$

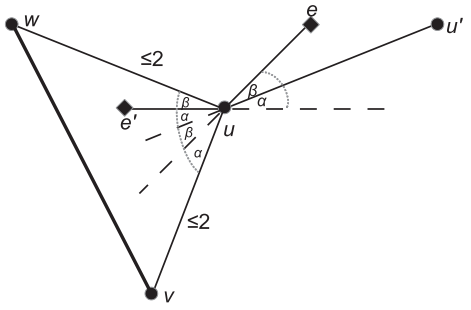
Recall that we need to show $d(e, v) \le 3 - \rho$ or $d(v, V(H)) \le 2 - \rho$. For this condition to hold, using Equations (5)–(8), it suffices to choose values for ρ, α, and β so that

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

Setting $\rho < 0.035$, $\alpha \approx 18.59°$ and $\beta \approx 10.73°$ satisfies all three constraints. Thus, the claim holds in all these cases. □

Lemma 6. *Any edge cover* Γ *of* G *corresponds to a k-supplier solution of value at most* $3 - \rho$.

Figure 6. Cases from Claim 3.



Proof. Because Γ is an edge cover of G, it covers clients $V(G)$ within distance one. It is clear that each client in $C\setminus V(G)$ gets removed because of (interference with) some component H of G. Consider any client $v \in C\setminus V(H)$ that gets removed because of some component H; that is, v interferes with some $u \in V(H)$. It suffices to show that the distance $d(v,\Gamma) \le 3-\rho$.

Let $e \in \Gamma \cap B(u)$ be the edge (facility) in the edge cover Γ that is incident to vertex $u \in V(H)$. If $\langle e,u\rangle \notin \{\langle f,x\rangle, \langle g,y\rangle\}$ (i.e., $\langle e,u\rangle$ is not an end point in the final component H), then by Claim 3, either $d(e,v) \le 3-\rho$ or $d(v,V(H)) \le 2-\rho$. So $d(v,\Gamma) \le \min\{d(e,v), 1+d(v,V(H))\} \le 3-\rho$.

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 possibilities:

Case 1. Client v interferes with with some client $u' \in V(H)\setminus\{x,y\}$. Then applying Claim 3 to client u', edge $e' \in \Gamma \cap B(u')$ and client v, we obtain $d(v,\Gamma) \le 3-\rho$.

Case 2. Client v does not have a fringe antipode interference with $\langle f,x\rangle$. Then, exactly as in cases 1 and 2 of Claim 3, we have $d(v,\Gamma) \le d(v,f) \le 3-\rho$.

Case 3. Client v does not interfere with any $V(H)\setminus\{x,y\}$ and has a fringe antipode interference with $\langle f,x\rangle$. As v was not added to H, it must be that $y \ne x$ and v has a nonantipode interference with $\langle g,y\rangle$. Exactly as in cases 1 and 2 of Claim 3, we have $d(v,V(H)) \le 2-\rho$ or $d(v,g) \le 3-\rho$. Now, let $e' \in \Gamma \cap B(y)$, and consider two more cases:

- $e' = g$. Then, as $g \in \Gamma$, we have $d(v,\Gamma) \le \max\{d(v,g), 1+d(v,V(H))\} \le 3-\rho$.
- $e' \ne g$. Then applying Claim 3 to y, edge e' and v (the claim applies as $\langle e',y\rangle \notin \{\langle f,x\rangle, \langle g,y\rangle\}$), we get $d(v,\Gamma) \le \max\{d(v,e'), 1+d(v,V(H))\} \le 3-\rho$.

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

3.2. Implementation Details

Here, we discuss the details of constructing graph G in near-linear time. We will maintain two ANN data structures using Theorem 3: $\mathscr{C}$ containing clients C and $\mathscr{F}$ containing facilities F.

First, we define some simple subroutines that use the ANN algorithm (Arya et al. [3]).

- RemoveInterference (Algorithm 4) takes as input client v and removes all clients in $\mathscr{C}$ interfering with it. In particular, it removes all facilities f in $\mathscr{F}$ within distance one of v and all clients in $\mathscr{C}$ within distance one of f. More points may be removed because we only have access to $(1+\epsilon)$ nearest neighbors; this affects the final approximation ratio only by a $1+\epsilon$ factor.
- NextPoint (Algorithm 5) takes as input an end point $\langle x,f\rangle$ and outputs two lists L and R. List L contains all client–facility pairs (u,h), where client u has a fringe antipode interference with $\langle x,f\rangle$ and facility $h \in B(u) \cap B(x)$. List R contains all clients u that interfere with x but are not fringe antipode with $\langle x,f\rangle$. We also use $f = \mathsf{NIL}$, in which case all conditions in the algorithm involving f are ignored. This procedure does not make any net change to $\mathscr{C}$ or $\mathscr{F}$.

The algorithm for constructing graph G appears as Algorithm 6. It constructs one component H of G in each iteration of the outer while loop as described in Section 3.1.1. The ANN data structure $\mathscr{C}$ maintains only the clients that do not have an interference with previously added vertices of G: when we identify a client that can never satisfy the conditions P1–P3 (used to construct H), we immediately remove it from $\mathscr{C}$. At any time in the algorithm, $\langle x,f\rangle$ and $\langle y,g\rangle$ denote the end points of component H. For any client $c \in \mathscr{C}$, we maintain X_c (Y_c) to be the facility (if any) that lies in the fringe antipode interference with $\langle x,f\rangle$ ($\langle y,g\rangle$). Any client $c \in \mathscr{C}$ with either $X_c \ne \mathsf{NIL}$ or $Y_c \ne \mathsf{NIL}$ is eligible as the next vertex to be added to H.

Algorithm 4 (RemoveInterference(v))

```
(1)  Set f ← ANN of v in 𝓕.
(2)  While d(v, f) ≤ 1 + ε, do
(3)      set w ← ANN of f in 𝓒.
(4)      While d(f, w) ≤ 1 + ε, do
(5)          remove w from 𝓒.
(6)          Set w ← ANN of f in 𝓒.
(7)      End while
(8)      Remove f from 𝓕.
(9)      Set f ← ANN of v in 𝓕.
(10) End while
```

Algorithm 5 (NextPoint(x, f))

$F_0 \leftarrow \emptyset$, $C_0 \leftarrow \emptyset$, $L \leftarrow \emptyset$, and $R \leftarrow \emptyset$.
Set $h \leftarrow$ ANN of x in $\mathscr{F}$.
While $d(x,h) \le 1+\epsilon$, **do**
  set $u \leftarrow$ ANN of h in $\mathscr{C}$.
  While $d(h,u) \le 1+\epsilon$, **do**
    remove u from $\mathscr{C}$ and $C_0 \leftarrow C_0 \cup \{u\}$.
    If $d(x,u) > 2\cos\beta$ **and** $\angle fxu > 180^o - \alpha$, **then**
      set $L \leftarrow L \cup \{(u,h)\}$.
    Else
      set $R \leftarrow R \cup \{u\}$.
    End if
    Set $u \leftarrow$ ANN of h in $\mathscr{C}$.
  End while
  Remove h from $\mathscr{F}$ and $F_0 \leftarrow F_0 \cup \{h\}$.
  Set $h \leftarrow$ ANN of x in $\mathscr{F}$.
End while
Add all points in C_0 to $\mathscr{C}$.
Add all points in F_0 to $\mathscr{F}$.
Return L, R.

Algorithm 6 (Nearly Linear Time Algorithm for Euclidean k-Supplier)

Construct ANN data structures: $\mathscr{C}$ containing points C and $\mathscr{F}$ containing F.
For each $c \in \mathscr{C}$, set $X_c, Y_c \leftarrow$ NIL.
While $\mathscr{C} \neq \emptyset$, **do**
  component $H \leftarrow \emptyset$.
  Pick any $x \in \mathscr{C}$.
  $(L,R) \leftarrow$ NextPoint $(x,$ NIL$)$.
  If $L = \emptyset$, **then**
    add component $H = \{x\}$ to G.
    RemoveInterference (x).
    Continue while loop.
  End if
  Set $\langle y,f\rangle \leftarrow$ any pair in L and $g \leftarrow f$.
  Add to H: vertices x, y and edge (x,y) with label f.
  $(L_1,R_1) \leftarrow$ NextPoint (x,f) and $(L_2,R_2) \leftarrow$ NextPoint (y,g).
  For each $(u,h) \in L_1$, set $X_u \leftarrow h$.
  For each $(u,h) \in L_2$, set $Y_u \leftarrow h$.
  For each $u \in R_1 \cup R_2$, remove u from $\mathscr{C}$.
  While $\exists\, c \in \mathscr{C}$ **with** $X_c \neq$ NIL **or** $Y_c \neq$ NIL, **do**
    pick any such client c.
    If $X_c \neq$ NIL, $Y_c \neq$ NIL, **and** $x \neq y$, **then**
      add to H: vertex c, edge (x,c) with label X_c, and edge (y,c) with label Y_c.
      RemoveInterference (x) and RemoveInterference (y).
      Update $x, y \leftarrow c$, $f \leftarrow X_c$, and $g \leftarrow Y_c$.
      For each $u \in L_1$, set $X_u \leftarrow$ NIL.
      For each $u \in L_2$, set $Y_u \leftarrow$ NIL.
      $(L_1,R_1) \leftarrow$ NextPoint (x,f) and $(L_2,R_2) \leftarrow$ NextPoint (y,g).
      For each $(u,h) \in L_1$, set $X_u \leftarrow h$.
      For each $(u,h) \in L_2$, set $Y_u \leftarrow h$.
    End if
    If (exactly one of $X_c, Y_c \neq$ NIL**) or** $(x = y)$, **then**
      suppose $X_c \neq$ NIL. (The case $Y_c \neq$ NIL can be handled identically.)
      Add to H: vertex c and edge (x,c) with label X_c.
      If $x \neq y$, then RemoveInterference (x).
      Update $x \leftarrow c$ and $f \leftarrow X_c$.

(35) For each $u \in L_1$, set $X_u \leftarrow$ NIL.
(36) $(L_1, R_1) \leftarrow$ NextPoint (x, f).
(37) For each $(u, h) \in L_1$, $X_u \leftarrow h$.
(38) For each $u \in R_1$, remove u from $\mathscr{C}$.
(39) **End if**
(40) **End while**
(41) Add component H to G.
(42) RemoveInterference (x) and RemoveInterference (y).
(43) **End while**

In Algorithm 6 lines 4–17 initialize H to either a singleton-vertex component (in which case the iteration ends) or with two clients x and y with a fringe interference. Lines 20–29 implement case 1 from Section 3.1.1. Lines 30–39 implement cases 2 and 3 from Section 3.1.1.

Note that a client u is removed from $\mathscr{C}$ *only* in the following situations:

- $x \neq y$ and u has an interference with $\langle x, f \rangle$ or $\langle y, g \rangle$ that is not fringe antipode. This can happen in lines 17 and 38.
- $x \neq y$, client u has an interference with x (y) and the value of x (y) is going to be updated to a new client. This can happen in lines 22 and 33.

It is easy to see that any such client u can never satisfy conditions P1–P3 used to construct H.

It can now be verified directly that Algorithm 6 is a correct implementation of the algorithm from Section 3.1.1. Next, we show that the running time of Algorithm 6 is $O(n \log n)$.

It is clear that the total number of ANN queries over all calls to RemoveInterference is $O(n)$ as each client/facility point is accessed at most once in an ANN query. The argument for NextPoint requires a bit more work as the same point in $\mathscr{C}$ or $\mathscr{F}$ may be accessed multiple times as the result of ANN queries. Still, we can show that each point is accessed at most a constant number of times during all calls to NextPoint. By the end of each iteration, we remove all interferences with clients in the new component H, so no client/facility is accessed in multiple iterations of the while loop. It now suffices to bound the number of accesses during a single iteration (in which one component H is built). We consider facilities and clients separately:

- Bounding number of accesses of $\mathscr{F}$. Note that all pairwise distances between clients in H are at least $2 \cos \beta > \sqrt{3}(1 + \epsilon)$. So any facility in $\mathscr{F}$ can be within distance $1 + \epsilon$ of at most two clients in H. Moreover, all facility points accessed in NextPoint are within distance $1 + \epsilon$ of some client in H. So each facility is accessed at most twice.
- Bounding number of accesses of $\mathscr{C}$. Consider any $u \in \mathscr{C}$ and suppose that it is accessed during the NextPoint calls from clients $S \subseteq H$. Note that we must have $d(u, v) \leq 2(1 + \epsilon)$ for all $v \in S$. Moreover, each pairwise distance in S is more than $\sqrt{3}$. So $|S| = O(1)$ as the dimension is constant.

Because each ANN query and insertion/deletion takes $O(\log n)$ time, the total time taken over all calls to RemoveInterference and NextPoint is $O(n \log n)$.

After constructing G, we apply the linear time edge-cover algorithm on cactus graphs (Theorem 5) to obtain the k-supplier solution. Finally, we perform binary search over the parameter L that incurs an additional logarithmic factor in the running time. We also lose an additional $1 + \epsilon$ factor in the approximation ratio (compared with Section 3.1.3) because of the use of approximate distances in $\mathscr{C}$ and $\mathscr{F}$. This completes the proof of Theorem 2.

4. Conclusions

We obtained improved approximation algorithms for the k-supplier problem on Euclidean metrics with guarantees strictly better than what is possible in general metrics. Our $(1 + \sqrt{3})$-approximation algorithm is close to the best possible approximation ratio because of the factor $\sqrt{7}$ hardness-of-approximation (Feder and Greene [10]). It would be interesting to close this gap. Another open question involves obtaining a near-linear time implementation of the $(1 + \sqrt{3})$ approximation algorithm. Finally, 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.

Acknowledgments

A preliminary version of this paper appeared in the conference on Integer Programming and Combinatorial Optimization (IPCO) 2013. Part of V. Nagarajan's work was done while at IBM T.J. Watson Research Center.

Endnotes

[1] 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.

[2] The self-loop can be labeled with any facility in the ball $B(c_0)$.

References

[1] An H, Bhaskara A, Chekuri C, Gupta S, Madan V, Svensson O (2015) Centrality of trees for capacitated *k*-center. *Math. Programming* 154(1/2):29–53.
[2] Arora S (1997) Nearly linear time approximation schemes for Euclidean TSP and other geometric problems. *Proc. 38th Annual Sympos. Foundations Comput. Sci.* (IEEE Computer Society, Piscataway, NJ), 554–563.
[3] Arya S, Mount DM, Netanyahu NS, Silverman R, Wu AY (1998) An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *J. Assoc. Comput. Machinery* 45(6):891–923.
[4] Baev ID, Rajaraman R, Swamy C (2008) Approximation algorithms for data placement problems. *SIAM J. Comput.* 38(4):1411–1429.
[5] Chan TM (1998) Approximate nearest neighbor queries revisited. *Discrete Comput. Geometry* 20(3):359–373.
[6] Chen DZ, Li J, Liang H, Wang H (2016) Matroid and knapsack center problems. *Algorithmica* 75(1):27–52.
[7] Clarkson KL (1994) An algorithm for approximate closest-point queries. *Proc. 10th Annual Sympos. Comput. Geometry* (ACM, New York), 160–164.
[8] Cygan M, Hajiaghayi M, Khuller S (2012) LP rounding for *k*-centers with non-uniform hard capacities. *Proc. 53rd Annual Sympos. Foundations Comput. Sci.* (IEEE, Piscataway, NJ), 273–282.
[9] Daskin MS, Snyder LV, Berger RT (2005) Facility location in supply chain design. Langevin A, Riopel D, eds. *Logistics Systems: Design and Optimization* (Springer, Boston), 39–65.
[10] Feder T, Greene DH (1988) Optimal algorithms for approximate clustering. Simon J, ed. *Proc. 20th Annual ACM Sympos. Theory Comput.* (ACM, New York), 434–444.
[11] Gonzalez TF (1985) Clustering to minimize the maximum intercluster distance. *Theoret. Comput. Sci.* 38:293–306.
[12] Har-Peled S, Mazumdar S (2004) On coresets for *k*-means and *k*-median clustering. Babai L, ed. *Proc. 36th Annual ACM Sympos. Theory Comput.* (ACM, New York), 291–300.
[13] Hochbaum DS, Shmoys DB (1985) A best possible heuristic for the *k*-center problem. *Math. Oper. Res.* 10(2):180–184.
[14] Hochbaum DS, Shmoys DB (1986) A unified approach to approximation algorithms for bottleneck problems. *J. Assoc. Comput. Machinery* 33(3):533–550.
[15] Kanungo T, Mount DM, Netanyahu NS, Piatko CD, Silverman R, Wu AY (2002) An efficient *k*-means clustering algorithm: Analysis and implementation. *IEEE Trans. Pattern Anal. Machine Intelligence* 24(7):881–892.
[16] Khuller S, Sussmann YJ (2000) The capacitated *K*-center problem. *SIAM J. Discrete Math.* 13(3):403–418.
[17] Kolliopoulos SG, Rao S (2007) A nearly linear-time approximation scheme for the Euclidean *k*-median problem. *SIAM J. Comput.* 37(3): 757–782.
[18] Krishnaswamy R, Kumar A, Nagarajan V, Sabharwal Y, Saha B (2015) Facility location with matroid or knapsack constraints. *Math. Oper. Res.* 40(2):446–459.
[19] Micali S, Vazirani VV (1980) An algorithm for finding maximum matching in general graphs. *Proc. 21st Annual Sympos. Foundations Comput. Sci.* (IEEE, Piscataway, NJ), 17–27.
[20] Mucha M, Sankowski P (2006) Maximum matchings in planar graphs via Gaussian elimination. *Algorithmica* 45(1):3–20.
[21] Panigrahy R, Vishwanathan S (1998) An $O(\log^* n)$ approximation algorithm for the asymmetric *p*-center problem. *J. Algorithms* 27(2): 259–268.
[22] Schrijver A (2003) *Combinatorial Optimization* (Springer, New York).
[23] Swamy C (2016) Improved approximation algorithms for matroid and knapsack median problems and applications. *Assoc. Comput. Machinery Trans. Algorithms* 12(4):1–22.
[24] Vaidya PM (1989) Approximate minimum weight matching on points in *k*-dimensional space. *Algorithmica* 4(4):569–583.
[25] Vaidya PM (1989) Geometry helps in matching. *SIAM J. Comput.* 18(6):1201–1225.