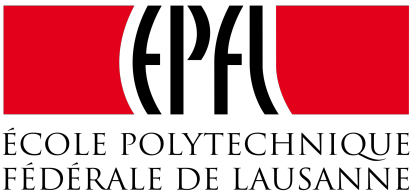


The k-center problem

Undergraduate Semester Project

Spring Semester 2014-15

Author:
Kirtan Padh

Supervisors:
Ola Svensson
Ashkan Norouzi-Fard

Abstract

The k-center problem is that of choosing k vertices as centers in a weighted undirected graph in which the edge weights obey the triangle inequality so that the maximum distance of any vertex to its nearest center is minimized. The problem is NP-hard, but there is a simple greedy 2-approximation algorithm which has been shown to be optimal. We consider here the capacitated k-center problem, where additionally each vertex has a capacity, which is a bound on the number of 'clients' it can serve if it is opened as a center. Unlike the uncapacitated k-center problem, our understanding of the capacitated version is far from complete and the first constant factor approximation was given fairly recently. We mainly concern ourselves with the case when all capacities are equal, which is called the uniform capacity k-center problem. We give here an L-approximation for the uniform k-center problem where each vertex has capacity L. This is an improvement over the current best approximation ratio for $L \leq 5$.

Contents

1 Introduction **1**
- 1.1 Location Problems . . . 1
- 1.2 History . . . 2
- 1.3 Some known problems . . . 4
  - 1.3.1 Minisum Problems . . . 4
    - 1.3.1.1 The Weber Problem . . . 4
    - 1.3.1.2 The Median Problems . . . 4
  - 1.3.2 Minimax Problems . . . 5
    - 1.3.2.1 The k-Center Problem on the Plane . . . 5
    - 1.3.2.2 The k-Center Problem . . . 6

2 The k-Center Problem **6**
- 2.1 The Uncapacitated k-Center Problem . . . 6
  - 2.1.1 Greedy algorithm . . . 7
  - 2.1.2 Parametric Pruning algorithm . . . 8
  - 2.1.3 Optimality of the approximation ratio . . . 9
- 2.2 Capacited k-Center Problem . . . 10
  - 2.2.1 9-approximation for the Capacited k-Center Problem . . . 10
  - 2.2.2 Uniform Capacity k-Center Problem . . . 13

3 L-approximation algorithm **13**
- 3.1 Proof outline . . . 13
- 3.2 Preliminaries . . . 14
- 3.3 The algorithm . . . 17
- 3.4 Proof of Correctness . . . 18
- 3.5 Future Directions . . . 22

Bibliography **22**

1 Introduction

Due to their wide and natural applicability location problems are studied in a wide range of fields such as operations research, mathematics, computational geometry, combinatorial optimization and geography, to name a few. A mathematician would probably define a location problem as follows: "given some metric space and a set of known points, determine a number of additional points so as to optimize a function of the distance between new and existing points." A geographer might be interested in the following:"given a large wildlife sanctuary, find the optimal positions of locating k security stations so that the maximum distance from any point of the sanctuary to a security station is minimized" People in operations research might want to determine "the location of plants and market catchment areas in the presence of potential customers," while those in computational geometry might be interested in "the minimum number of equal geometrical shapes that are required to cover a certain area, and the positions of their centroids."

1.1 Location Problems

All these views have in common the basic components of a location problem: a space, in which a distance measure is defined, a set of given points, and candidate locations for a fixed or variable number of new points. We refer to the known points as "clients", and the new points to be located as "facilities" or "centers." As far as the space is concerned in which customers are and facilities are to be located, we distinguish between a subset of the d-dimensional real space and networks. Each of these two categories has two subcategories: one, in which the location of the facilities is continuous, and the other in which it is discrete. In discrete problems, the decision is whether or not to locate a facility at that spot, thus modeling the decision with a binary variable is obvious. The result is a integer linear programming problem.

On the other hand, continuous problems will have continuous variables associated with them, indicating the coordinates of the facilities that are to be located. For instance a simple example of a continuous facility location problem in 2-D space could be as follows:
Given a finite set of points S in the plane, find a point on the plane(not necessarily in S) for which the sum of distances of this point from the points of S is minimized.
If we only allow the point to be chosen from the set S, it would be a discrete facility location problem. We could also define a similar problem on a network as follows:
Given a weighted graph $G = (V, E)$, find a vertex $v \in V$ such that the sum of distances from v

to every other vertex in V is minimized.
This is an example of a discrete network location problem. If we allow v to lie on an edge of the graph, this would be a continuous network location problem. Having seen the very basic types of location problems, we now move on to give a short history of location problems. This will lead us to a discussion on discrete network location problems; which in turn will lead us to the k-center problem. Most of the material in the following chapter is adapted from [1].

1.2 History

In the early seventeenth century, a question was posed about how to solve the following puzzle: "Given three points in a plane, find a fourth point such that the sum of its distances to the three given points is minimized." This question is credited to Fermat, and is apparently considered to be one of the first questions which could be called a formal formulation of what we now call a facility location problem. The earliest geometrical solution is believed to be due to Torricelli(15981647), Fermats pupil and the discoverer of the barometer, although no one is really sure due to the elementary nature of the problem. A generalization of this problem is called the Weber's problem. Weber generalized the problem by assigning different weights to the known points, thus transforming the mathematical puzzle into an industrial problem, in which a plant is to be located (the unknown point) so to minimize transportation costs from suppliers and to customers (the known points) requiring different amounts of products (the weights).

Even if the first formal occurrence of a location problem were due to Fermat or one of his contemporaries, it is likely that location analysis must be much older than that, because of the natural occurrence of location problems in everyday life. For centuries before, people may have wondered in which cave to live, where to build houses, villages, churches, and other "facilities" all of which could be thought as facility location problems. And they must have solved these problems using some sort of heuristic method.

Location problems frequently require solving an associated allocation or assignment problem: if locations for more than one facility are known, which facility will serve what customers? A step towards the solution of the allocation problem was taken very early in the seventeenth century. Descartes imagined the universe as a set of regions around each star-the "heavens", and illustrated his theory with a drawing that made an informal use of what later would become known as Voronoi polygons and Voronoi diagrams(Figure 1.1). Voronoi diagrams are generally useful tools when consumers have to be assigned to their closest facilities.

In 1857, Sylvester asked another location-related question [2] "Find the smallest circle that encloses a given set of points in the plane" This question was later answered by Sylvester

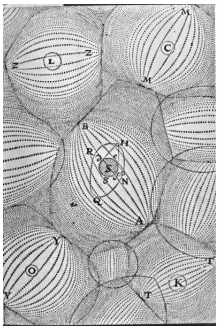


FIGURE 1.1: Descartes' Vortices: The first instance of a Voronoi Diagram to appear in literature

himself in 1860 and by Chrystal in 1885. Today we call it *one-center problem in the plane* or the *Smallest enclosing circle problem* and there is a linear time algorithm given by Megiddo [3].

Hakimi [4] is believed to be the paper which introduced and gave a boost to modern quantitative location theory. The paper introduced what we today call as the k-median problem.

Year	Number of Publications
1982	10
1981	14
1980	12
1979	11
1978	9
1977	10
1976	6
1975	5
1974	4
1973	2
1972	6
1971	3
1970	4
1969	1
1968	1
1967	2
1966	1
1965	1
1964	1
1963	0
1962	1
⋮	
1869	1

FIGURE 1.2: A data table from [5]: Frequency distribution of publication of network location references

According to Tansel et al. [5], Figure 1.2 demonstrates that the literature on network location problems has grown rapidly since the appearance of Hakimi's paper [4]. Many different kinds of location problems have been introduced and studied since. Let us look at a few of them.

1.3 Some known problems

Facility location problems can take many forms, depending on the objective function. Let us look at two general classes of such problems called *Minisum* and *Minimax* problems.

1.3.1 Minisum Problems

Minisum problems are named so due to the fact that some function of a sum of center-client distances is minimized. These problems include single-facility, multiple-facility and weighted versions of the Weber problem, the simple plant location problem and the k-median problem.

1.3.1.1 The Weber Problem

The Weber problem is the problem of choosing a point $X = (x, y)$ on the plane such that given n points $P_i = (a_i, b_i)$ on the same plane and weights $w_1 \dots w_n$ for these points, we wish to minimize the sum of weighted distances from (x, y) to all the known points. Formally:

$$\min \quad z(X) = \sum_{i=1}^{n} w_i.d(X, P_i)$$

where $d(X, P_i)$ is the distance between the facility and point P_i. The most common distance metric used is the Euclidean metric. An iterative numerical solution for the problem was given by Kulin and Kuenne [6] in 1962.

An application of this problem could be to locate a single warehouse (the facility) that supplies different amounts (weights) of products to a number of dealers (the demands), in such a way that the total transportation cost is minimized, assuming that this cost depends on the distance and amount of transported product. Clearly, this problem is a continuous facility location problem. We now look at an example of a minisum network location problem.

.

1.3.1.2 The Median Problems

The network equivalent of the single facility Weber problem on the plane is the 1-*median problem*, whose formulation is exactly the same as that of the single facility Weber problem. However, in the 1-median problem the clients are in the from of the nodes of a network, represented as a weighted graph. The problem is now to find a node v such that the sum of distances from v to each of the other nodes is minimized. This is clearly a discrete network location problem. The 1-median problem was solved for location on tree networks in 1962 by Hua Lo-Keng [7].

The k-median problem

This is a natural generalization of the 1-median problem problem. Here, we have to choose exactly k centers. As more than one facility is to be located, an allocation problem must be solved together with the location problem, meaning that not just do we have to choose centers, but also allocate each client to a center. The objective is still the same: to minimize the sum of distances from every client to its assigned center. This naturally leads to the formation of k clusters in the network, each cluster consisting of a center and all the clients which have been assigned to that center. Therefore, these kinds of algorithms are commonly used as clustering algorithms in computational geometry and social network theory. The most common clustering algorithm used for social networks is the k-means clustering, where we try to minimize the average distance from a client to its assigned center. The first constant factor approximation for the k-median problem was given by Charikar, Guha, Tardos and Shmoys [8].

Because of the nature of the problem as a decision problem, where we have to decide for each node whether we wish to open it as a center, there is a natural formulation of the k-median problem in terms of an integer linear program. Clearly, this is a discrete network location problem. If we consider a variant of this problem where we allow the open centers to be on the edges of the graph, it would be a continuous facility location problem.

1.3.2 Minimax Problems

The idea behind minimax problems is to minimize the longest distance between a customer and its assigned facility, which is why we call it minimax. As before, these problems have been defined both on a plane and on a network. These kinds of problems can be considered for the location of centers which provide emergency services, for example hospitals and police stations, where the aim is not to reduce transport costs but to decrease the time in which immediate service can be provided for everybody.

1.3.2.1 The k-Center Problem on the Plane

Let us first look at what is now called the 1-center problem on the plane. The problem consists of finding the smallest circle containing a set of given points. Finding the solution of this problem is equivalent to finding the location of the center of a circle that encloses all given points, in such a way that its radius is minimized. If a facility is located at the center of the circle, and the points represent demands, the maximum distance between the facility and a demand will be minimized. An optimal algorithm for the k-center problem on the plane was given by Drezner [9] in 1984.

We can naturally generalize the 1-center problem on the plane to a k-center problem on the plane. We now have to choose k points as centers and assign each known point to a center so that the maximum distance from a point to its assigned center is minimized. Since we do not have capacity constraints on the centers, each client is simply assigned to the center which is closest to it. This is called the *uncapacitated k-center problem on the plane.* This is clearly a continuous facility location problem.

Let us now look at the network location version of this problem, which is the one that is of primary interest to us in this article.

1.3.2.2 The k-Center Problem

This is a natural extension of the k-center problem on the plane where the nodes are in the form of a network instead of being on the plane. The task is now to choose k nodes as centers and assign each node to a center such that the maximum distance of a client from a center is minimized. It was first formulated and solved by Hakimi in 1964. Once again, we can formulate it as a linear program because of the decision problem of whether to open a node. We now look at the k-center problem and its variants in detail.

2 The k-Center Problem

In this section we deal with the *k-center problem* and describe some known algorithms for some of its variants. We are given a weighted graph $G = (V, E)$ with weights $w : E \to \mathbb{R}^+$ satisfying triangle inequality. Define $d_G(u, v)$ to be the distance between vertices u and v in G and $d(u, S) = \inf_{z \in S} d(u, z)$. The k-center problem is to choose a subset of vertices $S \subseteq V$ as centers such that $|S| \leq k$ and find an assignment of vertices to centers $a : V \to S$ so that $\sup_{v \in V} d_G(v, a(v))$ is minimized. Additionally we could also have capacity constraints on each vertex as an upper bound on the number of clients that it can serve if chosen as a center, in which case we call it the *capacitated k-center problem.* The the most commonly considered case though is the one where there are no capacity constraints on the vertices and is called the *uncapacitated k-center problem* or the *metric k-center problem.* Let us first look at the uncapacitated k-center problem.

2.1 The Uncapacitated k-Center Problem

In the case when we do not have capacity constraints, the problem is only to choose the subset S, since we can assign each vertex to the center that is nearest to it. Using the same notation

as described above, the uncapacitated k-center problem can therefore be formulated as follows: Find $S \subseteq V, |S| \leq k$ such that $\sup_{v \in V} d_G(v, S)$ is minimized.
Approximation algorithms for this problem have been well studied and are known to be optimal [10, 11]. Let us first describe a greedy algorithm given by Gonzalez [11].

2.1.1 Greedy algorithm

This is perhaps the simplest algorithm for the uncapacitated k-center problem and was given by Gonzalez in 1985. In this algorithm, we simply build the set S by iteratively adding a vertex to which is farthest from the set of vertices we have already chosen. We keep doing this until we have chosen k vertices. Formally, the algorithm is as follows:

Algorithm 1 Greedy algorithm

```
1: procedure ChooseS(V, w)
2:     initialize S = φ;
3:     while |S| < k do
4:         Find v ∈ V for which d_G(v, S) is maximum over all v ∈ V
5:         Add v to S
6:     end while
7: end procedure
```

Lemma 2.1. *Algorithm 1 is a 2-approximation algorithm for the uncapacitated k-center problem.*

Proof. Let O be the set of optimal k-centers and the optimum value $\sup_{v \in V} d_G(v, O) = \mathsf{OPT}$. We need to show that $\sup_{v \in V} d_G(v, S) \leq 2.\mathsf{OPT}$. Note that the sequence of distances from a new chosen center, to the closest center to it among previously chosen centers is non-increasing. Suppose towards contradiction that $\sup_{v \in V} d_G(v, S) > 2.\mathsf{OPT}$. This implies that each pair of vertices in S must be at a distance greater than $2.\mathsf{OPT}$ from each other, since the sequence of distances of a new chosen center is non-increasing. So we now have $k+1$ vertices such that the pairwise distance between them is greater than $2.\mathsf{OPT}$. By pigeonhole principle at least two of these vertices, say u_1 and u_2, must share the same center, say o_1, in the optimal assignment. We know that $d_G(u_1, o_1) \leq \mathsf{OPT}$ and $d_G(u_2, o_1) \leq \mathsf{OPT}$. By triangle inequality $d_G(o_1, o_2) \leq 2.\mathsf{OPT}$, which is a contradiction. Therefore, we cannot have $\sup_{v \in V} d_G(v, S) > 2.\mathsf{OPT}$, which means that $\sup_{v \in V} d_G(v, S) \leq 2.\mathsf{OPT}$. This proves that the algorithm is a 2-approximation. □

We now describe another interesting 2-approximation given by Hochbaum and Shmoys [10].

2.1.2 Parametric Pruning algorithm

The idea of this algorithm is to prune out the irrelevant edges from the input. What we mean by this is that suppose $\mathsf{OPT} = t$, then since we want a 2-approximation, all edges that are of length more than $2t$ are irrelevant. Meaning that if two vertices are connected by such an edge and one of them is chosen as center, the other is still too far away. We start by ordering the edges by weight: $w(e_1) \leq \dots w(e_m)$. Let $E_i = \{e_1 \dots e_i\}$ and $G_i = (V, E_i)$. Clearly, the problem is now equivalent to finding the minimum i such that G_i has a dominating set of size k. Note that the *dominating set* for a graph $G = (V, E)$ is a $D \subseteq V$ such that every vertex not in D is adjacent to at least one vertex in D. Let i^* be such a minimal i, then $\mathsf{OPT} = w(i^*)$. We denote $G^2 = (V, E')$ as the square graph of G, where $(u, v) \in E' \iff d_G(u, v) \leq 2$ and $u \neq v$.

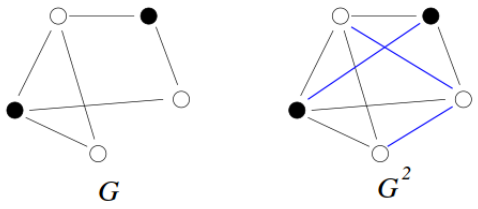


FIGURE 2.1: A squared graph: The blue edges are the edges added after squaring

Lemma 2.2. *For any independent set I in G^2, $|I| \leq |D|$, where D is the minimum dominating set in G.*

Proof. If we consider stars in G centered at the vertices in D, these $|D|$ stars must cover G since D is a dominating set. A star in G becomes a clique in G^2 and therefore we have $|D|$ cliques in G^2 that cover G^2. Any independent set I in G^2 can have at most one vertex from each of these $|D|$ cliques. Therefore $|I| \leq |D|$. □

Now the algorithm takes advantage of the fact that maximal independent sets can be found in polynomial time. The algorithm is as follows:

Algorithm 2 Parametric pruning algorithm

```
1: procedure ChooseS(V, w)
2:     Construct G_1^2 ... G_m^2;
3:     Find the maximal independent set M_i in each G_i^2
4:     Find the smallest i for which |M_i| ≤ k and call it i'
5:     Return M_i'
6: end procedure
```

Lemma 2.3. $w(e_{i'}) \leq \mathsf{OPT}$

Proof. We denote by D_i the minimum dominating set of G_i. By the algorithm, for every $i < i'$ we have $|M_i| > k$. Therefore by lemma 2.2 $|D_i| > k$. So we must have $w(e_i) < \mathsf{OPT}$ for every $i < i'$. Therefore $w(e_{i'}) \leq \mathsf{OPT}$. □

Finally, we can prove the following:

Lemma 2.4. *Algorithm 2 is a 2-approximation algorithm for the uncapacitated k-center problem.*

Proof. Any maximal independent set I in $G_{i'}^2$ is also a dominating set because if there was a vertex u which was not dominated, $I \cup u$ would also be independent, contradicting that I is maximally independent. In $G_{i'}^2$ we have $|M_{i'}|$ stars that cover $G_{i'}^2$. We know by the algorithm that $|M_{i'}| \leq k$. If we choose $S = M_{i'}$, we have $\sup_{v \in V} d_G(v, S) \leq 2.w(e_{i'})$. We know by the previous lemma that $w(e_{i'}) \leq \mathsf{OPT}$. Therefore $\sup_{v \in V} d_G(v, S) \leq 2.\mathsf{OPT}$. □

It was proven by Baum et al. [10] in 1985 that a 2-approximation is optimal. We present the result below.

2.1.3 Optimality of the approximation ratio

We use reduction from dominating set to prove the following result.

Theorem 2.5. *If $\mathsf{P} \neq \mathsf{NP}$, no approximation algorithm for the uncapacitated k-center problem gives a $(2-\epsilon)$-approximation for any $\epsilon > 0$.*

Proof. Given a graph G, we define a graph G' as follows:

$$d_{G'}(u, v) = \begin{cases} 1 & \text{if } (u, v) \in E \\ 2 & \text{otherwise} \end{cases}$$

Clearly, G' satisfies triangle inequality. If G has a dominating set of size at most k, then it is a k-center of cost 1 in G'. A $(2-\epsilon)$-approximation algorithm will give us one with *cost* < 2 which in this case has to be 1. If no such dominating set exists, every k-center has *cost* $\geq 2 > 2 - \epsilon$. Thus, a $(2 - \epsilon)$-approximation algorithm for the uncapacitated k-center problem can be used to determine in polynomial time whether there exists a dominating set of size k. This implies $\mathsf{P} = \mathsf{NP}$. Therefore this cannot happen assuming $\mathsf{P} \neq \mathsf{NP}$. □

We can perhaps say that proving the optimality of the algorithm for this problems leads to a satisfying conclusion to the uncapacitated k-center problem. But it also naturally leads us to consider generalizations of of the problem, where each vertex additionally has a capacity constraint. We discuss this problem, called the *capacitated k-center problem* next.

2.2 Capacited k-Center Problem

The capacitated k-center problem is the k-center problem along with a set of capacity constraints $L : V \to \mathbb{N}$. This constraints mean that if we open $v \in V$ as a center, it can serve at most $L(v)$ clients. In the uncapacitated case we just assigned every vertex as a client to the center which was nearest to it, but we cannot do this anymore. Now not just do we have to choose the set S, but also solve an allocation problem where we have to decide which open center we should allocate a vertex to.

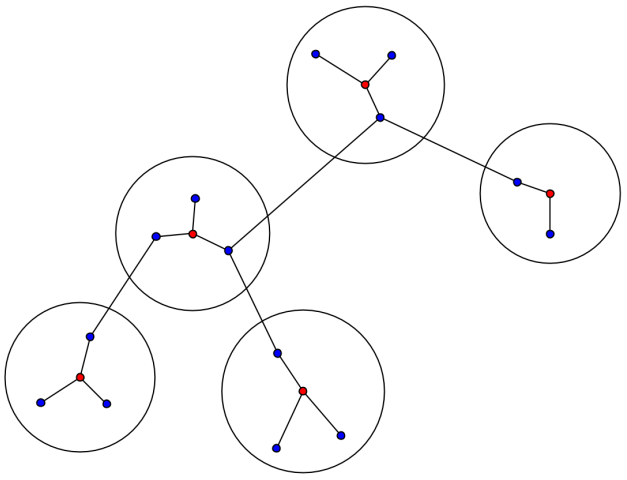
This additional constraint makes the problem surprisingly harder than the uncapacitated one. This problem has been studied since the 1990s and the first constant factor approximation for a special case of the problem, when the capacity constraints at all vertices are same was given by by Bar-Ilan, Kortsarz and Peleg [12] in 1993. But we did not have a constance factor approximation for general capacities until a long time after that. The first constant factor approximation for general capacities was only obtained in 2012 by Cygan et al. [13] for an unspecified constant.

This lies in stark contrast to the existence simple greedy optimal algorithm for the uncapacitated case and points to a large gap in our understanding of the capacitated and uncapacitated versions of the problem. The most significant step till date in bridging this gap is perhaps the 9-approximation algorithm for the capacitated k-center problem given by An, Bhaskara and Svensson [14], published in 2013. This is because the idea behind the algorithm was relatively simple and it was the first algorithm for the case with general capacities which had a reasonable approximation ratio. It remains the best known algorithm for the capacitated k- center problem in terms of approximation ratio. We attempt to describe the idea of the algorithm below.

2.2.1 9-approximation for the Capacited k-Center Problem

This is an LP-rounding algorithm which proceeds by transferring the opening variable given by a standard LP relaxation of the problem. The algorithm starts by reducing the problem from a weighted graph to one on an unweighted graph. We have described this reduction in detail in Section 3.2. Basically the idea of this reduction is that we make a guess at an optimal value τ and check the feasibility of a standard LP for this optimal value. This determines if it is possible to fractionally open k vertices while assigning every vertex to a center that is adjacent in $G_{\leq\tau}$(as defined in Section 3.2). If it is feasible, the algorithm gives a 9-approximation.

The LP solution specifies a set of *opening variables* that indicate the fraction to which each vertex is to be opened by the optimal fractional solution. We refer to the unweighted graph on which we are doing the LP rounding as $G = (V, E)$. The core of this algorithm is to round these opening variable by "transferring" openings between vertices to make them integral.



h!

FIGURE 2.2: An example of how the clustering works. Each large circle corresponds to a cluster. Red vertices are cluster centers. The clustering is done such that each vertex has a total opening variable of at least one. Each cluster center is at a distance of 3 from its nearest cluster center.

The algorithm first partitions V into clusters such that each cluster has a total opening variable of at least one. We can see that the cluster centers form a tree instance.
The algorithm then introduces one representative vertex for each cluster, which is called the *auxiliary vertex* of the cluster. The algorithm now proceeds to transfer openings to make the opening variable of every auxiliary vertex to be equal to one. This transfer happens within the cluster itself. This can be done because each cluster has a total opening variable of at least one. Now we have an auxiliary vertex for each cluster which has opening variable one.

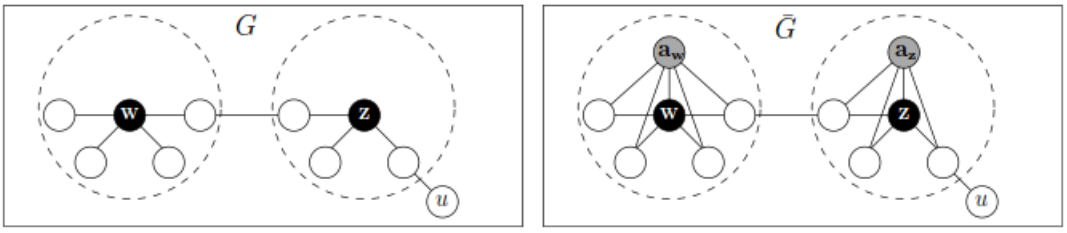


FIGURE 2.3: a_w and a_z are the auxiliary vertices of the clusters(taken from [14])

We now use the auxiliary vertices to define a tree instance. The internal nodes of the tree are all the auxiliary vertices. Corresponding to each auxiliary vertex, the leaf nodes for this vertex are the nodes which are in the same cluster as that auxiliary vertex and still have a non-zero opening variable. This gives us a tree such that each of its internal nodes has opening variable one.

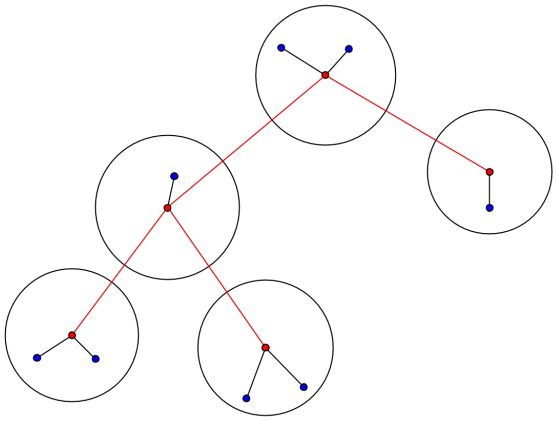


FIGURE 2.4: A tree instance obtained from the auxiliary vertices. Red vertices are the auxiliary vertices. Since all auxiliary vertices are fully open, all internal nodes in the tree have a opening variable of one. Note that the diagram is representative.

Here we must observe that a transfer of distance one in the tree instance does not necessarily correspond to a transfer of distance one in the original graph. But it is still a transfer of small distance in the original graph though. Hence if we can round the tree instance using transfer of openings within a small distance, we can do these transfers in the original graph using only slightly longer distances. And this is exactly what the algorithm now does. The algorithm now recursively solves the tree instance starting with the leaves.

The algorithm departs from previous works by using a simple local strategy that does not depend on distant vertices and applies to every non-leaf node. the paper says that "The reason our strategy works locally is that the decision of closing fully open centers is determined using solutions to sub-instances, which are solved recursively. This key idea significantly eases the analysis and leads to our optimal algorithm for tree instances."

It seems that when we are trying to study a hard problem, considering simplified cases and variants of the problem turns out to be quite illuminating sometimes and gives us some insight into the original problem that we are interested in solving. It would seem that sometimes a small tweak to a problem can make it much easier to solve, perhaps pointing us to where the 'bottleneck' for solving the problem is. For instance the uniform capacity k-center problem is apparently much easier to solve than the non-uniform case. It took over 15 years after the 6-approximation for the uniform capacities case to get the first constant factor approximation algorithm for the non-uniform case. Next we look at the uniform capacity k-center problem.

2.2.2 Uniform Capacity k-Center Problem

The uniform capacity k-center problem is the same as the capacitated k-center problem where the capacity of every vertex is the same. We will denote here by $L \in \mathbb{N}$ the capacity of every vertex. The first constant factor algorithm for the uniform capacity k-center problem was given by Bar-Ilan, Kortsarz and Peleg [12] in 1993. Already in 1996 Khuller and Sussmann [15] gave a 6-approximation algorithm for this problem. There was another 6-approximation given by An et al. [14] in 2013 using completely different techniques.Apart from this, there has been no improvement in the approximation ratio for the uniform capacity k-center problem since the 6-approximation was suggested by Khuller and Sussmann in 1996.

We present here an algorithm which beats the current best approximation ratio for small capacities, in particular for $L < 6$. We give an algorithm which gives an L-approximation when the capacity of each vertex is L. Also, it is much simpler than the algorithms that have been previously suggested for the uniform capacity k-center problem. We describe our algorithm next.

3 L-approximation algorithm

The result which we prove here is as follows.

Theorem 3.1. *There exists an L-approximation for the uniform capacity k-center problem with where each node has capacity L and each node can be opened as a center only once.*

We give a simple algorithm which gives us a better approximation guarantee than the current best algorithm, when the capacities are small. Observe that even though our algorithm is not a constant factor approximation, it differs from other non-constant factor approximations in that the approximation ratio does not depend on the size of the input graph, as is usually the case for non-constant factor approximation algorithms. So in that sense it can be thought of as a constant factor approximation for a fixed L. We now go on to give an outline of our algorithm.

3.1 Proof outline

Our algorithm guesses the optimal solution value τ and considers the unweighted graph $G_{\leq\tau}$ as in An, Bhaskara and Svensson [14] on the given set of vertices where two vertices are adjacent if and only if they are at a distance of at most τ. We prove in Lemma 3.2 that we can assume this graph to be connected(Cygan et al. [13]). We then solve a standard relaxed LP on this graph to check whether there is a feasible solution that fractionally opens k vertices and assigns

centers so that each vertex is assigned to a center at distance at most one from itself. If the LP is feasible, our algorithm assigns centers so that each vertex is assigned to a center within a distance of at most L in $G_{\leq\tau}$, thus giving us an L-approximation. If the LP is infeasible, we consider $G_{\leq\tau}$ with a larger value of τ. Up to this point, our algorithm is identical to Cygan et al. [13] and An et al. [14], but while these algorithms use powerful LP rounding techniques to solve the fractional solution given by the LP, we give a purely combinatorial algorithm that does not consider the fractional openings given by the LP since we are interested in improving the result only for small capacities. This naturally leads to the question whether we can use our idea together with the opening variables to get a stronger result.

We consider a spanning tree $\mathcal{T}$ of $G_{\leq\tau}$ and give a recursive algorithm that assigns centers such that every client is at a distance of at most L from it's assigned center, in the tree $\mathcal{T}$, and thus also in the graph $G_{\leq\tau}$. We start by arbitrarily choosing a vertex r as the root of $\mathcal{T}$ and 'mark' an appropriate number of vertices in the subtrees rooted at each of the children of r, where 'marking' a vertex means that it has been assigned to a center. We then proceed to recursively solve sub-trees rooted at each of the children of r. Each of the recursive instances involves marking an appropriate number of vertices to maintain the following invariants that will help us prove the correctness of our algorithm:

1. We wish to mark exactly 0 (mod L) number of vertices in each recursive instance, except possibly when we start the algorithm with the root we have chosen.
2. When we start to recursively solve a sub-tree rooted at some vertex u, the set of vertices that have already been marked at this point is a subset of the vertices that we wish to mark before moving on to the children of u.

We define a few preliminaries and make this arguments formal.

3.2 Preliminaries

Given a weighted graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}^+$ which satisfies triangle inequality and an integer capacity L, the *uniform capacity k-center problem* is to choose k vertices to *open* as *centers* and an assignment of every vertex to an open center which minimizes the longest distance between a vertex and the center it is assigned to while while respecting the capacity constraints. We denote by $d_G(u, v)$ the distance between u and v is the graph G.

Reduction to an unweighted problem: We reduce the weighted problem to an unweighted one(An et al. [14]) by determining a lower bound τ^* on the optimal value of the solution. We make a guess τ at OPT and try to decide whether $\tau < \mathsf{OPT}$. We consider $G_{\leq\tau} = (V, E_{\leq\tau})$ to be

an unweighted graph in which two vertices are adjacent if and only if they are at a distance less than τ in G. That is, $E_{\leq\tau} = \{(u,v)|d_G(u,v) \leq \tau\}$. A feasible solution of value τ assigns every vertex to a center that is adjacent in $G_{\leq\tau}$ and conversely, if a solution assigns every vertex to a center that is adjacent in $G_{\leq\tau}$, its value is at most τ. We check the feasibility of τ by considering a standard LP((3.1)) that checks whether there is a fractional feasible solution that asigns vertices to centers that are adjacent in $G_{\leq\tau}$. We denote the following feasibility LP by $LP_k(G)$

$$\begin{aligned}
\sum_{u\in v} y_u &= k && \\
x_{uv} &\leq y_u && \forall u,v \in V; \\
\sum_{v:(u,v)\in E} x_{uv} &\leq L.y_u && \forall u \in V; \\
\sum_{u:(u,v)\in E} x_{uv} &= 1, && \forall v \in V; \\
0 \leq x, y &\leq 1 &&
\end{aligned} \tag{3.1}$$

x_{uv} here is the *assignment variable* which tells us what fraction of the opening of v has been given to the center u. y_u is the *opening variable* which tells us by what fraction vertex u has been opened as a center. It turns oot that this LP has an unbounded integrality gap in general, but assuming it to be connected drastically changed the situation, as shown by Cygan et al. [13].

Lemma 3.2. *We can assume without loss of generality that $G_{\leq\tau}$ is connected.(Cygan et al. [13])*

Proof. Consider the connected components of $G_{\leq\tau}$. If $\tau > \mathsf{OPT}$, a vertex can only be assigned to vertices in the same connected component. For each connected component G_i of $G_{\leq\tau}$ the algorithm decides the minimum value of k_i for which $LP_{k_i}G_i$ is feasible. If $\sum_i k_i \geq k$, this certifies that $\tau < \mathsf{OPT}$. We consider τ^* to be the smallest τ for which the algorithm fails to certify that $\tau < \mathsf{OPT}$. This τ^* is obtained by binary search on the τ. We now solve each connected component separately. □

Given a connected graph G and integer k and L for which $LP_k(G)$ is feasible, our algorithm chooses a set of k vertices to open with an assignment of every vertex to an open center that is within the distance of L. $d_{G_{\leq\tau^*}}(u,v) \leq L$ implies $d_G(u,v) \leq L\tau^* \leq L.\mathsf{OPT}$.

Lemma 3.3. *Suppose there exists an algorithm that given a connected graph G , capacity $L \in \mathbb{N}$ and $k \in \mathbb{N}$ for which $LP_k(G)$ is feasible, finds an assignment of centers such that for each vertex u which has been assigned to a center v; $d_{G_{\leq\tau^*}}(u,v) \leq \alpha$, then there exists an α-approximation for the uniform capacity k-center problem.*

Our problem has now been reduced to an unweighted version; that of finding an assignment of centers in $G_{\leq\tau^*}$. We now give some definitions relevant to our algorithm.

Definition 3.4 (C_v). Given a rooted undirected tree $\mathcal{T}$ and a vertex $v \in \mathcal{T}$, we define C_v to be the set of children of v in $\mathcal{T}$. We assume an arbitrary but fixed ordering of vertices in C_v and denote them by $\{v_1, v_2 \ldots\}$. We define $C_v = \phi$ if v is a leaf.

If at any point we talk about the *first* child or *next* child of a vertex, it is in the ordering $\{v_1, v_2 \ldots\}$.

Definition 3.5 ($\mathcal{T}_v$). Given an undirected tree $\mathcal{T}$ with root r and a vertex $v \in \mathcal{T}$, we denote by $\mathcal{T}_v$ the sub-tree rooted at v. We denote by $|\mathcal{T}_v|$ the number of vertices in $\mathcal{T}_v$. Clearly $\mathcal{T}_r = \mathcal{T}$.

Definition 3.6 (Set of interest $\mathcal{I}_v$ of a vertex v)**.** Given an undirected tree $\mathcal{T}$ with root r, capacity L and a vertex $v \in \mathcal{T}$, we define $\mathcal{I}_v$ by the following algorithm.

Algorithm 3 Constructing $\mathcal{I}_v$

```
1: procedure ConstructSOI(T_v,L)
2: initialize I_v = {v};
3:     for each node u ∈ C_v  do
4:         t = |T_u| (mod L);
5:         U = Select(T_u, t);        ▷ selects and returns a set of t vertices from T_u
6:         I_v = I_v ∪ U;
7:     end for
8: return I_v;
9: end procedure
```

ConstructSOI($\mathcal{T}_v$,L) constructs the *Set of Interest* for vertex v given the sub-tree rooted at v and the capacity L. The set $\mathcal{I}_v$ consists of:

1. v itself.
2. For each child u of v, exactly $|T_u| \pmod L$[1] vertices from the sub-tree rooted at u.

In the algorithm when we are solving the sub-tree rooted at some vertex v, we will wish to mark all the vertices in the set $\mathcal{I}_v$ before we move on to the next recursive instance, which is why we call $\mathcal{I}_v$ the *Set of interest*.
The procedure Select($\mathcal{T}_u$, t)(Algorithm 4) selects $|T_u| \pmod L$ vertices from the sub-tree rooted at u as follows:

- Let w be the first child of u. If $|T_w| \pmod L$ is smaller than the number of vertices that we still need to add to $\mathcal{T}_u$, we recursively select $|T_w| \pmod L$ vertices from w and set w be the next child of u.

[1] For any $p \in \mathbb{N}$ we assume that $p \pmod L \in \{0, 1 \ldots L-1\}$

- Repeat until either we have selected a sufficient number of vertices from $\mathcal{T}_u$ or $|T_w|$ (mod L) has more vertices than we need.
- If we have not selected a sufficient number of vertices from $\mathcal{T}_u$ yet, select exactly as many vertices from $\mathcal{T}_w$ as we need.

Algorithm 4 Subroutine for selecting vertices

```
1:  procedure SELECT(T_u, t)
2:      if t == 0 then
3:          return φ;
4:      end if
5:  initialize S = {u};
6:      i = 1;
7:      w = u_i;
8:      while ( |S| < t and |T_w| (mod L) ≥ (t − |S|) ) do
9:          S = S ∪ Select(T_w, |T_w| (mod L));
10:         i = i + 1;
11:         w = u_i;
12:     end while
13:     if |S| < t then
14:         S = S∪Select(T_w, t − |S|);
15:     end if
16: return S;
17: end procedure
```

Corollary 3.7. $|\mathcal{I}_v| \equiv |\mathcal{T}_v|$ $(mod\ L)$

We now have all the tools and definitions we need to describe our algorithm.

3.3 The algorithm

Given a weighted graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$ which satisfies triangle inequality and an integer capacity L with each vertex having capacity L, our algorithm first constructs $G_{\leq \tau^*}$ as described in Lemma 3.2. We assume $G_{\leq \tau^*}$ to be connected(Lemma 3.2) and denote it by $\mathcal{G}$. Our algorithm finds a spanning tree $\mathcal{T}$ of $\mathcal{G}$ and chooses an arbitrary vertex r to call as the root of $\mathcal{T}$. We now have a rooted tree $\mathcal{T}$ which our algorithm will use. We denote the *set of marked vertices* by $\mathcal{M}$.

Our algorithm starts with the root r and recursively does a top down processing of $\mathcal{T}$. Our algorithm proceeds by *marking* vertices at each step, where marking a vertex means that it has been assigned as a client to an open center. At the step when we are processing the sub-tree rooted at some vertex $u \in \mathcal{T}$, we wish to mark $\mathcal{I}_v$ before we move on.

Algorithm 5 L-approximation algorithm

procedure ProcessTree($\mathcal{T}_r$, $\mathcal{M}$)
    $t = \left\lceil \frac{|\mathcal{I}_r \setminus \mathcal{M}|}{L} \right\rceil$
    $S = \{r_1, \ldots, r_t\}$
    Open every vertex in S as a center
    Assign every vertex of $\mathcal{I}_r \setminus \mathcal{M}$ as a client to one of the vertices in S
    $\mathcal{M} = \mathcal{M} \cup \mathcal{I}_r$
    for each child u of r **do**
        ProcessTree($\mathcal{T}_u$, $\mathcal{M}$)
    end for
end procedure

Our algorithm will always open exactly $\left\lceil \frac{|\mathcal{G}|}{L} \right\rceil$ vertices. This follows from the fact that $\mathcal{I}_v \setminus \mathcal{M}$ is always of size 0 (mod L) except possibly when v is r.

3.4 Proof of Correctness

We first show that the Procedure *Select* is well defined. We need to show this because *Select*$(\mathcal{T}_u, t)$ selects a particular number of vertices from the sub-trees rooted at each of the children of u and we need to show it is always possible to choose a total of t vertices in this way.

Lemma 3.8 (Procedure *Select* is well defined)**.** *Algorithm 4 can always select t vertices as long as $t \leq |\mathcal{T}_u| \pmod L$.*

Proof. In this algorithm we are selecting t vertices from $\mathcal{T}_u$ by selecting u itself, and at most $|\mathcal{T}_{u_i}| \pmod L$ from each u_i. We thus have to prove that $\sum_i |\mathcal{T}_{u_i}| \geq t - 1$.

$$
\begin{aligned}
|\mathcal{T}_u| &= 1 + \sum_{u_i \in C_u} |\mathcal{T}_{u_i}| \\
\implies |\mathcal{T}_u| \pmod L &= 1 + \sum_{u_i \in C_u} (|\mathcal{T}_{u_i}| \pmod L) \\
\implies (|\mathcal{T}_u| - 1) \pmod L &= \sum_{u_i \in C_u} (|\mathcal{T}_{u_i}| \pmod L) \qquad (3.2)
\end{aligned}
$$

By assumption $t \leq |\mathcal{T}_u| \pmod L$ and therefore $t - 1 \leq (|\mathcal{T}_u| - 1) \pmod L$. Combining this with (3.2) we get

$$
\begin{aligned}
\sum_{u_i \in C_u} (|\mathcal{T}_{u_i}| \pmod L) &\geq t - 1 \\
\implies \sum_{u_i \in C_u} |\mathcal{T}_{u_i}| \geq \sum_{u_i \in C_u} (|\mathcal{T}_{u_i}| \pmod L) &\geq t - 1
\end{aligned}
$$

It follows that

$$\sum_{u_i \in C_u} |\mathcal{T}_{u_i}| \geq t - 1$$

which is what we needed. □

This shows that as long as $t \leq |\mathcal{T}_u| \pmod L$, *Select*($\mathcal{T}_u$, t) is well-defined.

It follows that the selection of $\mathcal{I}_v$ is always valid.

Lemma 3.9. *If $t_1 \leq t_2 \leq |\mathcal{T}_u| \pmod L$, Select($\mathcal{T}_u$,$t_1$) $\subseteq$ Select($\mathcal{T}_u$,t_2), for all $u \in \mathcal{T}$. This essentially means that the selection of vertices happens in a specific order, independent of when the procedure is called.*

Proof. There is a well-defined order in which we are selecting vertices in Algorithm 4, because of the fact that the set of children for each vertex is ordered. The lemma follows directly from this fact. □

We now prove a lemma from which it will follow that the size of $\mathcal{I}_u \backslash \mathcal{M}$(which is the set of vertices we wish to mark before moving on to the next recursive instance) is exactly 0 (mod L), except when u is the root.

Lemma 3.10. *$\mathcal{I}_u \cap \mathcal{T}_{u_i} \subseteq \mathcal{I}_{u_i}$ for all $u \in \mathcal{T}$ and $u_i \in C_u$. The part of the set of interest of u which is in a subtree rooted at one of the children u_i of u is a subset of the set of interest of u_i.*

Proof. Observe that

$$\mathcal{I}_u \cap \mathcal{T}_{u_i} = Select(\mathcal{I}_{u_i}, |\mathcal{I}_{u_i}| \pmod L)$$

Furthermore, from the way Algorithm 4 works, if $v \in Select(\mathcal{I}_{u_i}, |\mathcal{I}_{u_i}| \pmod L)$ we have the following cases for v:

- **Case 1:** $v = u_i$
  Clearly, $v \in \mathcal{I}_{u_i}$ by the definition of $\mathcal{I}_{u_i}$.
- **Case 2:** $v \in Select(\mathcal{T}_w, t)$ for some $w \in C_{u_i}$ and $t \leq |\mathcal{T}_w| \pmod L$
  By lemma 3.9 $Select(\mathcal{T}_w, t) \subseteq Select(\mathcal{T}_w, |\mathcal{T}_w| \pmod L)$ and by the definition of $\mathcal{I}_{u_i}$, $Select(\mathcal{T}_w, |\mathcal{T}_w| \pmod L) \subseteq \mathcal{I}_{u_i}$. Therefore we have

  $$v \in Select(\mathcal{T}_w, t) \subseteq Select(\mathcal{T}_w, |\mathcal{T}_w| \pmod L) \subseteq \mathcal{I}_{u_i}$$

  Therefore $v \in \mathcal{I}_{u_i}$

We have thus shown that $v \in \mathcal{I}_u \cap \mathcal{T}_{u_i} \implies v \in \mathcal{I}_{u_i}$ which gives us the desired result. □

We are now ready to show that in *step 2* of Algorithm 5, $|\mathcal{I}_u \backslash \mathcal{M}| \equiv 0 \pmod{L}$ for all $u \in \mathcal{T} \backslash \{r\}$. We need to show this to prove that the algorithm does not use more centers than the optimal solution.

Lemma 3.11. *When ProcessTree($\mathcal{T}_u$, $\mathcal{M}$) is called in Algorithm 5 for some $u \neq r$, we have $|\mathcal{I}_u \backslash \mathcal{M}| \equiv 0 \pmod{L}$.*

Proof. By the algorithm, when we call ProcessTree($\mathcal{T}_u$, $\mathcal{M}$), $\mathcal{T}_u \cap \mathcal{M} = \mathcal{T}_u \cap \mathcal{I}_{p_u}$, where p_u is the parent of u in $\mathcal{T}$. By lemma 3.10 $\mathcal{T}_u \cap \mathcal{I}_{p_u} \subseteq \mathcal{I}_u$ and therefore $|\mathcal{T}_u \cap \mathcal{I}_{p_u}| = |\mathcal{I}_u \cap \mathcal{I}_{p_u}|$. Therefore we have:

$$\begin{aligned}
|\mathcal{I}_u \backslash \mathcal{M}| &= |\mathcal{I}_u| - |\mathcal{I}_u \cap \mathcal{M}| \\
&= |\mathcal{T}_u| \pmod{L} - |\mathcal{I}_u \cap \mathcal{I}_{p_u}| \\
&= |\mathcal{T}_u| \pmod{L} - |Select(\mathcal{T}_u, |\mathcal{T}_u| \pmod{L})| \\
&= |\mathcal{T}_u| \pmod{L} - |\mathcal{T}_u| \pmod{L} \\
&= 0 \pmod{L}
\end{aligned}$$

Therefore $|\mathcal{I}_u \backslash \mathcal{M}| \equiv 0 \pmod{L}$ □

Corollary 3.12. *Our algorithm opens exactly $\left\lceil \frac{|\mathcal{G}|}{L} \right\rceil$ centers.*

Proof. We know that $|\mathcal{I}_r| \equiv |\mathcal{T}| \pmod{L}$ and therefore $|\mathcal{T} \backslash \mathcal{I}_r| \equiv 0 \pmod{L}$. It follows from lemma 3.11 that we open exactly $\frac{|\mathcal{T} \backslash \mathcal{I}_r|}{L}$ vertices to accommodate the vertices in $\mathcal{T} \backslash \mathcal{I}_r$ and it follows from the algorithm that we open $\left\lceil \frac{|\mathcal{I}_r|}{L} \right\rceil$ to accommodate the vertices in $\mathcal{I}_r$. So the number of centers we open is:

$$\frac{|\mathcal{T} \backslash \mathcal{I}_r|}{L} + \left\lceil \frac{|\mathcal{I}_r|}{L} \right\rceil = \left\lceil \frac{|\mathcal{T}|}{L} \right\rceil = \left\lceil \frac{|\mathcal{G}|}{L} \right\rceil$$

□

Observe that the optimal assignment must also open at least $\left\lceil \frac{|\mathcal{G}|}{L} \right\rceil$ vertices and therefore our algorithm never exceeds the number centers to open. We now show that the selection of set S in *step 4* of Algorithm 5 is valid.

Lemma 3.13. *For every vertex $u \in \mathcal{T}$, $|C_u| \geq \left\lceil \frac{|\mathcal{I}_u|}{L} \right\rceil$*

Proof. Let $|C_u| = l$. We know from the definition of $\mathcal{I}_u$ that it consists of at most $L-1$ vertices from each $\mathcal{T}_{u_i}, u_i \in C_u$ apart from u itself. It follows that:

$$\begin{aligned}
& |\mathcal{I}_u| \leq 1 + |C_u|(L-1) \\
\implies & \frac{|\mathcal{I}_u|}{L} \leq \frac{1}{L} + |C_u| - \frac{1}{L} = C_u \\
\implies & \frac{|\mathcal{I}_u|}{L} \leq C_u
\end{aligned}$$

Since C_u is an integer, $C_u \geq \frac{|\mathcal{I}_u|}{L} \implies C_u \geq \left\lceil \frac{|\mathcal{I}_u|}{L} \right\rceil$. This is the desired result. □

It follows directly that in *step 4* of Algorithm 5, $t \leq |C_u|$ and therefore the set S is valid in that step. We now show that each vertex has been assigned to a center at a distance of at most L.

Next we show that it is indeed an L-approximation.

Lemma 3.14. *Every $v \in \mathcal{I}_u$ is at a distance of at most $L-1$ from u for every $u \in \mathcal{T}$.*

Proof. $\mathcal{I}_u$ has at most $L-1$ vertices from each $u_i \in C_u$. Therefore, clearly no $v \in \mathcal{I}_u \cap u_i$ can be at a distance more than $L-1$ from u. □

Corollary 3.15. *Every $u \in \mathcal{I}_v$ is at a distance of at most L from $u_i \forall u_i \in C_v$.*

This directly follows from lemma 3.14 due to triangle inequality.

Lemma 3.16. *No vertex is chosen more than once as a center.*
Proof. In the recursive instance when we are solving the subtree rooted at u, we only open the children of u as centers. So when we move on to the next recursive instance, say $\mathcal{T}_v$, only v could possibly have been chosen as a center in any of the previous instances. In particular, when we are solving $\mathcal{T}_v$ for any v, no vertex in C_v could have been chosen as a center in any of the previous recursive instances. Since now the algorithm will choose new centers only among C_v, no vertex will be chosen more than once as a center. □

Note that it is easy to see that every vertex is marked at termination. The way the algorithm works, we solve $\mathcal{T}_v$ for every $v \in \mathcal{T}$ at some point. And when we are finished with solving $\mathcal{T}_v$, v is always marked. So eventually, every vertex will be marked.

Note also that the algorithm is clearly polynomial time. The reduction to the unweighted problem is polynomial time due to an upper bound on the value of τ(the diameter of the graph is an upper bound on τ), which enables us to do the binary search in logarithmic time.

Observation 3.17. *We have therefore established the following facts:*

- *Our algorithm does not choose more centers than the optimal assignment.*
- *We assign every vertex to center at a distance of at most L from it.*
- *Every vertex is marked at termination.*
- *The algorithm is polynomial time.*

Theorem 3.1 follows from Observation 3.17. This completes the proof of the result.

3.5 Future Directions

In our algorithm, we are only using a sparse subgraph of the unweighted graph that we had. One of the things that we are considering is whether we can somehow use this idea and consider all the edges of the unweighted graph to get a better approximation ratio, perhaps a 5-approximation? Another possible direction could be to consider whether we can use the fractional openings given by the LP, and use them intelligently along with our idea to get a better approximation. The aim now is to work towards these directions to try and get a constant factor approximation for a constant less than 6.

Acknowledgments

I would sincerely like to thank Ashkan Norouzi-Fard for guiding me in the project through the whole semester, giving a lot of useful and important advice on writing the report and preparing presentations and for many interesting and useful discussions from which I learnt a lot. Thanks to Laurent Feuilloley for the interesting discussions through the semester on various topics and for his comments on the report. I would like to thank Prof. Ola Svensson for giving me the opportunity to do this project and for his constant guidance and motivation throughout the project.

Bibliography

[1] Vladamir Marianov and H. A. Eiselt, editors. *Foundations of Location Analysis*. Springer US, 2011. ISBN 978-1-4419-7571-3.

[2] J. J. Sylvester. A Question in the Geometry of Situation. *Quarterly Journal of Pure and Applied Mathematics*, 1, 1857.

[3] Nimrod Megiddo. Linear-time algorithms for linear programming in R3 and related problems. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:329–338, 1982. ISSN 0272-5428. doi: 10.1109/sfcs.1982.74. URL http://dx.doi.org/10.1109/sfcs.1982.74.

[4] S. L. Hakimi. Optimum locations of switching centers and the absolute centers and medians of a graph. *Operations Research*, 12(3):450–459, 1964. doi: 10.1287/opre.12.3.450. URL http://dx.doi.org/10.1287/opre.12.3.450.

[5] Barbaros C. Tansel, Richard L. Francis, and Timothy J. Lowe. State of the artlocation on networks: A survey. part i: The p-center and p-median problems. *Management Science*, 29

(4):482–497, 1983. doi: 10.1287/mnsc.29.4.482. URL http://dx.doi.org/10.1287/mnsc.29.4.482.

[6] Harold W. Kulin and Robert E. Kuenne. An efficient algorithm for the numerical solution of the generalized weber problem in spatial economics. *Journal of Regional Science*, 4 (2):21–33, 1962. ISSN 1467-9787. doi: 10.1111/j.1467-9787.1962.tb00902.x. URL http://dx.doi.org/10.1111/j.1467-9787.1962.tb00902.x.

[7] Hua Lo-Keng. Application of mathematical methods to wheat harvesting. *Chinese Mathematics*, 2:77 – 91, 1962. ISSN 0196-6774.

[8] Moses Charikar, Sudipto Guha, va Tardos, and David B. Shmoys. A constant-factor approximation algorithm for the k-median problem. *Journal of Computer and System Sciences*, 65 (1):129 – 149, 2002. ISSN 0022-0000. doi: http://dx.doi.org/10.1006/jcss.2002.1882. URL http://www.sciencedirect.com/science/article/pii/S0022000002918829.

[9] Zvi Drezner. The p-centre problem-heuristic and optimal algorithms. *The Journal of the Operational Research Society*, 35(8):pp. 741–748, 1984. ISSN 01605682. URL http://www.jstor.org/stable/2581980.

[10] D. S. Hochbaum and D. B. Shmoys. A Best Possible Heuristic for the k-Center Problem. *Mathematics of Operations Research*, 10:180–184, 1985. doi: 10.1287/moor.10.2.180.

[11] Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38(0):293 – 306, 1985. ISSN 0304-3975. doi: http://dx.doi.org/10.1016/0304-3975(85)90224-5. URL http://www.sciencedirect.com/science/article/pii/0304397585902245.

[12] J. Barilan, G. Kortsarz, and D. Peleg. How to allocate network centers. *Journal of Algorithms*, 15(3):385 – 415, 1993. ISSN 0196-6774. doi: http://dx.doi.org/10.1006/jagm.1993.1047. URL http://www.sciencedirect.com/science/article/pii/S0196677483710473.

[13] Marek Cygan, Mohammad Taghi Hajiaghayi, and Samir Khuller. LP rounding for k-centers with non-uniform hard capacities. *CoRR*, abs/1208.3054, 2012. URL http://arxiv.org/abs/1208.3054.

[14] Hyung-Chan An, Aditya Bhaskara, and Ola Svensson. Centrality of trees for capacitated k-center. *CoRR*, abs/1304.2983, 2013. URL http://arxiv.org/abs/1304.2983.

[15] Samir Khuller and Yoram J. Sussmann. The capacitated k-center problem. In *In Proceedings of the 4th Annual European Symposium on Algorithms, Lecture Notes in Computer Science 1136*, pages 152–166. Springer, 1996.