The Capacitated K-Center Problem[*]

Samir Khuller [†]
Dept. of Computer Science and UMIACS
University of Maryland
College Park, MD 20742

Yoram J. Sussmann [‡]
Dept. of Computer Science
University of Maryland
College Park, MD 20742

Abstract

The capacitated K-center problem is a fundamental facility location problem, where we are asked to locate K facilities in a graph, and to assign vertices to facilities, so as to minimize the maximum distance from a vertex to the facility to which it is assigned. Moreover, each facility may be assigned at most L vertices. This problem is known to be NP-hard. We give polynomial time approximation algorithms for two different versions of this problem that achieve approximation factors of 5 and 6. We also study some generalizations of this problem.

1. Introduction

The basic K-center problem is a fundamental facility location problem [17] and is defined as follows: given an edge-weighted graph $G = (V, E)$ find a subset $S \subseteq V$ of size at most K such that each vertex in V is "close" to some vertex in S. More formally, the objective function is defined as follows:

$$\min_{S \subseteq V} \max_{u \in V} \min_{v \in S} d(u, v)$$

where d is the distance function. For example, one may wish to install K fire stations and minimize the maximum distance (response time) from a location to its closest fire station. The problem is known to be NP-hard [8].

An approximation algorithm with a factor of ρ, for a minimization problem, is a polynomial time algorithm that guarantees a solution with cost at most ρ times the optimal solution.

[*]A preliminary version of this paper will appear in the Proceedings of the Fourth Annual European Symposium on Algorithms (ESA 1996).

[†]Research supported by NSF Research Initiation Award CCR-9307462, and NSF CAREER Award CCR-9501355. Email addr: `samir@cs.umd.edu`

[‡]Research supported by NSF CAREER Award CCR-9501355. Email addr: `yoram@cs.umd.edu`

Approximation algorithms for the basic K-center problem have been very well studied and are known to be optimal [7, 9, 10, 11]. These schemes present natural methods for obtaining an approximation factor of 2. Several approximation algorithms are known for interesting generalizations of the basic K-center problem as well [3, 10, 16]. The generalizations include cases when each node has an associated "cost" for placing a center on it, and rather than limiting the number of centers, we have a limited budget [10, 16]. Other generalizations include cases where the vertices have weights and we consider the weighted distance from a node to its closest center [3, 16].

Recently, a very interesting generalization that we call the capacitated K-center problem was studied by Bar-Ilan, Kortsarz and Peleg [1]. The input specifies an upper bound on the number of centers K, as well as a maximum load L. We have to output a set of at most K centers, as well as an assignment of vertices to centers. No more than L vertices may be assigned to a single center. Under these constraints we wish to minimize the maximum distance between a vertex u and its assigned center $\phi(u)$.

$$\min_{S \subseteq V} \max_{u \in V} d(u, \phi(u))$$

such that

$$|\{u \mid \phi(u) = v\}| \leq L \quad \forall v \in S$$

where

$$\phi : V \rightarrow S.$$

Bar-Ilan, Kortsarz and Peleg [1] gave the first polynomial time approximation algorithm for this problem with an approximation factor of 10. Various applications for capacitated centers were first mentioned in [14, 15]. A slightly different problem, where the radius is fixed, and one has to minimize the number of centers, shows up in the Sloan digital sky survey project [13].

1.1. Our Results

In Section 2 we discuss a simplification of the problem where a node may appear multiple times in S (i.e. more than one center can be put at a node). We will refer to this problem as the capacitated multi-K-center problem. By introducing some new ideas and using the basic approach proposed in [1], we are able to give a polynomial time algorithm that achieves an approximation factor of 5. In Section 3 we show how to solve the problem when we are allowed only one center at a vertex. The high level structure of the algorithm is the same, but the assignment of centers to vertices has to be done extremely carefully. This problem will be referred to as the capacitated K-center problem. For this version of the problem we obtain an approximation factor of 6. It is worth noting that in fact we prove that our solution is at most 6 times an optimal solution that is allowed to put multiple centers at a single vertex. This clearly indicates that there is room for further improvement, since a better lower bound on the optimal should be possible when at most one center may be placed at a vertex.

The algorithm can be easily extended to the more general case when each vertex has a demand d_i, and multiple centers may be used to satisfy its demand. The total demand assigned to any center should not exceed L. Using the method in [1], we can obtain an approximation factor of 13 for the version with costs. (Each vertex has a cost for placing a center on it, and we are working with a fixed budget.) In Section 4 we study some other variants of this problem. Let a (c_1K, c_2L, c_3R) solution denote a solution using at most c_1K centers, each with a load of at most c_2L, which assigns every node to a center at distance at most c_3R, where R is the radius of the optimal solution. For any $x \geq 1$, we obtain a $\left(\frac{2}{c}K, cL, 2R\right)$ solution for the capacitated multi-K-center problem, and a $\left(\frac{2}{c}K, cL, 4R\right)$ solution for the capacitated K-center problem, where $c = \frac{x+1}{x}$.

2. Algorithm for Capacitated Multi-K-Centers

We first give a high-level description of the algorithm. We may assume for simplicity that G is a complete graph, where the edge weights satisfy the triangle inequality. (We can always replace any edge by the shortest path between the corresponding pair of vertices.)

High-Level Description

The algorithm uses the threshold method introduced by Edmonds and Fulkerson in [4] and used for the K-center problem by Hochbaum and Shmoys [9, 10]. Sort all edge weights in non-decreasing order. Let the (sorted) list of edges be $e_1, e_2, \ldots e_m$. For each i, let the threshold graph G_i be the subgraph obtained from G by including edges of weight at most $w(e_i)$. Run the algorithm below for each i from 1 to m, until a solution is obtained. (Hochbaum and Shmoys [10] suggest using binary search to speed up the computation. If running time is not a factor, however, it does appear that to get the best solution (in practice) we should run the algorithm for all i, and take the best solution.) In each iteration, we work with the subgraph G_i and view it as an unweighted graph. Since G_i is an unweighted graph, when we refer to the distance between two nodes, we refer to the number of edges on a shortest path between them. In iteration i, we find a solution using some number of centers. If the number of centers exceeds K, we prove that there is no solution with cost at most $w(e_i)$. If the number of centers is at most K, we show that the maximum distance in G_i between a vertex and its assigned center is at most five. This gives an approximation factor of 5.

First find a maximal independent set I in G_i^2. (G_i^2 is the graph obtained by adding edges to G_i between nodes that have a common neighbor.) This technique was introduced by Hochbaum and Shmoys [9, 10] and has been used extensively to solve K-center problems. We refer to a node in the maximal independent set as a *monarch*. The algorithm also constructs a "tree" of monarchs which will be used to assign vertices to centers. There are two key differences between our algorithm and the one presented in [1]:

1. We use a specific procedure to find a maximal independent set, as opposed to selecting an arbitrary maximal independent set.
2. We deal with all monarchs uniformly rather than dealing with the light and heavy[1] monarchs separately as in [1].

[1]These terms will be explained shortly.

Each monarch has an empire that consists of a subset of vertices within the immediate neighborhood of the monarch in G_i^2. When a monarch is added to the maximal independent set, all such vertices that do not currently belong to an empire are added to this monarch's empire. The algorithm also constructs a tree of monarchs as the monarchs are selected. This tree has the property that an edge in the tree corresponds to two monarchs whose distance in G_i is *exactly* three. Each monarch then tries to collect a domain of size L — a subset of vertices that are close to the monarch and assigned to it. In doing so, a monarch may grab vertices from other empires if none are available in its own empire. After this process is complete there may still be unassigned vertices. We then use the tree of monarchs to assign new centers to handle the unassigned vertices. Nodes that are left unassigned in a particular empire may be assigned to the parent monarch. Eventually, we will put additional centers at the monarch vertices (recall that more than one center may be located at a single vertex).

```
CAPACITATED-CENTERS ($G = (V, E), K, L$).
1  Sort all edges in non-decreasing weight order $(e_1, \ldots, e_m)$.
2  **for** $i = 1$ **to** $m$ **do**
3      Let $G_i = (V, E_i)$ where $E_i = \{e_1, \ldots, e_i\}$.
4      Unmark all vertices.
5      **if** ASSIGNCENTERS($G_i$) then **exit**.
6  **end-proc**
```

```
ASSIGNCENTERS ($G_i$).
1  SUCCESSFUL = **true**
2  Let $n_i^c$ be the number of vertices in connected component $G_i^c$.
3  **if** $(\sum_c \lceil \frac{n_i^c}{L} \rceil) > K$ **then** SUCCESSFUL = **false**
   **else**
4    **for** each connected component $G_i^c$ of $G_i$ **do**
5        ASSIGNMONARCHS($G_i^c$).
6        ASSIGNDOMAINS($G_i^c$).
7        REASSIGN($G_i^c$).
8    **if** total centers used $> K$ **then** SUCCESSFUL = **false**
9  **return** (SUCCESSFUL)
10 **end-proc**
```

ASSIGNCENTERS (G_i) tries to assign centers within each connected component. Each component is dealt with separately. (This can be done because if there is an optimal solution with maximum radius $w(e_i)$, then no center in the optimal solution will be assigned vertices that belong to different connected components of G_i.) A lower bound on the number of centers in each component is $\lceil \frac{n_c}{L} \rceil$, where n_c is the number of vertices in G_i^c. If the number of allowed centers is smaller than $\sum_c \lceil \frac{n_c}{L} \rceil$ then there is no solution.

ASSIGNMONARCHS(G_i^c) assigns monarchs (nodes in the independent set) in a BFS manner. After we put a vertex in the independent set, we mark all unmarked nodes within distance two in G_i. To pick a new vertex to add to the independent set, we pick an unmarked vertex that is adjacent to a marked vertex. Rather than describing the algorithm in terms of $(G_i^c)^2$ we

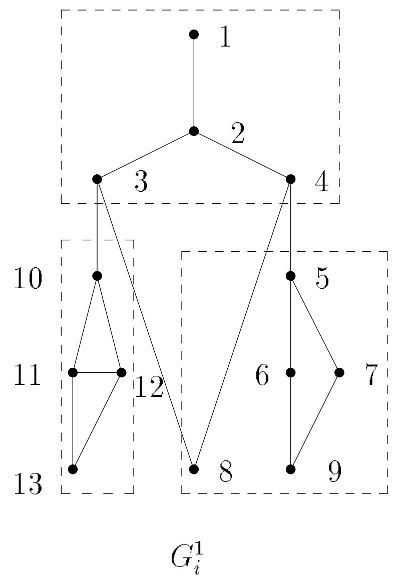

Figure 1: Example to show execution of AssignMonarchs.

work with G_i^c, to separate the nodes in a monarch's empire into level-1 and level-2 nodes. The level-1 nodes are adjacent to the monarch, and the level-2 nodes are at distance two from the monarch. We define $E_1(v)$ and $E_2(v)$ to be the level-1 and level-2 nodes, respectively, in v's empire. Thus the empire of v is $E_1(v) \cup E_2(v)$.

```
AssignMonarchs($G_i^c$).
1  Pick an arbitrary vertex $v \in G_i^c$ and set $Q = \{v\}$.
2  $p(v) = $ nil.
3  while $Q$ has unmarked nodes do
4          Remove an unmarked node $v$ from $Q$.
5          Make $v$ a monarch and mark it.
6          for all $u \in Adj(v)$ do
7              if $u$ is unmarked then
8                  Add $u$ to $E_1(v)$ and mark $u$
9          for all $u \in Adj(v)$ do
10             for all $w \in Adj(u)$ do
11                 if $w$ is unmarked then
12                     Add $w$ to $E_2(v)$ and mark $w$
13         for all $u \in E_2(v)$ do
14             for all $w \in Adj(u)$ do
15                 if $w$ is unmarked and $w \notin Q$ then
16                     Set $p(w) = v$ and add $w$ to $Q$
17 end-proc
```

Before discussing the pertinent properties of AssignMonarchs, we show its execution on the simple example given in Fig. 2. The algorithm starts from vertex 1, which is made a monarch. Vertex 2 is added to $E_1(1)$ (level 1 in its empire) and vertices 3 and 4 are added to $E_2(1)$. Q currently contains vertices 5, 8 and 10. Vertex 5 is then chosen as a monarch and vertices 6 and 7 are added to $E_1(5)$. Vertices 8 and 9 are both added to $E_2(5)$. Q now contains vertices 8, 10, 14 and 16. The next vertex chosen from Q is 10 since 8 is marked. Vertices 11 and 12 are added to $E_1(10)$, and vertex 13 is added to $E_2(10)$. Q now contains vertices 8, 14 and 16. Vertex 14 is now chosen from Q, and vertices 15 and 16 are added to $E_1(14)$ and $E_2(14)$, respectively. The algorithm stops since there are no more unmarked vertices in Q.

There are a few important properties of the monarchs produced by algorithm Assign-Monarchs.

Important Properties:

1. The distance between any two monarchs is at least three.
2. The distance between a monarch m (except for the root) and its "parent monarch" $p(m)$ in the tree is exactly three.
3. The distance between a monarch and any vertex in its empire is at most two.
4. Each monarch (except for the root) has at least one edge to a level two vertex in its parent monarch's empire. Moreover, each such level two vertex has only *one* such neighbor that is a monarch. More generally, *any* vertex can have at most *one* neighbor that is a monarch (Corollary of property 1).

Procedure AssignDomains tries to assign a domain of size at most L to each monarch. The objective is to assign as many vertices to a domain as possible subject to the following constraints.

1. A vertex may be assigned to a monarch's domain only if it is at distance at most two from the monarch.
2. A monarch may include in its domain a vertex from another empire only if each vertex in its empire belongs to some domain.

One way to implement this procedure is by finding a min cost maximum flow in an appropriate bipartite graph. The flow problem has a very simple structure and there are only two types of costs on edges (0 and 1).

```
AssignDomains($G_i^c$).
1  Let $M$ be the set of monarchs in $G_i^c$.
2  Let $E' = \{(m, v) \mid m \in M,\ v \in V$, distance from $m$ to $v$ is at most two $\}$.
3  Construct a bipartite graph $G' = (M, V, E')$.
4  Add vertices $s$ and $t$. Add edges $\{(s, m) \mid m \in M\}$ and $\{(v, t) \mid v \in V\}$.
5  For $m \in M$, $v \in V$, set capacities $u(s, m) = L$, $u(m, v) = 1$ and $u(v, t) = 1$.
```

```
6  Cost of edge $c(m, v) = 1$ if $v$ is not in $m$'s empire.
   Cost of all other edges is 0.
7  Compute a min-cost maximum integral flow in $G'$.
8  For each monarch $m$, set
   $domain(m) = \{v \mid v$ receives one unit of flow from $m$ in $G'\}$.
9  For each $v$, if $v \in domain(m)$ then define $\phi(v) = m$.
10 end-proc
```

Definition 1: *A* light *monarch is one that has a domain of size $< L$. A* heavy *monarch is one that has unassigned vertices in its empire. A* full *monarch is one that is neither heavy nor light.*

Lemma 2.1: *If m is a heavy monarch then each vertex in m's domain belongs to m's empire.*

Proof. Assume that there is a vertex u in m's domain that is not in m's empire. Let x be a vertex in m's empire that is not assigned to any domain. We can change the flow function in G' and send one unit of flow from m to x instead of m to u. This produces a max flow in G' of lower cost, a contradiction. □

If there is no heavy monarch, all vertices belong to a domain and the algorithm halts successfully.

Let K_L be the number of light monarchs. Let n_L be the number of vertices belonging to the domains of light monarchs, and let n be the total number of vertices.

Theorem 2.2: *The number of centers required is at least $K_L + \lceil (n - n_L)/L \rceil$.*

The proof is simpler than the proof given in [1]. The following lemmas were established in [1]. We repeat them for completeness.

Let $\mathcal{E}$ be the set of monarchs as defined in [1]. We repeat the definition here.

Let $\mathcal{E}_0$ be the set of light monarchs. Iteratively, add to $\mathcal{E}_0$ any monarch that contains a vertex in its domain that could have been assigned to a monarch in $\mathcal{E}_0$.

$$\mathcal{E}_j = \mathcal{E}_{j-1} \cup \{m \in M | \exists v \in V, \exists m' \in \mathcal{E}_{j-1},\ \phi(v) = m \text{ and } d(v, m') \leq 2 \text{ in } G_i\}$$

Let $\mathcal{E}$ be the largest set $\mathcal{E}_j$ obtained in this process. Let $\mathcal{F}$ be the set of remaining monarchs.

Lemma 2.3: *The set $\mathcal{E}$ does not contain any heavy monarchs.*

Proof. Suppose heavy monarch θ was added at iteration j. We can transfer a node v to a center θ' in $\mathcal{E}_{j-1}$. By a sequence of such transfers, we eventually reach a center in $\mathcal{E}_0$ which has at most $L - 1$ nodes in its domain, and can absorb the extra node. This corresponds to a higher flow, since the heavy monarch can absorb an unassigned node, a contradiction. □

Lemma 2.4: *Consider a center in an optimal solution that covers a monarch in $\mathcal{E}$. This center cannot be assigned any nodes that are not in the domains of monarchs in $\mathcal{E}$.*

Proof. Assume for contradiction that θ is a center in the optimal solution that covers both $e \in \mathcal{E}$ and u. If u does not belong to any domain, then since the distance from u to e in G_i is at most two, we can perform a sequence of transfers, eventually reaching a center in $\mathcal{E}_0$ which has at most $L-1$ nodes in its domain, and can absorb the extra node, resulting in a higher flow, a contradiction. If u belongs to the domain of monarch f, then since the distance from e to u in G_i is at most two, and the distance from u to f in G_i is at most two, it must be the case that $f \in \mathcal{E}$ as required by the lemma. $\square$

Proof. (Of Theorem 2.2) Each monarch in $\mathcal{E}$ is covered by a distinct center in the optimal solution, and these centers of the optimal solution cannot cover any other nodes in $\mathcal{F}$. Let $n_{\mathcal{E}}$ be the number of vertices in the domains of monarchs in $\mathcal{E}$. Then we need at least $|\mathcal{E}| + \lceil(\frac{n-n_{\mathcal{E}}}{L})\rceil$ centers. This is the same as $K_L + \lceil(\frac{n+(|\mathcal{E}|-K_L)\cdot L - n_{\mathcal{E}}}{L})\rceil$. Since $n_{\mathcal{E}} = (|\mathcal{E}| - K_L)\cdot L + n_L$ we get $K_L + \lceil(\frac{n-n_L}{L})\rceil$. $\square$

We will prove that this is also an upper bound on the number of centers we use. We now describe procedure ReAssign.

```
ReAssign($G_i^c$).
1  Let $M$ be the set of monarchs in $G_i^c$.
2  For each monarch $m \in M$, set
   $unassigned(m) = (\{m\} \cup E_1(m) \cup E_2(m)) \setminus (\cup_{u \in M} domain(u))$.
3  Let $T$ be the tree of monarchs in $G_i^c$.
4  for all nodes $m$ in $T$, set $passed(m) = \emptyset$.
5  while $T$ is not empty do
6        Remove a leaf node $m$ from $T$.
7        Let $|unassigned(m)| + |passed(m)| = k'L + \epsilon$
8        Allocate $k'$ new centers at monarch $m$ and assign $k'L$ of the nodes
         to them. For each such node $v$ we define $\phi(v) = m$.
9        Assign the $\epsilon$ remaining nodes to monarch $m$ and for each such
         node $v$ define $\phi(v) = m$, freeing up to $\epsilon$ nodes in $domain(m)$.
10       Add the freed nodes to $passed(p(m))$.
11       Delete $m$ from $T$.
12 end-proc
```

In practice, one would pass to the parent monarch $p(m)$ those nodes which are closest to $p(m)$.

Theorem 2.5: *Each vertex is assigned to a center whose distance in G_i is at most 5. Moreover, we use at most $K_L + \lceil(n-n_L)/L\rceil$ centers.*

Proof. All centers except possibly light monarchs cover L nodes by construction. The size of the domain of a light monarch does not decrease. Therefore the total number of centers used is at most $K_L + \lceil(n-n_L)/L\rceil$.

A node is either covered by the node from which it receives flow, or by the parent of its original monarch. In the former case, it is at distance at most two from the center that covers it. In the latter case the passed nodes are always covered by their monarch's parent, i.e., they are only passed once. Thus the distance from a node to the center that covers it is at most five (at most two to its original monarch, and three more to the parent monarch). ∎

3. Algorithm for K-Centers

We now consider the version where we are required to pick K distinct vertices as centers. We use the same high level approach as in the previous case, but need to pick the centers carefully. We are able to show that the algorithm obtains an approximation factor of 6. (Obtaining a factor of 7 using the previous approach is easy.)

The main difficulty lies in allocating centers to cover the vertices left unassigned by AssignDomains. We first introduce some new notation.

Nodes in a monarch's empire are called its subjects. In AssignMonarchs, each level-2 subject w of a monarch is brought in by a node u at distance 1 from the monarch. We define $link(w) = u$. Each monarch m (except the root) was placed into Q by a unique level-2 subject of its parent monarch. This node is called the *spouse* $s(m)$ of monarch m (Fig. 2). Note that each monarch has a unique spouse, and a node can be the spouse of at most one monarch (by property 4).

We need to be careful when allocating new centers to cover unassigned nodes. We require that: (1) a node can only be allocated as a center once; (2) monarchs have sufficient available nodes to allocate centers for the nodes passed to them. To ensure this we enforce the following rule. A monarch may allocate centers of the following types only:

1. Nodes in its empire, or
2. Nodes at distance 1 from itself (which may not be in its empire), as long as a monarch does not allocate its spouse as a center.

We define a *tree* $T(m)$ of height 2 corresponding to each monarch m. The root of $T(m)$ is monarch m. The leaves of this tree are all the level-2 subjects of m that are the spouse of some other monarch. For any leaf w, we make $link(w)$ the parent of the leaf. These nodes are the children of m in the tree $T(m)$.

In Fig. 3 we show a monarch m together with all the level-2 subjects of m that are the spouse of some other monarch (for example m'). For each leaf w, we also show $link(w)$. Notice that $link(w)$ may not be in monarch m's empire.

Observe that nodes that are the spouse of some monarch may belong to two trees. We therefore specify that a monarch may assign a center to any vertex in its tree $T(m)$ other than its spouse. This ensures that no vertex is assigned as two centers by two different monarchs.

Tree $T(m)$ will be used in assigning vertices that are passed to monarch m. Nodes passed from monarch m' to monarch m are covered by one of five nodes: The spouse of monarch

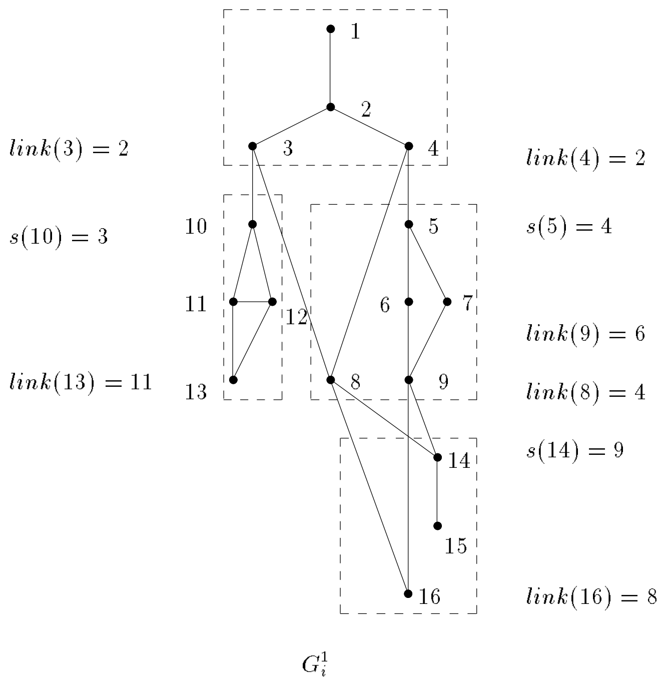

Figure 2: Example to illustrate links and spouses.

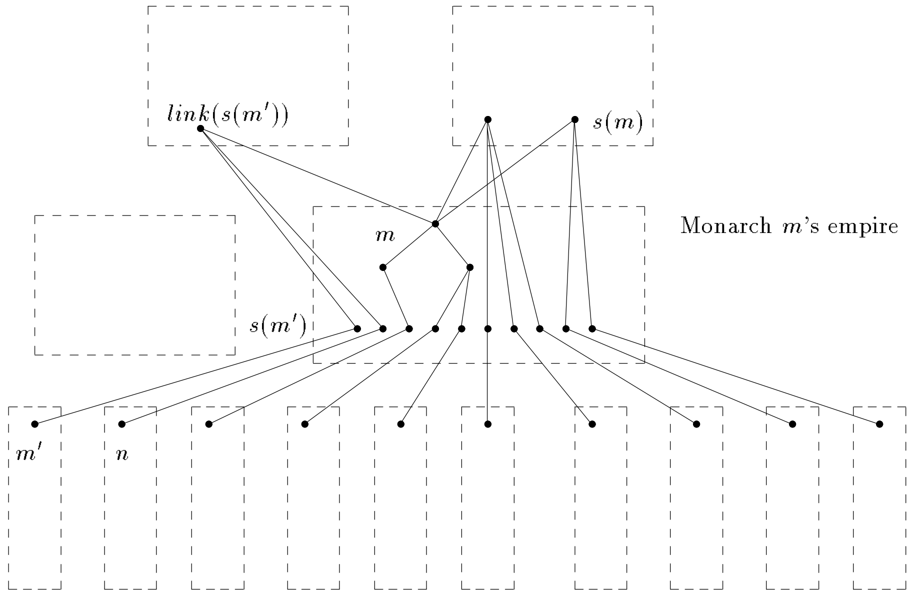

Figure 3: Example to illustrate tree $T(m)$ of a monarch m.

m', i.e., $s(m')$, the spouse's link $link(s(m'))$, the spouse of a sibling monarch $s(n)$ (where $p(n) = p(m') = m$ and $link(s(n)) = link(s(m'))$), the link of a sibling monarch's spouse $link(s(n))$ (where $p(n) = p(m') = m$), or monarch m.

A monarch does not allocate centers at nodes that are passed to it. Because of this, we may have to allocate centers at vertices that are already assigned to a center. We therefore specify that in this case the new center does not cover itself, but covers L other vertices. A vertex allocated as a center which is not assigned to a center covers itself as well as $L - 1$ other vertices.

The algorithm given in this section differs from that in Section 2 in the selection of new centers to cover the vertices left unassigned by AssignDomains. We first give a high-level description of the new selection scheme and then a new Reassign procedure that implements the scheme.

High-Level Description

We repeatedly select a leaf monarch in the tree of monarchs and allocate centers to cover nodes in its empire as well as nodes passed to it from its children monarchs. Let m be the monarch currently under consideration. If m' is a child monarch of m, we will assume that m' passes the excess nodes in its empire to m. Each leaf node $s(m')$ in $T(m)$ is labelled with the number of excess nodes that monarch m' is passing to m.

Nodes passed to m are assigned to centers placed on nodes in $T(m)$. There are a few important things to note here. (1) When we begin to process monarch m, no centers are currently placed at any nodes in $T(m)$ (except for m itself). (2) Monarch m is responsible for allocating centers for all the nodes that are passed to it from its children monarchs. (3) Monarch m is responsible for the free nodes in its empire. However, some of the free nodes at distance 2 from m may belong to trees of other monarchs, and may have centers already placed on them, in which case we will assume they are assigned to their own centers. If a vertex at distance 2 is in monarch m's domain, and a center is placed on it by the tree it belongs to, then it remains in m's domain and does not change its assignment.

Monarch m first assigns the nodes that are passed to it from children monarchs, using $T(m)$ to place full centers. Any nodes that are left over (at most $L - 1$), that were not assigned, are assigned to monarch m, displacing vertices that were in m's domain, which become unassigned. At this stage there may be many free nodes in m's empire – nodes that were never part of a domain, as well as the nodes that were recently displaced from m's domain. Note that the nodes that were never part of a domain do not have a center on them, while the ones that were displaced from m's domain could have centers placed on them (since they may belong to other trees and may have been chosen as centers). However, there are at most $L - 1$ of these, so any which do not get assigned within m's empire can be passed to m's parent monarch. We now choose centers from the set of nodes that never belonged to any domain. In doing so, we may assign some of the displaced nodes as well. The remaining unassigned vertices, including the displaced nodes, are passed to m's parent monarch.

We now describe in detail how the passed vertices are handled. Group the leaves of $T(m)$, placing leaves u and v in the same group iff $link(u) = link(v)$. We process the groups in turn, processing the group whose common link is $s(m)$ last, if such a group exists.

We assign passed nodes to centers by processing the groups in order. We process a leaf node in a group as follows: We start by adding the vertices passed to the leaf node to a list called *pending*. Whenever *pending* has at least L vertices, we create a center and assign vertices to it. There are two things we have to be careful about – if we create a center at a vertex that is free, we have to assign the vertex to its own center. If a center is going to be assigned vertices from different groups we move the center up one level, from a leaf node to the link node. Centers chosen in the last group are not assigned any vertices from other groups, and so we never assign a center at $s(m)$, but only at leaf nodes in this group.

To ensure that nodes in $T(m)$ do not have centers placed on them, we process the monarchs in T in the reverse of the order in which they were placed in T. Note that if a node v in $T(m)$ belongs to the empire of another monarch m', then m' must have been placed in T before m, otherwise m would have placed v in its own empire. We thus process m before m'. If a center is placed at v by m then v is assigned to itself in case it was free. When we eventually process m', we are guaranteed that if v is free, it does not have a center placed on it.

For the last group, we proceed as above, except that any nodes carried over from the last group are picked up by monarch m, possibly replacing some nodes already assigned to m. These replaced nodes are either passed or allocated a center in $E_1(m) \cup E_2(m)$ by monarch m. (Note that if monarch m is light, then the nodes are passed, if not then it does not grab nodes from other empires, so it is safe to allocate centers at/for them).

Example

Before describing the pseudo-code, we discuss the example given in Fig. 4 in detail. We process the leaves from left to right. Each leaf is labeled with the number of vertices that are passed to it from the corresponding child monarch. Assume that L is 10. After we process u_1, $pending(m)$ has size 4, and no centers are allocated. When we process u_2, pending has size 12. Since we can allocate a full center at u_2, we do so. Since u_2 is free, we assign u_2 to itself and assign 9 ($= L - 1$) vertices from $pending(m)$ to u_2. The size of $pending(m)$ is now 3. In processing u_3, we add 2 more vertices to $pending(m)$. Before we process the leaves in v_2's group, we set $X = v_1$. Observe that vertices passed to nodes in v_1's group are going to share a center with vertices passed to nodes in v_2's group, hence we "promote" the center one level up. When we process u_4 and u_5, we add 3 more vertices to $pending(m)$ that now has size 8. We then process u_6, adding 8 vertices to $pending(m)$. Since we can allocate a full center, we allocate a center at v_1 (current value of X). Since v_1 is currently unassigned, we assign v_1 to itself and assign 9 ($= L - 1$) vertices from $pending(m)$ to it. The size of $pending(m)$ is now 7. When we process u_7, we add 5 vertices to pending. Since we can allocate a full center, we create a center at u_7. Since u_7 is assigned, we assign 10 vertices from $pending(m)$ to it. This leaves 2 vertices in $pending(m)$ that are assigned to monarch m, possibly displacing other assigned vertices.

ReAssign(G_i^c).

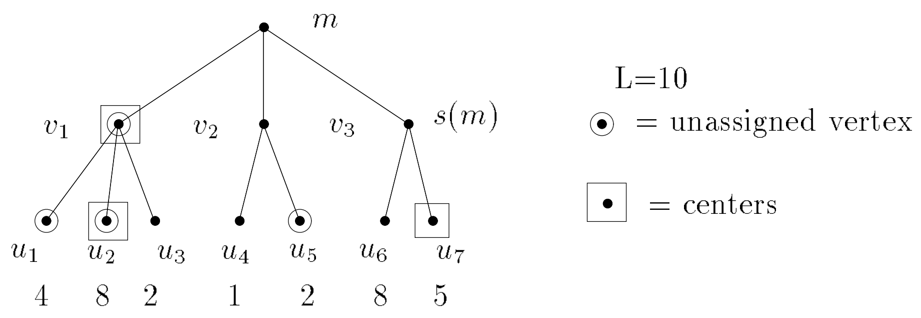

Figure 4: Example to illustrate assignment of centers in tree $T(m)$.

```
1  Let $M$ be the set of monarchs in $G_i^c$.
2  For each monarch $m \in M$, set
   $unassigned(m) = (\{m\} \cup E_1(m) \cup E_2(m)) \setminus (\cup_{u \in M} domain(u))$.
3  For each monarch $m \in M$, let $pending(m)$ be an ordered list, initially $\emptyset$.
4  Let $T$ be the tree of monarchs in $G_i^c$.
5  **for** all nodes $m$ in $T$ **do**
6      Define $T(m)$ such that the group containing $s(m)$ comes last.
7      **for** all leaf nodes $v$ in $T(m))$ **do** set $passed(v) = \emptyset$.
8  **for** all nodes $v$ in $G_i^c$, let $\chi(v) = \{v\}$ **iff** $v$ is unassigned and $\emptyset$ otherwise.
9  **for** all nodes $m$ in $T$, process them in reverse order of their insertion into $T$:
10     Set $X =$ **null**.
11     **for** all children $v$ of $m$ in $T(m)$ **do**
12         **for** all children $u$ of $v$ in $T(m)$ **do**
13             Append $passed(u)$ to $pending(m)$.
14             **if** $X =$ **null then** set $X = u$.
15             **if** $|\chi(X)| + |pending(m)| \geq L$ **then**
16                 Allocate a center at $X$ and assign $\chi(X)$ to it.
17                 Assign the first $L - |\chi(X)|$ nodes from $pending(m)$ to $X$ and
                   remove them from $pending(m)$.
18                 Set $X =$ **null**.
19             **else if** $X = u$ **then** set $X =$ **null**.
20         **if** $v \neq s(m)$ **and** $X =$ **null then**
21             **if** $|\chi(v)| + |pending(m)| = L$ **then**
22                 Allocate a center at $v$ and assign $\chi(v)$ to it.
23                 Remove all nodes from $pending(m)$ and assign them to $v$.
24             **else** set $X = v$.
25     Let $displaced(m) = |domain(m)| + |pending(m)| - L$.
26     Assign all nodes in $pending(m)$ to $m$,
       possibly displacing nodes in $domain(m)$.
27     Let $|unassigned(m)| + displaced(m) = k'L + \epsilon$.
28     Allocate $k'$ new centers at nodes in $unassigned(m)$.
```

```
29     Assign $k'L$ of the nodes to them, assigning nodes in $unassigned(m)$ first.
30     Add the $\epsilon$ remaining nodes to $passed(s(m))$.
31     Delete $m$ from $T$.
32 end-proc
```

Lemma 3.1: *Each node is allocated as a center at most once.*

Proof. A node v may be allocated as a center either by its monarch, or by the monarch m whose tree $T(m)$ it belongs to. If v's monarch allocates v as a center, then it must be the case that v is unassigned, which means v cannot be currently allocated as a center. If m allocates v as a center, then since m was processed before v's monarch, v could not have already been allocated as a center.

If v belongs to trees $T(m)$ and $T(m')$ for two different monarchs, then it must be the case that for one of the monarchs, say m', $v = s(m')$. Then m' will not allocate v as a center. $\square$

Lemma 3.2: *Each monarch m has sufficient available nodes in its tree $T(m)$ to allocate centers for nodes passed to it.*

Proof. Each monarch passes at most $L-1$ nodes to its parent. If monarch m has N children monarchs, then it must be the case that each child monarch has a unique spouse in $T(m)$. In addition, all these spouses are level-2 nodes in $T(m)$, so they are all available to allocate as centers. $\square$

Lemma 3.3: *Every node is assigned to a center.*

Proof. All nodes passed to a monarch m are assigned centers from $T(m) \cup \{m\}$. Unassigned nodes in m's empire are either assigned a center or passed. There are at most $L-1$ nodes displaced from $domain(m)$, hence they are either allocated a center from m's empire or passed to m's parent. $\square$

Theorem 3.4: *Each vertex is assigned to a center whose distance in G_i is at most 6. Moreover, we use at mose $K_L + \lceil (n - n_L)/L \rceil$ centers.*

Proof. All centers except possibly light monarchs cover L nodes by construction. The size of the domain of a light monarch does not decrease. Therefore the total number of centers used is at most $K_L + \lceil (n - n_L)/L \rceil$.

A node which is not passed is covered either by the monarch from which it receives flow or by a node in its monarch's empire. In the former case, it is at distance at most two from the center that covers it, and in the latter case it is at distance at most four from the center that covers it. A node which is passed from monarch m' to monarch m is covered by one of the following: (1) $s(m')$, (2) $link(s(m'))$, (3) $s(n)$ where $p(n) = p(m') = m$ and $link(s(n)) = link(s(m'))$, (4) $link(s(n))$ where $p(n) = p(m') = m$, or (5) monarch m. The distance bounds are as follows:

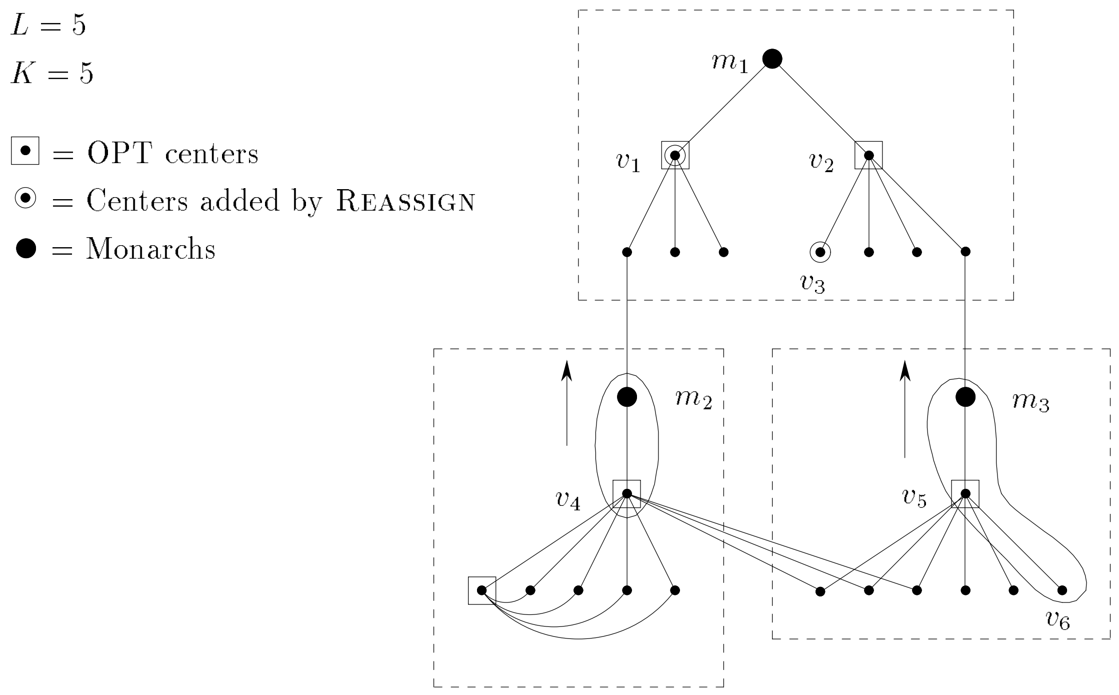

Figure 5: Example to show that the factor of 6 is tight.

Case	Distance
Case (1)	≤ 3
Case (2)	≤ 4
Case (3)	≤ 5
Case (4)	≤ 6
Case (5)	≤ 5

□

In Fig. 5 we give an example showing that the factor of 6 is tight for our algorithm. All edges in the example are edges in G_i. In the example, monarchs m_2 and m_3 pass 2 and 3 nodes, respectively, to their parent m_1. Monarch m_2 passes itself and v_4, and monarch m_3 passes itself, v_5 and v_6. The algorithm assigns all 5 passed nodes to a center which it places at v_1, leaving v_6 at a distance of 6 from its center. Vertex v_3 absorbs the remaining nodes in m_1's empire. It is clear, however, that OPT covers all nodes within distance 1.

Running Time

The bottleneck in the running time of this algorithm is the flow computation. If we use binary search in CAPACITATED-CENTERS, the algorithm computes $O(\log n)$ maximum flows.

3.1. Capacitated Centers with Costs

The capacitated K-center problem with costs is a generalization of the capacitated K-center problem where a cost function is defined on the vertices and the objective is to pick a set of centers whose total cost is at most K, such that the radius is minimized. (Note that this is equivalent to the weighted capacitated K-center problem in [1]. We use *cost* here to distinguish from weights as defined in, for example, [3, 16].) More formally, we are given a cost function $c : V \rightarrow \mathcal{R}^+$, and we add the constraint $\sum_{v \in S} c(v) \leq K$ to the statement of the capacitated K-center problem.

Bar-Ilan, Kortsarz and Peleg gave the first polynomial time approximation algorithm for this problem with an approximation factor of 21. Their technique, which involves finding a minimum-cost perfect matching in a bipartite graph, generalizes to finding a $2\rho + 1$ solution given a ρ-approximation algorithm for the capacitated K-centers problem. It therefore yields an approximation algorithm with an approximation factor of 13 when combined with our algorithm for capacitated K-centers.

4. Remarks

It is possible to improve the quality of the approximation if one is willing to allow some slack on the number of centers used and the maximum load. Let a (c_1K, c_2L, c_3R) solution denote a solution using at most c_1K centers, each with a load of at most c_2L, which assigns every node to a center at distance at most c_3R, where R is the radius of the optimal solution. Thus the algorithms given above obtain a $(K, L, 5R)$ solution to the capacitated multi-K-center problem and a $(K, L, 6R)$ solution to the capacitated K-center problem.

For the capacitated multi-K-center problem, we can obtain for any $x \geq 1$ a $\left(\frac{2}{c}K, cL, 2R\right)$ solution, where $c = \frac{x+1}{x}$. For example, when $x = 1$, this gives a $(K, 2L, 2R)$ solution. We can also achieve a $(2K, L, 2R)$ solution to this problem by always allocating $\left\lceil \frac{unassigned(m)}{L} \right\rceil$ extra centers in each empire.

Tha algorithm works as follows. We modify Reassign to overload centers in some empires and allocate extra centers in others. Specifically, let $unassigned(m) = k'L + \epsilon$. If $\epsilon \leq \frac{L}{x}$, then allocate k' additional centers at monarch m and use the centers at m to cover all nodes in $unassigned(m)$. Clearly, no center at monarch m has to cover more than $\frac{x+1}{x}L$ nodes in this case. If $\epsilon > \frac{L}{x}$, then allocate $k' + 1$ additional centers at monarch m and use them to cover all nodes in $unassigned(m)$. This may cause us to use more than K centers, since we allocate new lightly loaded centers at the end. We show below that we use at most $\frac{2x}{x+1}K$ centers.

Theorem 4.1: *For any $x \geq 1$, the algorithm gives a $\left(\frac{2}{c}K, cL, 2R\right)$ solution, where $c = \frac{x+1}{x}$.*

Let the sets $\mathcal{E}$ and $\mathcal{F}$ be defined as before. Let $\mathcal{X}$ be the set of monarchs at which an extra center was allocated to cover $\epsilon > \frac{L}{x}$ nodes. Let C_1 and C_2 be the number of centers used in the optimal solution to cover nodes in empires in $\mathcal{E}$ and $\mathcal{F}$, respectively. Clearly, any extra centers we use must be allocated at monarchs in $\mathcal{F}$. Let $\mathcal{S}_{\mathcal{L}}$ and $\mathcal{S}_{\mathcal{F}}$ denote the sets of nodes in the domains of monarchs in $\mathcal{E}$ and $\mathcal{F}$, respectively. Let f be the number of full centers allocated by the algorithm that cover nodes in $\mathcal{S}_{\mathcal{F}}$.

Lemma 4.2: *The number of centers allocated by our algorithm is at most $C_1 + f + |\mathcal{X}|$.*

Proof. No additional centers are used by the algorithm to cover nodes in $\mathcal{S}_{\mathcal{L}}$. Therefore the algorithm uses $|\mathcal{E}| + f + |\mathcal{X}|$ centers. Because the monarchs form an independent set in G_i^2, the optimal solution must use at least $|\mathcal{E}|$ centers to cover nodes in $\mathcal{S}_{\mathcal{L}}$. Therefore $C_1 \geq |\mathcal{E}|$, implying the lemma. $\square$

Lemma 4.3: *A lower bound on C_2 is given by $C_2 \geq f + \frac{|\mathcal{X}|}{x}$.*

Proof. By Lemma 2.4, the centers in the optimal solution which cover nodes in $\mathcal{S}_{\mathcal{L}}$ cannot cover any nodes not in $\mathcal{S}_{\mathcal{L}}$. Therefore the optimal solution uses at least $\frac{|\mathcal{S}_{\mathcal{F}}|}{L}$ centers to cover nodes in $\mathcal{S}_{\mathcal{F}}$. Since the lightly loaded centers allocated by our algorithm to cover nodes in $\mathcal{S}_{\mathcal{F}}$ each cover at least $\frac{L}{x}$ nodes, it follows that $fL + \frac{|\mathcal{X}|L}{x} \leq |\mathcal{S}_{\mathcal{F}}|$. Therefore $C_2 \geq \frac{|\mathcal{S}_{\mathcal{F}}|}{L} \geq f + \frac{|\mathcal{X}|}{x}$. $\square$

Proof. (Of Theorem 4.1) By lemma 4.2, the number of centers used by our algorithm is at most

$$\begin{aligned}
C_1 + f + |\mathcal{X}| &\leq C_1 + \frac{1+x}{1+x}f + \frac{1+x}{1+x}|\mathcal{X}| \\
&\leq C_1 + \frac{1+x}{1+x}f + \frac{x-1}{1+x}|\mathcal{X}| + \frac{2}{1+x}|\mathcal{X}| \\
&\leq C_1 + \frac{1+x}{1+x}f + \frac{x-1}{1+x}f + \frac{2}{1+x}|\mathcal{X}| \qquad \text{(since } |\mathcal{X}| \leq f) \\
&\leq C_1 + \frac{2x}{1+x}(f + \frac{|\mathcal{X}|}{x}) \\
&\leq C_1 + \frac{2x}{1+x}C_2 \qquad \text{(by Lemma 4.3)} \\
&\leq \frac{2x}{x+1}(C_1 + C_2) \qquad \text{(since } x \geq 1) \\
&\leq \frac{2x}{x+1}K. \qquad \text{(because the optimal solution does not overuse centers)}
\end{aligned}$$

$\square$

For the capacitated K-center problem, the same approach gives a $\left(\frac{2}{c}K, cL, 4R\right)$ solution.

Results of Lund and Yannakakis [12] and Feige [6] imply that no polynomial time $(c_1 K, c_2 L, (2-\epsilon)R)$ approximation algorithm is possible unless $NP \subseteq DTIME(n^{O(\log\log n)})$, since this would imply a constant-factor approximation algorithm for set cover.

Acknowledgements: We thank Robert Pless and Balaji Raghavachari for useful discussions. We thank the ESA referee for helpful comments.

References

[1] J. Bar-Ilan, G. Kortsarz and D. Peleg, "How to allocate network centers", *J. Algorithms*, 15:385–415, (1993).

[2] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*, The MIT Press, 1989.

[3] M. Dyer and A. M. Frieze, "A simple heuristic for the p-center problem", *Operations Research Letters*, Vol 3:285–288, (1985).

[4] J. Edmonds and D. R. Fulkerson, "Bottleneck extrema", *Journal of Combinatorial Theory*, Vol 8:299-306, (1970).

[5] T. Feder and D. Greene, "Optimal algorithms for approximate clustering", *Proc. of the* 20^{th} *ACM Symposium on the Theory of Computing*, pages 434–444, (1988).

[6] U. Feige, "A threshold of $\ln n$ for approximating set cover", *Proc. of the* 28^{th} *ACM Symposium on the Theory of Computing*, pages 314–318, (1996).

[7] T. Gonzalez, "Clustering to minimize the maximum inter-cluster distance", *Theoretical Computer Science*, Vol 38:293–306, (1985).

[8] M. R. Garey and D. S. Johnson, "Computers and Intractability: A guide to the theory of NP-completeness", *Freeman, San Francisco* (1978).

[9] D. Hochbaum and D. B. Shmoys, "A best possible heuristic for the k-center problem", *Mathematics of Operations Research*, Vol 10:180–184, (1985).

[10] D. Hochbaum and D. B. Shmoys, "A unified approach to approximation algorithms for bottleneck problems", *Journal of the ACM*, Vol 33(3):533–550, (1986).

[11] W. L. Hsu and G. L. Nemhauser, "Easy and hard bottleneck location problems", *Discrete Applied Mathematics*, Vol 1:209-216, (1979).

[12] C. Lund and M. Yannakakis, "On the hardness of approximating minimization problems", *Journal of the ACM*, Vol 41(5):960-981, (1994).

[13] R. Lupton, F. M. Maley, and N. E. Young, "Data collection for the Sloan digital sky survey – a network-flow heuristic", *Proc. of the* 7^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 296–303, (1996).

[14] H. L. Morgan and K. D. Levin, "Optimal program and data locations in computer networks", *Communications of the ACM*, Vol 20:315–322, (1977).

[15] K. Murthy and J. Kam, "An approximation algorithm to the file allocation problem in computer networks", *Proc. of* 2^{nd} *ACM Symposium on Principles of Database Systems*, pages 258–266, (1983).

[16] J. Plesnik, "A heuristic for the p-center problem in graphs", *Discrete Applied Mathematics*, Vol 17:263–268, (1987)

[17] C. Toregas, R. Swain, C. Revelle and L. Bergman, “The location of emergency service facilities”, *Operations Research*, Vol 19:1363–1373, (1971).