Efficient Recovery from Power Outage

(Extended Summary)

Sudipto Guha* Anna Moss† Joseph (Seffi) Naor ‡ Baruch Schieber §

Abstract

We study problems that are motivated by the real-life problem of efficient recovery from a wide scale electric power outage caused by a major disaster such as a hurricane or an equipment failure. In most of these cases an optimized scheduling of the workforce is required since the work crews on hand are not enough for immediate recovery of the whole network.

We model two variants of this problem: the budgeted problem, and the minimum weighted latency problem. We consider the problems for the general case as well as for two special cases: trees, and bipartite networks. All but one of the problems (the budgeted tree problem) are NP-Hard and the algorithms given for them are approximation algorithms. For the budgeted tree problem we give an optimal solution. Interestingly, the budgeted problem for bipartite networks is exactly the budgeted maximum set cover problem, for which we give the best ratio approximation algorithm.

---

*Computer Science Department, Stanford University, Stanford, CA 94305. E-mail: *sudipto@cs.stanford.edu.* Supported by IBM Cooperative Fellowship, an ARO MURI Grant DAAH04-96-1-0007 and NSF Award CCR-9357849, with matching funds from IBM, Schlumberger Foundation, Shell Foundation, and Xerox Corporation. Part of this work was done while this author visited IBM T.J. Watson Research Center.

†Computer Science Department, Technion, Haifa 32000, Israel. E-mail:*annaru@cs.technion.ac.il.*

‡Bell Laboratories, Lucent Technologies, 600 Mountain Ave., Murray Hill, NJ 07974. On leave from the Computer Science Department, Technion, Haifa 32000, Israel. E-mail: *naor@research.bell-labs.com.* Part of this work was done while this author visited IBM T.J. Watson Research Center.

§IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598.
E-mail: *sbar@watson.ibm.com.*

1 Introduction

Efficient and speedy recovery of electric power networks following a major outage, caused by a disaster such as extreme weather or equipment failure, is one of the main challenges facing electric power utility companies. These companies are tackling the problem in two levels. In the *planning level*, they are trying to design more reliable and robust networks, and in the *operational level*, they are attempting to manage the recovery optimally, in particular to manage their maintenance workforce in an efficient way. In this paper we are motivated by the operational level problem.

Most utility companies use a Trouble Call Management System (TCMS), which includes, along with other components, an assignment engine responsible for assigning the trouble calls to the available work teams at any given time. One of the problems that needs to be resolved by this assignment engine is the prioritization of the trouble calls, since in most cases the available workforce does not suffice for immediate recovery of the whole network. For example, a trouble call that involves a hospital or any other emergency service should receive higher priority than a call that involves a small number of residential customers. State of the art TCMS products such as the ones offered by GE Harris Energy Control Systems [LT96] and ASCADA [As] contain an assignment engine. However, most utility companies still resort to manual assignment and scheduling in cases of major outage.

1.1 The model and problems

We model the input by an undirected graph $G(V, E)$, where the set of vertices V is partitioned into three disjoint sets of vertices: *generator* vertices S, *relay* vertices R, and *customer* vertices C. As will be clear later, we may assume without loss of generality that the set S of generator vertices is a singleton set. Denote the generator vertex by s. Each relay vertex $r \in R$ is associated with a cost, $\mathsf{cost}(r)$, and each customer vertex $c \in C$ has a weight (or profit), $\mathsf{w}(c)$. We assume that the graph G is connected. Note that the paths from the generator vertex s to a customer may consist of both relay vertices and other customer vertices.

An outage occurs when at some point of time a subset $F \subseteq R$ of the relay vertices "fail". The subgraph left after the removal of these vertices (i.e., the graph induced by $V - F$) may contain several customer vertices that are disconnected from the generator s. Our goal is to reconnect these customer vertices by recovering some relay nodes from F.

We consider two variants of this problem. The first variant is a "one-time" efficient assignment of the workforce at hand, or maximizing the total importance of the customers recovered in a single day. That is, for a given "budget", the goal is to reconnect to the generator s a maximum weight subset of the customers. The second variant requires a recovery of the whole network, i.e., connecting all disconnected customers, which may be a lengthy process that takes several days. The goal is to minimize the weighted latency suffered by the customers, where this latency is scaled by the relative importance of each customer. We consider these problems for general networks as well as for tree networks and for bipartite networks.

These problems model the power outage recovery problem in a rather straightforward way. The failures causing the power outage are modeled by the failed relay vertices. The cost of each such relay vertex represents the estimated number of repairmen (or other resources) required to recover the corresponding failure in a day (or any other time unit). Each customer vertex corresponds to a customer of the utility company, and the weight of this vertex corresponds to the "importance" of this customer. Finally, the budget corresponds to the total number of repairmen available in a day (or any other time unit).

Our modeling of the power outage recovery problem captures the maximum coverage problem as a special case. In this problem, a set system with weights on the elements is given, and the goal is to cover a maximum weight subset of the elements using, say, k sets. This problem has been studied extensively in the literature since it models many applications arising in circuit layout, job scheduling, facility location, and other areas. (See [Ho97, pp. 135-138] and the references therein for examples of applications.)

In fact, the budgeted problem for bipartite networks can be viewed as a *budgeted maximum coverage problem* that introduces a more flexible model for the applications mentioned above. Consider, for example, the appliaction of locating k identical facilities so that the market share is maximized, introduced in [MZH83]. Namely, we are given clients with associated profits, situated in known locations. A client uses a facility if the facility is within a specified distance from the client. The goal is to locate k facilities so that the total profit of the clients served by these facilities is maximized. A variant of this aplication, considered in [BLF92, BBL95], is known as *optimal location of discretionary service facilities*. In this variant facilities gain profits associated with travel paths of customers; a facility covers a path if it is located in one of the nodes on the path, or at some vertex "close" to the path. Clearly, the above applications can be modeled by the unit cost maximum coverage problem. However, in practice, the cost of constructing a facility may depend on certain factors associated with the location of the facility. For example, each candidate site for constructing a facility is associated with some cost, and one is assigned a limited budget for constructing the facilities. This generalization of the problem of locating facilities to maximize market share is also discussed in [MZH83]. The budgeted maximum coverage problem is a model that allows us to handle this type of applications.

1.2 Results

All the problems that we consider, except for the budgeted tree problem, are NP-Hard. Therefore, we focus on polynomial time approximation algorithms. Below, we detail the results achieved.

General networks: For general networks we present an $O(\log^2 n)$-approximation algorithm for the budgeted reconnection problem, where n is the number of customers (or, in reality, customer clusters) that are disconnected from the generator s. Our result is a pseudo-approximation in the sense that the budget used by our algorithm can be at most double the budget used by the optimal solution to which our sloution is compared. For the minimum weighted latency problem, we first present a simple $O(\log^2 n)$-approximation algorithm. Next, we extend this algorithm to achieve an $O(\log n \log\log n)$ approximation factor.

Our solutions involve solving several instances of the minimum node weighted Steiner tree problem. This is an NP-hard problem, and the first approximation was given by Klein and Ravi [KR95]. To bound the performance of our algorithms, we need to bound the cost of the computed node weighted Steiner tree as a function of an optimal *fractional* solution which is derived from a linear relaxation of the problem. However, the algorithm of [KR95] bounds the cost of the approximate Steiner tree as a function of the optimal *integral* Steiner tree. We give a tighter analysis of the algorithm of Klein and Ravi and bound its performance as a function of the optimal fractional solution. This proof also suggests an alternative "sphere growing" algorithm for the node weighted Steiner tree problem.

Tree networks: Here, the graph G is a tree where the generator s is the root of the tree. We present a polynomial time optimal algorithm for the budgeted reconnection problem in tree networks in which the budget is polynomially bounded, and a strongly polynomial approximation scheme for arbitrary budgets. This is a bottom-up dynamic programming algorithm. We apply these algorithms to derive a logarithmic factor approximation algorithm for the minimum weighted latency problem. The latter problem can be shown to be strongly NP-Hard.

Bipartite networks: Here, the graph G is a three layered graph: the first layer is the generator s, the second layer is the set R of relay vertices which we assume is an independent set, and the third layer is the set C of customers. The budgeted problem in bipartite networks is a budgeted version of the maximum coverage problem. This is an NP-hard problem, and for the special case of this problem, where each set has unit cost, a $(1-\frac{1}{e})$-approximation algorithm was known, e.g., [Ho97, pp. 135-138]. Yet, no approximation algorithms were known for the budgeted, or general cost version of the problem. We give an $(1-\frac{1}{e})$-approximation algorithm for the budgeted maximum coverage problem, and argue that this approximation factor is the best possible, unless $NP \subseteq DTIME(n^{\log\log n})$. Similar to the tree networks case, we apply this algorithm to derive a logarithmic factor approximation algorithm for the minimum weighted latency problem for bipartite networks.

The rest of the paper is organized as follows. In the next section we give a formal statement of the problems considered. In Section 3 we give the approximation algorithms for general networks. In Sections 4 and 5 we briefly consider tree and bipartite networks. We conclude with some remarks and open problems.

2 Problem statement

We give here a formal statement of the problems.

The budgeted problem: Given a budget B, find a subset $F' \subseteq F$, where $\mathsf{cost}(F') \leq B$, such that the weight of the customer vertices reconnected to s through F' is maximized. More formally, the weight of the customer vertices that are connected to s in the subgraph induced by $V - (F - F')$, but disconnected from s in the subgraph induced by $V - F$, is maximized.

The minimum weighted latency problem: Given a budget B, find disjoint subsets $F'_1, \ldots, F'_k \subseteq F$ with the following properties:

1. The cost of each F'_i, $1 \leq i \leq k$, does not exceed the budget B.
2. All the customer vertices are connected to the generator vertex in the subgraph induced by $V - (F - \cup_{i=1}^{k} F'_i)$.

For $j = 1, \ldots, k$, let C_j be the set of customer vertices that are connected to s in the subgraph induced by $V - (F - \cup_{i=1}^{j} F'_i)$, but disconnected from s in the subgraph induced by $V - (F - \cup_{i=1}^{j-1} F'_i)$. Denote by $\mathsf{w}(C_j)$ the total weight of the customer vertices in C_j. Our goal is to find the subsets such that the sum $\sum_{j=1}^{k} j \cdot \mathsf{w}(C_j)$ is minimized.

Due to the fact that the problem requires an explicit schedule of repairs, we can assume that the cost of vertices are polynomially bounded. This assumption however will not be of concern to us in most of the problems discussed below.

It is not difficult to see that for the above problems, we may indeed assume without loss of generality, that only a single generator vertex exists, since we can always coalesce all the generator vertices into a single vertex. It can also be shown that we may assume without loss of generality that the sets R and C are disjoint.

To simplify the presentation, we make the following assumptions, all without loss of generality: (1) Ignore all relay vertices of cost more than B; (2) Treat all the customer vertices that remain connected after the removal of the set of failed relay vertices, as well as unfailed relay vertices, as relay vertices of zero cost; (3) Ignore the distinction between relay vertices and customer vertices. Instead, assume that the weights of the relay vertices and the costs of the customer vertices are set to zero.

The budgeted problem can be viewed as the node version of the *k-mst problem* with edge costs. In the k-mst problem we are given a graph with weights on its nodes and a specified node s. The goal is to find a maximum weight tree spanning k nodes rooted at s. In a generalization of this problem edges are associated with costs and instead of the parameter

k we are given a budget B. The goal is to find a maximum weight tree rooted at s the cost of which is bounded by B. The k-mst problem and its generalization are discussed in [**?**].

As mentioned earlier, the budgeted problem for bipartite networks is identical to the budgeted maximum coverage problem. In this problem we are given a collection $\mathcal{S}$ of sets with associated costs defined over a domain of weighted elements, and a budget B. The goal is to find a subset $\mathcal{S}' \subseteq \mathcal{S}$ such that the total cost of sets in $\mathcal{S}'$ does not exceed B, and the total weight of the elements covered by $\mathcal{S}'$ is maximized.

Observe that any instance of the budgeted problem for bipartite networks can be translated to an instance of the budgeted maximum coverage problem, as follows. Each relay vertex $r \in F$ corresponds to a set in $\mathcal{S}$ consisting of the elements that correspond to the customer vertices connected to r in G. The budget in both problems is the same. The translation from an instance of the budgeted maximum coverage problem to an instance of the budgeted problem for bipartite networks is analogous.

For our algorithms we need to solve instances of the node weighted Steiner tree problem. In this problem we are given an undirected graph with costs on the edges and on the vertices and a set of terminal vertices. The goal is to connect the terminals using a minimum cost Steiner tree, where the cost is accumulated on both vertices and edges of the tree.

3 General networks

3.1 The budgeted problem

Under our assumption that there is no distinction between relay vertices and customer vertices, the budgeted problem can be formulated as follows. Given a graph $G(V, E)$, and a specified source s, find a connected subgraph $T(V', E')$ of G that contains the source s, such that $\mathsf{cost}(V') \leq B$ and $\mathsf{w}(V')$ is maximized. Clearly, we may assume that T is a tree. We show a polynomial time algorithm that computes such a tree T whose weight is $\Omega(1/\log^2 n)$ times the weight of an optimal tree whose cost is bounded by B. We allow our algorithm to use a budget of $2B$. In this sense our result is a pseudo-approximation.

The cost function $\mathsf{cost}(\cdot)$ induces a distance metric on the graph, where the cost of a path (in the metric) is equal to the sum of the costs of the vertices belonging to it. Since the available budget is B, no vertex whose distance from s is more than B, can be connected to s in the optimal solution. Thus, we can preprocess the graph and omit all vertices whose distance from s is more than B.

The algorithm consists of three steps.

1. Compute an upper bound P on the weight using a linear program.
2. Find a "good ratio" tree $T_r(U, F)$; i.e., a tree whose weight is a least $P/4$ and the ratio of its weight to cost is $\Omega(P/(B \log^2 n))$.

3. Find a subtree of T_r which has cost between $B/2$ and $2B$ that maintains the same ratio of weight to cost. Clearly, the weight of this tree is $\Omega(1/\log^2 n)$ times the optimal weight.

Computing an upper bound

We formulate a multi-commodity flow problem whose optimal solution is an upper bound on the weight. The LP relaxation is as follows.

$$
\begin{array}{llll}
\max & \sum_i w_i x_i & & \\
& \sum_w f^u_{wv} \leq x_v & & \text{for } v \in V \\
& \sum_w f^u_{wv} = \sum_w f^u_{vw} & & \text{for } v \in V,\ v \notin \{s, u\} \\
& \sum_v x_v c_v \leq B & & \\
& 0 \leq x_v \leq 1 & &
\end{array}
$$

In the flow problem we have one commodity per each vertex and the goal is to maximize the weighted flow from the source s. The budget constraint is maintained using the capacities. More formally, transform each edge into two anti-parallel directed arcs. Let x_v denote the capacity of every vertex. Let f^u_{vw} denote the flow of commodity u on the directed arc (v, w). The net outgoing flow of commodity u from the vertex v has to be less than x_v. The source ships x_v units of commodity v to every vertex v.

Observe that any integral solution of the linear program corresponds to a possible outage recovery within budget B. Thus, the value of an optimal solution to the LP relaxation of the problem yields an upper bound on the weight of an optimal (integral) solution.

Extracting a good ratio tree

Denote the LP optimum weight by P. Consider the set of vertices receiving non-zero flow from the source s in the optimal solution. First, we can ignore the set of vertices that receive flow less than $\frac{1}{n^2}$. This might reduce the entire weight by at most a factor of $(1 - \frac{1}{n})$. This is because the weight of each of these vertices cannot exceed P, otherwise choosing such a single vertex v with the cheapest path from s to v would have been a feasible solution (since the cost of the path does not exceed B) of weight more than P, violating optimality.

Note that we are only disregarding the weight accrued by these vertices. We are not disregarding the flow through these vertices or the cost associated with them.

Partition the rest of the vertices receiving positive flow into disjoint sets: the vertices in the set V_i receive flow from the source s in the interval $(2^{-i}, 2^{-i+1}]$. Clearly, there are at most $k = O(\log n)$ sets, and the total weight accrued by the vertices in these sets is at least $P(1 - \frac{1}{n})$.

We distinguish between two cases.

Case 1: The total weight accrued by the vertices in $V_1 \cup \cdots \cup V_{\log\log n}$ is at least $P/4$. In this case, we find a node weighted Steiner tree for the set of terminals $V_1 \cup \cdots \cup V_{\log\log n}$, where the cost of each vertex v is $\mathsf{cost}(v)$ and the cost of an edge e is zero if $e \in E$ and infinity otherwise. Consider the fractional relaxation of the node weighted Steiner tree problem. In this relaxation we need to assign capacities to vertices and edges such that the total capacity of any cut that separates the terminals from the source is at least one. (The cut may involve edges and vertices.) The goal is to minimize the total cost which is the sum of edge and vertex capacities multiplied by their cost.

The solution for our original LP induces a feasible solution to this relaxation with cost at most $B \log n$, since we need to multiply each flow in our original solution by at most $\log n$ to get a solution to the relaxation.

In the next subsection we show how to find a node weighted Steiner tree whose cost is $O(\log n)$ times the cost of an optimal fractional solution. We run this procedure on our instance, and we claim that this tree is a "good ratio" tree since it connects vertices whose total weight is at least $P/4$, and its cost is $O(B \log^2 n)$.

Case 2: The total weight accrued by the vertices in $V_1 \cup \cdots \cup V_{\log\log n}$ is less than $P/4$. In this case there exists a set V_i, where $i > \log\log n$, that accrues weight at least $P/(2\log n)$. Note that the total weight of the vertices in V_i is at least $2^{i-1} \cdot P/(2\log n) \geq P/4$.

We find a node weighted Steiner tree on the set of terminals V_i. Consider again the fractional relaxation of the node weighted Steiner tree problem. The solution of our original LP induces a feasible solution to this relaxation with cost at most $B \cdot 2^i$, since we need to multiply each flow in our original solution by at most 2^i to get a solution to the relaxation.

In the next subsection we show how to find a node weighted Steiner tree whose cost is $O(\log n)$ times the cost of the optimal relaxed solution. We claim that this tree is a "good ratio" tree since it connects vertices whose total weight is at least $2^{i-2} \cdot P/(\log n)$ and its cost is $O(B \cdot 2^i \cdot \log n)$.

Extracting a good ratio tree with bounded budget

In the previous step we computed a tree whose weight is at least $\frac{P}{4}$ and its ratio of weight to cost is $\gamma = O(\frac{P}{B\log^2 n})$. We now show how to extract a tree of cost at most $2B$ and weight $\Omega(\gamma B)$ from this tree.

Orient all the edges in the tree towards s. Pick an arbitrary subtree in the tree rooted at any vertex. If, after deleting this subtree: (1) the cost of the remaining tree is at least

$B/2$; and (2) the ratio of the remaining tree is at least γ, then delete this subtree. We stop when no such subtrees exist anymore. Let the resulting tree be T. It has weight at least $\gamma B/2$. If the cost of T is at most $2B$, then we are done. Suppose that the cost of T is more than $2B$.

We now look for a subtree T' of T with ratio less than γ such that no subtree of T' (if such exists) has ratio less than γ. It may be the case that such a subtree T' does not exist. We consider this possibility in Case 1. Otherwise, we distinguish between Cases 2 and 3.

Case 1: There is no such subtree T' of T. In this case it must be that all subtrees in the tree T have a ratio at least γ. In this case find a subtree T'' such that its cost is more than $B/2$, yet the cost of subtrees rooted at the children of the root of T'' is not. Such a vertex must exist since the weight of T is more than $2B$. We have two possibilities depending on the total cost of the children of the root of T''. (a) If the total cost of the children is more than B, then pick subtrees rooted at the children of the root of T'' arbitrarily, till a set of subtrees having cost between $B/2$ and B is accumulated. Since each subtree has ratio at least γ the total weight is at least $\gamma B/4$. To obtain the desired tree connect these subtrees to s, incurring an additional cost of at most B. (b) If the total cost of the children is less than B (or if the root of T'' has no children), then the cost of the tree T'' must be at most $2B$. Since it has ratio at least γ, its weight is at least $\gamma B/2$.

Case 2: If there exists a leaf of T' with ratio less than γ it must be the case that the rest of the tree has weight less than $B/2$. Otherwise, we would have deleted the leaf. Since the leaf has cost at most B, the total cost of T is clearly less than $2B$ and weight more than $\gamma B/2$.

Case 3: The only remaining case is that there exists an internal vertex of v of T' such that the subtree rooted at v has ratio less than γ and all the subtrees rooted at the children of v have ratio at least γ. Since all these subtrees have good ratio, if any one of them has cost between $B/2$ and B we can connect it to s directly at an additional cost of B to get the desired tree. If any such subtree of T' has cost more than B, we can use arguments similar to Case 1 to get the desired tree. The only remaining case is when all the subtrees of T' have cost less than $B/2$. In this case we can pick a set of them of cost between $B/2$ and B, once again connect them to s paying an additional B. Notice that we can always pick such a subset since the total cost of the subtrees of T' is at least $B/2$. Otherwise we would get that the cost of T is less than $2B$ because the cost of all but T' is at most $B/2$ (or otherwise we would have removed the tree T'), and the cost of T' is at most $B + B/2$.

3.2 The node weighted Steiner tree

In this subsection we show how to find a node weighted Steiner tree whose cost is $O(\log n)$ times the cost of an optimal solution of the fractional relaxation of this problem.

Klein and Ravi [KR95] gave an $O(\log n)$ approximation algorithm for this problem. However, their approximation is with respect to the optimal *integral* solution. We present a tighter analysis of the KR algorithm proving the approximation factor relative to the *fractional* optimal solution as well. As a matter of fact our analysis also induces an alternative formulation of the KR algorithm as a "sphere growing" algorithm in the sense of the primal-dual method for approximation algorithms [Ho97].

As in [KR95], we assume without loss of generality that the cost of the terminals is zero. The KR algorithm works in iterations, and stops when all the terminals are connected. Iteration i starts with a collection of ϕ_{i-1} connected components where each contains at least one terminal. These connected components were computed in previous iterations. Initially each connected component consists of a single terminal.

Define a *spider* as a graph with at most one vertex of degree more than two. The ith iteration of the KR algorithm is as follows. It finds, among all spiders that connect at least two components, a spider with the minimum ratio of cost to number of components it connects. This spider is used to decrease the number of components. To prove the logarithmic factor on the approximation factor, Klein and Ravi prove that this ratio can be no more than the value of an optimal solution divided by ϕ_{i-1}. We prove that the same claim but relative to a *fractional* optimal solution.

Theorem 1 *The ratio of the spider found in iteration i of the KR algorithm is no more than the value of the optimal fractional solution divided by ϕ_{i-1}.*

Proof: To prove the theorem we show a dual feasible solution such that this ratio is no more than the value of the dual feasible solution divided by ϕ_{i-1}.

The dual problem is a packing problem in which we need to pack all the cuts that separate the terminals. Since, in our case, the cost of all edges in E is zero and the cost of the rest of the edges is infinite, we may consider only vertex cuts. The packing constraint is that the total sum of the dual variables assigned to all cuts containing a vertex v must be at most $\mathsf{cost}(v)$.

Suppose that H is the best ratio spider found in iteration i and let h_i be the number of components connected by H. Define the *radius* of H, denoted r_H, to be $\mathsf{cost}(H)/h_i$. To define a dual feasible solution we identify all the cuts for which we are going to assign a non-zero dual variable, and then show that this assignment is feasible.

Consider a connected component C at the start of iteration i. For each vertex v outside C we define $\mathsf{dist}(C, v)$ to be the cost of the cheapest path that connects a vertex from C to v. (The cost of the cheapest path to v does not include the cost of its two endpoints.)

Let $v_1, v_2, \ldots$ be the vertices not in C ordered according to $\mathsf{dist}(C, v_\ell) + \mathsf{cost}(v_\ell)$. Let

v_k be the vertex with the maximum index such that $\mathsf{dist}(C, v_k) + \mathsf{cost}(v_k) < r_H$. We define $k+1$ cuts that separate the component C from the rest of the graph. These are the cuts "around" C. For $1 \le j \le k$ the cut, $\mathsf{cut}(C, j)$, is the cut that consists of all the vertices v_ℓ for which $\mathsf{dist}(C, v_\ell) < \mathsf{dist}(C, v_j) + \mathsf{cost}(v_j) \le \mathsf{dist}(C, v_\ell) + \mathsf{cost}(v_\ell)$. The value of the dual variable assigned to this cut is $\mathsf{dist}(C, v_j) + \mathsf{cost}(v_j) - (\mathsf{dist}(C, v_{j-1}) + \mathsf{cost}(v_{j-1}))$, where $\mathsf{dist}(C, v_0)$ and $\mathsf{cost}(C, v_0)$ are defined to be zero. The cut, $\mathsf{cut}(C, k+1)$, is the cut that consists of all the vertices v_ℓ for which $\mathsf{dist}(C, v_\ell) < r_H \le \mathsf{dist}(C, v_\ell) + \mathsf{cost}(v_\ell)$. The value of the dual variable assigned to this cut is $r_H - (\mathsf{dist}(C, v_k) + \mathsf{cost}(v_k))$. Observe that the total sum of the dual variables assigned to all the cuts around C is r_H.

We need to show that the dual variables assigned to the cuts indeed constitute a dual feasible solution. Consider a component C and a vertex v_ℓ such that $\mathsf{dist}(C, v_\ell) + \mathsf{cost}(v_\ell) < r_H$. We first claim that this vertex is not part of any cut around any other cut C'. To prove this assume that this is not the case. Then, it must be $\mathsf{dist}(C', v_\ell) \le r_H$. But then the spider that connects C and C' has a cost strictly less than $2r_H$, contradicting the assumption that H has the best ratio. Let ℓ' be the maximum index for which $\mathsf{dist}(C, v_{\ell'}) + \mathsf{cost}(v_{\ell'}) = \mathsf{dist}(C, v_\ell)$. By the definition of $\mathsf{dist}(C, v_\ell)$ such an index must exist. Note that the cuts around C that contain v_ℓ are the cuts $\mathsf{cut}(C, j)$, for $\ell' < k \le \ell$. It is easy to see the total sum of the dual variables assigned to these cuts is exactly $\mathsf{cost}(v_\ell)$.

All that remains to consider is a vertex v_ℓ such that $\mathsf{dist}(C, v_\ell) < r_H \le \mathsf{dist}(C, v_\ell) + \mathsf{cost}(v_\ell)$. The total sum of the dual variables assigned to all the cuts around C that contain this vertex is $r_H - \mathsf{dist}(C, v_\ell) \le \mathsf{cost}(v_\ell)$. This vertex may belong to cuts around other components C' as well. This may happen only if $\mathsf{dist}(C', v_\ell) < r_H < \mathsf{dist}(C', v_\ell) + \mathsf{cost}(v_\ell)$. To obtain a contradiction assume that v_ℓ is around all components $C \in \mathcal{C}$ and that $\sum_{C \in \mathcal{C}} (r_H - \mathsf{dist}(C, v_\ell)) > \mathsf{cost}(v_\ell)$. Then, there must exist a spider connecting all the components in $\mathcal{C}$ whose cost is $\sum_{C \in \mathcal{C}} \mathsf{dist}(C, v_\ell) + \mathsf{cost}(v_\ell) < |\mathcal{C}| r_H$, contradicting the assumption that H has the best ratio.

Since the total value of the cuts around each connected component is exactly r_H and since there are ϕ_{i-1} such components, we conclude that there is a feasible dual solution with total value $\phi_{i-1} \cdot r_H$. The theorem follows since the cost of the spider is $h_i \cdot r_H$. □

3.3 The minimum weighted latency problem

We describe how to approximate the minimum weighted latency. First, we give a simple $O(\log^2 n)$ approximation algorithm. Similar algorithms achieve logarithmic factor approximation for tree and bipartite networks.

Suppose that we know the optimal weighted latency, denoted by L^*. In reality we will try $O(\log n)$ values increasing by a factor of $1 + \epsilon$ and consider the best schedule. This will introduce an additional error factor not exceeding $1 + \epsilon$. The algorithm progresses in iterations. In each iteration it connects a set S_i of nodes having total weight $W_i = \mathsf{w}(S_i)$. Let $W = \mathsf{w}(V)$ and $W_0 = 0$, and define $t_{i+1} = \frac{L^*}{W - \sum_{j=1}^{i} W_j}$.

In iteration i we consider the graph given by coalescing all the vertices in $\{s\} \cup_{j=1}^{i-1} S_j$ to a single vertex, and solve the linear program relaxation for the budgeted problem described above for this graph with budget $2Bt_i$. The set S_i is the set of all vertices v that receive a flow of at least $1/4$; i.e. all vertices v for which $x_v \geq 1/4$. We connect the vertices of S_i to the (coalesced) source s by solving a node weighted Steiner tree. Note that the cost of this tree is $O(\log n)$ times the optimal value of the linear program. Since the total cost of the vertices in S_i is $cBt_i \log n$, for some constant c, these vertices can be connected by identifying at most $2ct_i \log n$ subsets of S_i, each of budget at most B.

We continue in this manner until the total weight of the vertices that remain disconnected is less than L^*/n^2. We connect the remaining vertices one by one incurring an additional weighted latency of at most L^*/n since at each time unit we connect at least one vertex.

We now bound the contribution to the bounded latency from the iterations described above.

Lemma 2 $W_{i+1} \geq (1/3)(W - \sum_{j=1}^{i} W_j)$.

Proof: The weight $W - \sum_{j=1}^{i} W_j$ is the weight of the vertices that remain disconnected after iteration i. The optimal weighted latency of these nodes is at most L^*, thus by Markov's inequality the set of nodes which complete by time $2\frac{L^*}{W - \sum_{j=1}^{i} W_j} = 2t_{i+1}$ is at least $(1/2)(W - \sum_{j=1}^{i} W_j)$.

Thus the LP will have a solution which gives at least that much weight. Consider W_{i+1} which is the total weight of the vertices for which $x_v \geq 1/4$. Clearly, the maximum weight that the LP can achieve is $W_{i+1} + 1/4(W - \sum_{j=1}^{i} W_j - W_{i+1})$. Since this must be at least $1/2(W - \sum_{j=1}^{i} W_j)$, we get $W_{i+1} \geq 1/3(W - \sum_{j=1}^{i} W_j)$. □

This Lemma implies that number of iterations is $O(\log n)$. Now we claim the following theorem.

Theorem 3 *The weighted latency is* $O(L^* \log^2 n)$.

Proof: The reconnection time incurred in iteration $i+1$ is $O(t_{i+1} \log n)$. Thus, the contribution of this time to the weighted latency is no more than $O\left((W - \sum_{j=1}^{i} W_j) \cdot t_{i+1} \log n\right)$. This is because this time is added to the latency of all vertices that complete after iteration i.

So the total contribution is bounded, up to a constant factor, by

$$\begin{aligned} \sum_i (W - \sum_{j=1}^{i} W_j) t_{i+1} \log n &= \sum_i (W - \sum_{j=1}^{i} W_j) \frac{L^*}{W - \sum_{j=1}^{i} W_j} \log n \\ &= \sum_i L^* \log n \;=\; L^* \log^2 n \end{aligned}$$

□

We achieved the $O(\log^2 n)$ approximation factor in the above algorithm by lower bounding the total weight of the nodes connected in each iteration. The lower bound is computed from the estimate of the optimal latency L^*. However, this bound may not be so accurate. Ituitively, the reason for this is the fact that we cannot distinguish between two "extreme" cases that may result in the same weighted latency. In one case the number of iterations is small and the weight of the nodes connected in each iteration is roughly the same. In the other case the number of iterations is large but most of the weight is connceted in the first iteration while in the rest of the iterations only a small amount of weight is connected. To get a better approximation factor we try to use two bounds simultaneously: a lower bound on the total weight of the nodes connected in each iterations, as before; and a lower bound on the decrease in the residual weighted latency after each iteration. We show that using both bounds we can improve the approximation factor to $O(\log\log n \log n)$.

An $O(\log n \log\log n)$ algorithm:

To refine our bounds we partition each iteration into at most m sub-iterations. (The parameter m will be computed later.) The output of the iteration is again a set of vertices S_i that are to be connected. Before we describe the algorithm, let us set some notation.

Define Φ_{i+1} as an upper bound on the contribution to the optimal weighted latency of all nodes remaining after i iterations. Set $\Phi_1 = L^*$. The algorithm described below will proceed, either until the total weight of all the remaining nodes to be connected is less than L^*/n^2, or until the end of iteration ℓ, for the first ℓ for which $\Phi_{\ell+1}$ is less than L^*/n^2. In both cases it is easy to see the latency incurred by connecting all the remaining vertices (one by one) does not exceed L^*/n.

Define Γ_i^0 to be the set of all vertices that remain disconnected after $i-1$ iterations. Each iteration consists of m sub-iterations. We describe sub-iteration j of iteration i.

The input to this sub-iteration is the set of vertices Γ_i^{j-1}. In this sub-iteration we first solve the (fractional) budgeted problem relative to the vertices in Γ_i^{j-1} with budget

$$\frac{B\alpha\Phi_i}{\mathsf{w}(\Gamma_i^{j-1})2^{j-1}},$$

where $\alpha \geq 4$ is a parameter to be computed later. To solve the budgeted problem we solve the LP given in Section 3.1 on an instance of the original graph in which the weights of the vertices not in Γ_i^{j-1} are set to zero and the weights of the vertices in Γ_i^{j-1} are their original weights in the graph.

Define Γ_i^j to be the set of nodes receiving a flow of $\frac{1}{2}$ or more in the (fractional) solution of the LP.

If $\mathsf{w}(\Gamma_i^j) \leq (\frac{\alpha-2}{\alpha})\mathsf{w}(\Gamma_i^{j-1})$, iteration i is terminated. Set $S_i = \Gamma_i^{j-1}$ and solve a node weighted Steiner tree to connect the nodes in set S_i as in the previous algorithm. In addition, set $\Phi_{i+1} = \Phi_i(1-2^{-j+1})$, and $\Gamma_{i+1}^0 = \Gamma_i^0 - S_i$.

Otherwise (i.e., if $\mathsf{w}(\Gamma_i^j) > (\frac{\alpha-2}{\alpha})\mathsf{w}(\Gamma_i^{j-1})$) we distinguish between two sub-cases. if

$j = m$ then again iteration i is terminated. Set $S_i = \Gamma_i^{m-1}$ and solve a node weighted Steiner tree to connect the nodes in set S_i as in the previous algorithm. In addition, set $\Phi_{i+1} = \Phi_i$, and $\Gamma_{i+1}^0 = \Gamma_i^0 - S_i$.

The only remaining subcase is $j < m$ and $\mathsf{w}(\Gamma_i^j) > (\frac{\alpha-2}{\alpha})\mathsf{w}(\Gamma_i^{j-1})$. In this case we proceed to the next sub-iteration $(j+1)$ of iteration i, and start it by solving the budgeted problem relative to the vertices in Γ_i^j with budget $B\alpha\Phi_i/(\mathsf{w}(\Gamma_i^j)2^j)$,

To prove the correctness of the algorithm, we first prove inductively that Φ_i is an upper bound on the weighted latency of the nodes in Γ_i^0. Since $\Phi_1 = L^*$, this is true for the first iteration. We assume that it is true for iteration i. Note that $\Phi_{i+1} \neq \Phi_i$ only when there is a $1 < j \leq m$ for which $\mathsf{w}(\Gamma_i^j) \leq \frac{\alpha-2}{\alpha}\mathsf{w}(\Gamma_i^{j-1})$.

Lemma 4 *If $\mathsf{w}(\Gamma_i^j) \leq \frac{\alpha-2}{\alpha}\mathsf{w}(\Gamma_i^{j-1})$, then the contribution of Γ_i^{j-1} to the weighted latency is at least $\Phi_i 2^{-j+1}$.*

Proof: The proof is by contradiction. To obtain a contradiction, suppose that the contribution of Γ_i^{j-1} is less than $\Phi_i 2^{-j+1}$. Then, at least a weight of $(1-\frac{1}{\alpha})\mathsf{w}(\Gamma_i^{j-1})$ of the nodes in Γ_i^{j-1} is completed within time

$$t = \frac{\alpha\Phi_i}{\mathsf{w}(\Gamma_i^{j-1})2^{j-1}}.$$

Thus, there exists a solution to the LP with budget Bt (which is the budget used in sub-iteration j) that achieves a weight larger than $(1-\frac{1}{\alpha})\mathsf{w}(\Gamma_i^{j-1})$. Now, the nodes in Γ_i^j received a flow of more than $\frac{1}{2}$ in the LP. Clearly, the maximum possible weight obtained by the LP is at most

$$\mathsf{w}(\Gamma_i^j) + \frac{1}{2}(\mathsf{w}(\Gamma_i^{j-1}) - \mathsf{w}(\Gamma_i^j)).$$

Thus, we must have

$$\mathsf{w}(\Gamma_i^j) + \frac{1}{2}(\mathsf{w}(\Gamma_i^{j-1}) - \mathsf{w}(\Gamma_i^j) > \left(1 - \frac{1}{\alpha}\right)\mathsf{w}(\Gamma_i^{j-1})$$

Rearranging the expression we get $\mathsf{w}(\Gamma_i^j) > \frac{\alpha-2}{\alpha}\mathsf{w}(\Gamma_i^{j-1})$, a contradiction. □

Observe that the fact that Φ_i is an upper bound on the contribution of the nodes in Γ_i^0 to the weighted latency implies that we never terminate after the first sub-iteration. To see this, note that since $\alpha \geq 4$ we terminate after the first sub-iteration of iteration i only if $\mathsf{w}(\Gamma_i^1) \leq \frac{1}{2}\mathsf{w}(\Gamma_i^0)$. However, this implies that the value of the LP is bounded by $\frac{3}{4}\mathsf{w}(\Gamma_i^0)$, which in turn implies that the contribution of the nodes in Γ_i^0 to the weighted latency is at least $\frac{3}{4}\Phi_i + \frac{1}{2}\Phi_i$.

Now we prove the following bound on the approximation factor.

Theorem 5 *The above algorithm is an $O(\log n \log\log n)$ approximation algorithm for the minimum weighted latency problem.*

Proof: For each $j > 1$, we upper bound the contributions to the total latency of iteration i in which Γ_i^j was chosen to be connected. Let this contribution be C_i. The first observation is that if in iteration i the algorithm chooses a certain Γ_i^j, for $j > 1$, to be connected then, $\mathsf{w}(\Gamma_i^j) \geq \left(\frac{\alpha-2}{\alpha}\right)^m \mathsf{w}(\Gamma_i^0)$.

For each such i, the nodes in Γ_i^j received flow of value at least $\frac{1}{2}$ under the budget

$$\frac{\alpha B \Phi_i}{\mathsf{w}(\Gamma_i^{j-1}) 2^{j-1}}$$

If we double the flow to every vertex and solve a node weighted Steiner tree instance on the nodes of Γ_i^j, the cost of the Steiner tree is at most $2c \log n$ times the above budget, where c is some constant (in our adaptation of [KR95], $c = 2$). The reconnection time of the nodes in Γ_j^i is $\frac{1}{B}$ times the cost of the tree. Thus, the contribution to the latency is $\mathsf{w}(\Gamma_i^0)/B$ times the cost of the tree. Since $\mathsf{w}(\Gamma_i^0) \leq (\frac{\alpha}{\alpha-2})^m \mathsf{w}(\Gamma_i^j)$, and $\mathsf{w}(\Gamma_i^j) \leq \mathsf{w}(\Gamma_i^{j-1})$, we can upper bound the contribution C_i to the latency by

$$C_i = 2c \frac{\alpha \Phi_i \log n}{2^{j-1}} \frac{\mathsf{w}(\Gamma_i^0)}{\mathsf{w}(\Gamma_i^{j-1})} \leq \frac{4c\alpha \Phi_i \log n}{2^j} \left(\frac{\alpha}{\alpha-2}\right)^m \tag{1}$$

Now, from the above equation, for those iterations i where Γ_m^i was not chosen, we can write

$$\Phi_i - \Phi_{i+1} = \frac{\Phi_i}{2^j} \geq \frac{C_i}{4c\alpha \left(\frac{\alpha}{\alpha-2}\right)^m \log n}$$

Notice that for the iterations that Γ_m^i *was indeed chosen*, the estimated upper bound Φ remained unchanged, that is $\Phi_i = \Phi_{i+1}$. Therefore, using the fact that $\sum_i (\Phi_i - \Phi_{i+1}) \leq \Phi_1$, over the iterations i such that Γ_i^m was not chosen,

$$\sum_i C_i \leq \Phi_1 \cdot 4c\alpha \left(\frac{\alpha}{\alpha-2}\right)^m \log n.$$

We now consider the iterations for which Γ_i^m was chosen. From equation (1), the total contribution to the latency from these iterations is

$$\sum_i C_i \leq \sum_i 4c\alpha \Phi_i \left(\frac{\alpha}{\alpha-2}\right)^m 2^{-m} \log n$$

By our observation, at least $(\frac{\alpha-2}{\alpha})^m \mathsf{w}(\Gamma_i^0)$ weight is connected in each iteration. Thus, there are $O((\frac{\alpha}{\alpha-2})^m \log n)$ iterations total. Since Φ_i in non-increasing, for these iterations i where Γ_i^m was chosen, $\sum_i \Phi_i$ can be bounded by $O(\Phi_1 (\frac{\alpha}{\alpha-2})^m \log n)$. Using this in the above equation, the contribution from these iterations is at most,

$$\sum_i C_i \leq O\left(\alpha \Phi_1 \left(\frac{\alpha}{\alpha-2}\right)^{2m} 2^{-m} \log^2 n\right)$$

Inspecting both the terms, since $\alpha/(\alpha-2) \geq 1$, the contribution to the total latency can be bounded by,

$$O\left(\alpha\Phi_1\left(\frac{\alpha}{\alpha-2}\right)^{2m}\max(\log n, 2^{-m}\log^2 n)\right)$$

Setting $\alpha = 2\log\log n + 2$, and $m = \log\log n$, the above is $O\left(L^* \log n \log\log n\right)$. This proves the theorem. □

4 Tree networks

We present a polynomial time algorithm that optimally solves the budgeted problem for polynomially bounded budgets in tree networks. For arbitrary budgets we present two strongly polynomial approximation schemes. Similar to the (first) algorithm for the minimum weighted latency problem in general networks, we can apply these algorithms to obtain a logarithmic approximation algorithm for the minimum weighted latency problem in tree networks. This latter problem can be shown to be strong NP-Hard.

We first consider polynomially bounded budgets. To obtain a polynomial time algorithm that solves the problem optimally we use a bottom up dynamic programming as follows.

Consider a vertex v with children $v_1, v_2, \ldots, v_m$. Assume that for each child v_i and every possible budget $B' \leq B$, we have already computed the maximum weight gained from the subtree T_i rooted at v_i with budget B', denoted $T[v_i, B']$. We show how to compute $T[v, B']$ for all $B' \leq B$. Note that if v is a leaf then $T[v, B']$ is v if its cost is bounded by B' and $\emptyset$ otherwise.

For $i = 1, \ldots, m$ and $B' \leq B$, let $A[i, B']$ denote the maximum weight when the budget B' is distributed over $T_1, \ldots, T_i$. Observe that if $B' = 0$ then $A[1, B']$ is given by the largest zero cost subtree rooted at v_1.

$$\begin{aligned}
A[1, B'] &= T[v_1, B'] \\
A[i+1, B'] &= \max_{0 \leq j \leq B'}\{A[i, B'-j] + T[v_{i+1}, j]\} \\
T(v, B') &= \mathsf{w}(v) + A[m, B' - \mathsf{cost}(v)]
\end{aligned}$$

The maximum weight for the tree T is $T[s, B]$. The correctness proof follows by a simple inductive argument on the depth plus the degree at the root. The running time of the algorithm is $O(B^2 n)$.

In case the budget is not polynomially bounded, we present two strongly polynomial time approximation schemes. The first scheme achieves the optimal weight by slightly over spending the budget B. The second scheme achieves slightly less than the optimal weight using budget B.

The first scheme is obtained by scaling the costs. Clearly, no vertex in the tree has cost more than the budget B. Round down the cost of every vertex to a multiple of B/n^2. Divide the rounded costs by B/n^2, and apply the above algorithm to compute the maximum weight that can be achieved with the rounded costs. The total costs which are unaccounted for due to the rounding is at most $\frac{B}{n}$. Thus with budget $B(1+\frac{1}{n})$ we get the optimal weight. To obtain the polynomial time approximation scheme that finds the optimal weight with budget $B(1+\epsilon)$, for any ϵ, we round down the cost of every vertex to a multiple of $B\epsilon/n$.

The second scheme maintains the budget B and approximates the weight by a factor of $1-\epsilon$ by scaling the weights. We apply dynamic programming based on the weights of the vertices. First, assume that the weights are polynomially bounded. Let $T_w[v,W]$ denote the minimum budget required to obtain a weight W from the subtree rooted at v. For a vertex v with children $v_1, v_2, \ldots, v_m$, let $A_w[i,W]$ denote the minimum budget required to obtain weight W from subtrees rooted at $v_1, v_2, \ldots, v_i$. We get

$$\begin{aligned}
A_w[1,W] &= T_w[v_1,W] \\
A_w[i+1,W] &= \min_{0\le j\le W}(A_w[i,W-j]+T_w[v_{i+1},j]) \\
T_w[v,W] &= \mathsf{cost}(v)+A_w[m,W-\mathsf{w}(v)]
\end{aligned}$$

The optimality of the above dynamic program can be proven by induction similar to the previous dynamic program. In case the weights are not polynomially bounded a strongly polynomial algorithm can be obtained by rounding up every weight to the nearest multiple of $P\epsilon/n$, where P is the optimal weight. The total weight of the final tree is at least $P(1-\epsilon)$, while maintaining the budget B. Note that P is not known. However, we can "guess" the weight P within $1-\epsilon$ factor, and check each guess by running the above algorithm. The use of the estimate for P rather than P increases the approximation factor to $(1-\epsilon)^2$.

5 Bipartite networks

We present a $(1-\frac{1}{e})$ approximation algorithm for the budgeted problem in bipartite networks. Again, similar to the (first) algorithm for the minimum weighted latency problem in general networks, we can apply this algorithm to obtain a logarithmic approximation algorithm for the minimum weighted latency in bipartite networks. The problem of reconnecting a maximum weight subset of the

customers with respect to a given budget B in bipartite networks is equivalent to the *budgeted maximum coverage problem*, defined as follows. A collection of sets $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ with associated costs $\{c_i\}_{i=1}^m$ is defined over a domain of elements $X = \{x_1, x_2, \ldots, x_n\}$ with associated weights $\{w_i\}_{i=1}^n$. The goal is to find a collection of sets

$\mathcal{S}' \subseteq \mathcal{S}$, such that the total cost of elements in $\mathcal{S}'$ does not exceed a given budget B, and the total weight of elements covered by $\mathcal{S}'$ is maximized.

A natural candidate for approximating this problem is the greedy heuristic that picks at each step a set maximizing the ratio of weight to cost. However, the greedy heuristic has an unbounded approximation factor. Consider, for example, two elements, x_1 of weight 1 and x_2 of weight p. Let $S_i = \{x_i\}$, $i = 1, 2$, let $c_1 = 1$, $c_2 = p + 1$, and let $B = p + 1$. The optimal solution contains the set S_2 and has weight p, while the solution picked by the greedy heuristic contains the set S_1 and has weight 1. The approximation factor for this instance is p, and is therefore unbounded.

We modify the greedy heuristic as follows. We first find a collection of sets according to the greedy heuristic. This collection is the first candidate for the final output. The second candidate is a single set S_t for which W_t is maximized. The modified algorithm outputs the candidate solution having the maximum weight. It can be shown that this algorithm achieves an approximation factor of $\frac{1}{2} \cdot (1 - \frac{1}{e})$ for the problem. (Due to space constraints details are omitted they can be found in [**?**].)

We improve the approximation factor by using enumeration. Let k be some fixed integer. We consider all subsets of $\mathcal{S}$ of cardinality k which have cost at most B, and we complete each such subset to a candidate solution using the greedy heuristic. Another set of candidate solutions consists of all possible k or fewer subsets of $\mathcal{S}$ that have cost at most B. The algorithm outputs the candidate solution having the greatest weight. Again, due to space constraints we do not give here the proof of the following theorem. (The proof can be found in [**?**].)

Theorem 6 *For $k \geq 3$, the above algorithm achieves an approximation factor of* $(1 - \frac{1}{e})$ *for the budgeted maximum coverage problem.*

6 Remarks and open problems

There are a lot of related open problems that may be considered. First, can the approximation factors be improved, or can it be shown (at least for some of the factors) that they are the best possible approximation factors.

A bigger challenge is to try to make the model more accurate. While we abstract the problem, we ignore many factors that affect the outage recovery. For example, we consider only one resource (workforce) and ignore the rest of the required resources. Although workforce is indeed the scarcest resource other resources such as parts have to be taken into account. Another factor that is ignored is travel time of the workforce. Here, we assume that each failure can be fixed within the given time unit and each repairman can work on any other failure in the next time unit, ignoring the cases where travel time from some failure locations to others may be too long.

References

[As] "ASCADA: Software Solutions for Utilities", more information on their offering can be found in http://www.ascada.com/external/products/tcs/tcs_prod.htm.

[BBL95] O. Berman, D. Bertsimas, and R. C. Larson, "Locating Discretionary Service Facilities, II: Maximizing Market Size, Minimizing Inconvenience", Operations Research, vol. 43(4), pp. 623-632, 1995

[BLF92] O. Berman, R. C. Larson, N. Fouska, "Optimal Location of Discretionary Service Facilities", Transportation Science, vol. 26(3), pp. 201-211, 1992

[Ho97] D. S. Hochbaum, "Approximation Algorithms for NP-Hard Problems, PWS Publishing Company, 1997.

[KR95] P. Klein and R. Ravi, "A nearly best-possible approximation algorithm for node-weighted Steiner trees", J. of Algorithms, (19):104–115, 1995.

[LT96] C. Letter and J. Tracey, "Next Generation trouble call management system heightens utility responsiveness", Synergy Newsletter, GE Harris Control Division, October 1996.

[MZH83] N. Megiddo, E. Zemel, and S. L. Hakimi. "The Maximum Coverage Location Problem", SIAM Journal on Algebraic and Discrete Methods, vol. 4(2), pp. 253-261