Locating Centers in a Dynamically Changing Network, and Related Problems

Dorit S. Hochbaum *
Department of Industrial Engineering and Operations Research and
Walter A. Haas School of Business,
University of California, Berkeley
email: dorit@hochbaum.berkeley.edu

Anu Pathria
HNC Software Inc.,
5930 Cornerstone Court West,
San Diego CA 92121-3728
email: akp@hnc.com

January 23, 1998

Abstract

In a dynamically changing network the costs or distances between locations are changing in each discrete time period. We consider the location of emergency facilities that must minimize the maximum distance to any customer on the network across *all* time periods. We call the problem of locating p centers over k underlying networks corresponding to k periods the k-*Network* p-*Center* problem. The problem is considered when in each period the network satisfies the triangle inequality. In this paper we provide a polynomial time 3-approximation algorithm for Δ k-*Network* p-*Center* for the case $k = 2$.

We discuss generalizations inspired by this problem to other optimization problems with multiple underlying networks and the objective of finding a single solution that varies as little as possible from the optimum for each network. The additional combinatorial problems discussed include: sorting; perfect matching; shortest path; minimum spanning tree; and minimum cut. All are shown to be NP-hard for $k \geq 2$.

Keywords: p-*Center*, bottleneck problems, approximation algorithms, location problems.

0 Introduction

The problem of locating facilities in a network becomes increasingly complex when conditions that affect the locational decisions are changing over time. We are interested here in changes in the distances between locations across different periods of time. The problem is to determine the

*Research supported in part by ONR contract N00014-91-J-1241, by NSF award No. DMI-9713482, and by SUN Microsystems.

location of permanent facilities that are permitted to serve different assignments of customers at different periods of time.

Our study of the problem of locating emergency facilities in dynamically changing networks was motivated by an application presented by Serra and Marianov [SM96] on locating firehouses in the city of Barcelona. The critical cost parameter - the transportation time - is affected by the time of day and the change in traffic patterns between off-peak hours and rush hours. In that particular application the locations for p emergency facilities were to be selected for the city of Barcelona that would optimally serve the population in the bottleneck sense: minimize the distance of the individual farthest from his closest facility at any time of the day. The day was divided into $k = 7$ periods and the center was to be chosen to serve "optimally" (Serra and Marianov considered two separate objective functions for combining the costs of a given center on the k networks) throughout the day. The "changing networks" concept has been studied in other specific instances (see [Das78, MO79, SR96, WC83]).

When locating emergency facilities it is common to use the model of the p-*Center* problem. In the p-*Center* problem the objective is to find the optimal location of p nodes so that the maximum distance of a node to its nearest center is minimized. This problem, which is known to be NP-hard (see [KH79]), turns up in the context of locating emergency facilities, say hospitals: we wish to locate no more than p hospitals such that the largest distance between each potential patient zone and the nearest hospital is minimized.

We model our problem similarly, except that instead of having a single underlying cost networks, there is one network for each period of time. We will require the location of *permanent* facilities so that the maximum distance over all k periods is minimized. We do allow for customers to be reallocated to the closest facility at each period of time. We call this problem the k-*Network* p-*Center* problem.

There are several modeling alternatives to locational decisions on dynamically changing network.

- *Mobile facilities*: Allowing the repositioning of facilities at each period can improve substantially the service and reduce costs. In such case it is necessary to take into account, in addition to the service costs, also the costs of moving the facilities. In practical situations it is important to consider the added value of allowing mobile facilities, and if justified, switch to this mode of operation.

- *Maintain fixed allocation*: It may be necessary to maintain the same assignment of customers to facilities throughout, rather than reassigning a customer to the closest facility at each period. If the allocation is fixed then there is some flexibility lost and the overall costs can only go up. Thus this is not a desirable requirement to have. Computationally, such problem is equivalent to p-*Center*. To see that we create a network where the cost/distance between two locations is the maximum over all k periods. In this new network, the problem is then solved as the p-*Center* problem.

- *On-line*: In certain situations the network costs and distances are changing dynamically, but there is no a-priori knowledge or prediction of the changes in costs. In these situations *on line* algorithms that devise a solution for one period at a time, after the costs become known, are called for. Such algorithms can however be used only if there is flexibility in changing either the facilities' locations or the customers allocations.

- *Directed network*: The problem we are modeling here suggests *directions* as an important concept. Traffic during rush hours is usually much heavier in one direction than in the opposite direction, and costs are not symmetric. Directed models have not been used much to date in the location literature, but their investigation is an important topic of research.

- *Continuous dynamic change*: The assumption here is that costs are changing in each discrete period. The cost function is thus a step function. In reality however, costs change continuously over time. If the step function approximation is not good enough, it may be appropriate to work directly with continuously changing costs.

We investigate here the differences between the p-*Center* problem and the k-*Network* p-*Center* problem with respect to approximation algorithms. Because the k-*Network* p-*Center* problem is a generalization of p-*Center* (k-*Network* p-*Center* is equivalent to p-*Center* when $k = 1$) it is NP-hard; it remains so under those restrictions for which p-*Center* remains NP-hard (ex. weights satisfying triangle inequality -Δ-inequality). For the Δ-inequality p-*Center* problem Hochbaum and Shmoys [HS85] devised a 'best possible' 2-approximation algorithm: The solution delivered by the algorithm is guaranteed to be at most twice the optimum, whereas any approximation factor smaller than 2 is proved to be an NP-hard problem. In the absence of Δ-inequality, *any* bounded approximation is NP-hard. The technique used in [HS85] has been generalized and shown applicable to numerous other bottleneck problems. Indeed, we explore here the use of the 'powers of graphs' technique [HS86], to the Δ k-*Network* p-*Center* for the case $k = 2$.

In the next section we formally define k-*Network* p-*Center* and consider the complexity of the problem; in particular, we develop a 3-approximation for $k = 2$ when the Δ-inequality holds using the "powers of graph" concept in conjunction with the *Bottleneck Algorithm* as introduced by Hochbaum and Shmoys [HS86]. We conclude with some final observations regarding this problem. We then discuss the concept of *"multiple underlying networks"* for other combinatorial problems and prove that this extension, of even simple polynomial problems, is NP-hard when $k = 2$.

1 The k-*Network* p-*Center* Problem

1.1 Definition and Complexity

A formal definition of k-*Network* p-*Center* is given below. We define the distance of a node, v, from a set of nodes, U, to be $\min_{x \in U}\{w_{\{x,v\}}\}$. Note that the distance of a node from itself is assumed to be 0: $w_{\{v,v\}} = 0$.

Instance: Given k sets of weights on a complete graph on the set of nodes V. Let $G_j = (V, E_j)$ represent the j'th graph, for $j = 1, 2, \ldots, k$, with edge e having weight $w_e^{(j)}$ in G_j. Let $\mathcal{G}$ refer to the set of k graphs.

k-*Network* p-*Center*: Select a set of p nodes, C, such that the maximum cost for any customer node in any network G_j is minimized. That is, the quantity $\max_{i \in V} \max_{j=1,\ldots k} \min_{s \in C} w_{is}^{(j)}$ is minimized.

In the Δ k-*Network* p-*Center* version of the problem, the edge weights in each of the k networks individually satisfy the triangle-inequality. The triangle-inequality assumption is not an unreasonable one as it often holds in practice, for example where edge *"costs"* may really mean to *shortest-distance* or *travel-time* between the endpoints of the edge.

Because the p-*Center* problem is a special case of k-*Network* p-*Center* ($k = 1$), certain complexity results for k-*Network* p-*Center* can be inferred from known results for p-*Center*.

NP-Hardness Results: p-*Center* is NP-hard, and therefore so is k-*Network* p-*Center*. p-*Center* remains NP-hard even when it assumed that there are only 2 distinct edge weights or when it is assumed that the edge weights satisfy the triangle inequality; this implies tht k-*Network* p-*Center* remains NP-hard under these restrictions.

Approximability: It is NP-hard to approximate p-*Center* (and hence k-*Network* p-*Center*) within any constant factor in polynomial time. It is NP-hard to approximate Δ p-*Center* (k-*Network* p-*Center* with $k = 1$) within a factor of better than 2 in polynomial time; however, there do exist polynomial time 2-approximation algorithms for p-*Center*.

We next develop a polynomial time 3-approximation algorithm for k-*Network* p-*Center* for the case $k = 2$.

1.2 Approximation Algorithm

The technique that we employ for constructing an approximation algorithm for Δ k-*Network* p-*Center* is to use the "powers of graphs" concept in conjunction with the *Bottleneck Algorithm*, as introduced by Hochbaum and Shmoys [HS86]. In [HS86], this approach was applied to numerous NP-complete bottleneck problems to provide approximation algorithms for the given problems. We slightly modify that approach to handle the "multiple underlying network" structure of the k-*Network* p-*Center* problem, and provide a 3-approximation algorithm for k-*Network* p-*Center*.

Powers of Graphs and the Bottleneck Algorithm: The p-*Center* Problem

In order to clarify the technique used we now sketch the 2-approximation algorithm for the Δ p-center problem. First we introduce the concept of powers of graphs:

Definition 1.1 *The c'th power of a graph G, for a positive integer c, is denoted as G^c. G^c is the graph with the same set of nodes as G, and with an edge between two nodes if and only if there is a path with at most c edges between those two nodes in G.*

Finding the c'th power of a graph G can be done simply in polynomial time. Using all pairs unweighted shortest paths computation, we add for any pair of nodes that are distance of $2, \ldots, c$ apart, an edge between the pair. The second power of a graph is also referred to as the *square* of the graph.

In bottleneck problems the value of the solution is determined by the weight of one of the edges in the graph. The procedure then consists of adding one smallest cost edge at a time, and examining the resulting graph for the required structure. For the p-center problem we examine each such graph for the existence of a dominating set of size $\leq p$. If such dominating set exists then there is a p-center solution the value of which does not exceed the largest edge cost in the graph. Identifying whether a given graph has a dominating set of size p, however, is an NP-complete problem. Consider instead the square of the graph. Now, if there is a dominating set of size $\leq p$ in the square of the graph, then taking this set as a feasible p-center solution will give an objective value at most twice the optimum. This follows since if there is a solution in the graph itself (and it is the first graph in which there is a dominating set of size $\leq p$), then there must be a dominating set of size $\leq p$ in the square of that graph. Because of the triangle inequality the largest edge in the square of the graph is at most twice the size of the largest edge in the graph, hence establishing the factor of 2. The only snag about this particular procedure, of course, is that finding a dominating set in a square of a graph is itself an NP-complete problem, just as difficult as finding a dominating set in the original graph.

We resolve this difficulty by finding instead a *maximal* independent set in the square of the graph. A set of nodes is independent if no two nodes in the set are adjacent; while the *maximum* independent set problem (where the objective is to find the largest independent set in the graph) is NP-complete (see [GJ79]), a maximal independent set (where the objective is to find an independent set of nodes such that it is impossible to add any additional node and have the resulting set remain independent) can be easily found in polynomial time by adding nodes to a set until every remaining node is adjacent to some node in the set. The maximal independent set has the same two required properties as the dominating set in the square of the graph: first, if the graph G has a dominating set of size p, then in G^2 every maximal independent set is also of size p or less; second, and moreover, a maximal independent set is also a dominating set and hence constitutes a feasible p-center using only the edges of G^2.

The worst case bound of value 2 now follows easily. At each phase of the algorithm one additional edge of smallest cost among the remaining edges is added to the graph. A maximal independent set is then found in the square of that graph. If it is greater than p, then there is no dominating set in the graph of size p or less and hence the value of the optimal p-center solution is greater than the cost of the edge added to the graph. The algorithm then continues to the next phase. When a maximal independent set in the square of the graph is $\leq p$ then we consider it a p-center feasible solution. Since it only uses in the assignment of nodes to centers the edges of the square of the graph, and the value of the optimum is at least the value of the largest edge in the graph, it follows that this feasible p-center uses edges of cost at most twice the optimum.

The Bottleneck Algorithm in General

The general approach relies on the property that the optimal solution value to bottleneck graph problems is often restricted to an easily computable set of values (for example, one of the edge weights, in our problem). The *Bottleneck Algorithm* is thus used in conjunction with a "test". Let a bottleneck graph be defined for edge weights ordered,

$$w_1 \leq w_2 \leq \cdots \leq w_m \ .$$

The bottleneck graph $G(w)$ is $(V, E(w))$, where $E(w) = \{e \in E | w_e \leq w\}$. That is, it contains only edges of weight less than the threshold value of w. The bottleneck procedure then inspects for increasing values of w the bottleneck graph for the possible existence of a certificate, that ensures that a feasible solution exists that uses only the restricted set of edges.

procedure **Bottleneck**

```
for i = 1..m do
    if Test(𝒢_i) = "no" then continue else output certificate; stop
end
```

For NP-hard bottleneck optimization problems, finding the optimal solution via **Bottleneck** involves solving a "test" that is typically an NP-complete decision problem (dominating set for the case of p-center problem). For k-*Network* p-*Center*, for example, let the possible optimal solution values be given by the graph edge weights,

$$w_1 \leq w_2 \leq \cdots \leq w_m \ .$$

Then, $\mathbf{Test}(\mathcal{G}_i)$ would be equivalent to finding a set of p nodes that dominates each of the graphs in $\mathcal{G}(w_i) = \{G_1(w_i), \ldots, G_k(w_i)\}$, where $G_j(w_i)$ is the unweighted graph with nodes V and those edges in E_j whose weight does not exceed w_i.

As was shown in [HS86], however, when confronted with bottleneck graph optimization problems in which the edge weights satisfy the Δ-inequality, solving a "test" problem (that is *not* NP-complete) on a related graph, namely the power of the original graph, can provide sufficient information for the derivation of an approximation algorithm running in polynomial time.

We note the following fact:

Fact 1.1 *Let G be a complete weighted graph whose edge weights satisfy the Δ-inequality, and let c be a positive integer. If H is a subgraph of G with maximum edge weight w, then H^c is a subgraph of G with maximum edge weight $\leq cw$.*

Definition 1.2 *Let $\mathcal{G} = \{G_1, \ldots, G_k\}$. Then $\mathcal{G}^c$ denotes the set $\{G_1^c, \ldots, G_k^c\}$.*

For the following discussion, we will assume that we have a minimization problem and that *Bottleneck* iterates through the possible optimal solution values in increasing order. Suppose we have a "test" such that, for a particular positive integer c, $\mathbf{Test}(\mathcal{G}_i)$ = "no" implies that the answer to the decision problem for $\mathcal{G}(w_i)$ is *no*, but that $\mathbf{Test}(\mathcal{G}_i)$ = "yes" implies that the answer to the decision problem for $\mathcal{G}^c(w_i)$ is *yes*. Then, let $i^* = i$ be the smallest index for which the answer to $\mathbf{Test}(\mathcal{G}_i)$ = "yes". As $\mathbf{Test}(\mathcal{G}_{i^*-1})$ = "no", we know that the optimal solution to the optimization problem is at least w_{i^*}; by Fact 1.1, however, $\mathbf{Test}(\mathcal{G}_{i^*})$ = "yes" returns a solution with value no greater than cw_{i^*}. Thus, given such a test, it is easy to see that **Bottleneck** is a c-approximation algorithm for the original problem. We note that, as the goal is to find the smallest index i for which **Test($\mathcal{G}_i$)** = "yes", it may be more efficient to implement **Bottleneck** using a binary search strategy rather than in the iterative manner presented.

A 3-Approximation Algorithm for k-*Network* p-*Center* for the Case $k = 2$

We now illustrate how to provide a suitable polynomial time "test", and thereby a polynomial time approximation algorithm, for the 2-*Network* p-*Center* problem.

Using **Bottleneck**, we could *optimally* solve k-*Network* p-*Center* if we could find a polynomial time algorithm to answer the following test:

> *Given a collection of unweighted graphs $\mathcal{G} = \{G_1, \ldots, G_k\}$ on the set of nodes V, does there exist a set of p nodes that is a dominating set for every graph in $\mathcal{G}$?*

In order to provide an answer to the test for $k = 2$ we construct in G_1^2 a *maximal* independent set I_1 and in G_2^2 a *maximal* independent set I_2. I_1 then dominates all the nodes in G_1^2, and I_2 dominates all nodes in G_2^2. Provided that both $I_1, I_2 \leq p$ there could exist p locations dominating all nodes in G_1^2 and G_2^2. The idea is now to look for a set of no more than p nodes D that will be adjacent to all nodes of I_1 in G_1 and all nodes of I_2 in G_2. Such a set of nodes is at most three edges away from each node of V in both G_1 and G_2, and is thus a 3-approximate solution.

An important observation is that the nodes of I_j are a distance of at least 3 edges apart, since they form an independent set in G_j^2. Thus each of the nodes in I_1 must be dominated by a different node of D in G_1 and, similarly, each of the nodes in I_2 must be dominated by a different node of D in G_2. We can determine in polynomial time the smallest set D by solving a *Set Cover* problem in which there are at most 2 elements per set. Each node in V is a potential 'set' that covers one or two nodes of I_1 and I_2. Even though *Set Cover* is in general NP-hard, instances with at most 2 elements per set can be solved in polynomial time using matching techniques, as this problem is essentially the edge cover problem (see [GJ79]). Construct an instance *SC* of *Set Cover* as follows:

- Let $I = I_1 \cup I_2$ be the set of elements to be covered.
- Associate a set S_v with each node $v \in V$.
  Place into S_v any: $i_1 \in I_1$ adjacent to v in G_1; $i_2 \in I_2$ adjacent to v in G_2.

Because I_1 is an independent set in G_1^2, at most one element from I_1 will be in any set; similarly for I_2. Thus, *SC* is an instance of *Set Cover* with at most 2 elements per set. Now, it is easy to see that covers for *SC* correspond exactly to sets of nodes D that cover I_1 in G_1 and I_2 in G_2. So, let D be a set of nodes corresponding to an optimal solution to *SC*:

- If $|D| > p$, then we know that there is no dominating set that covers the nodes I in $\mathcal{G}$. The answer to the test is "no".
- If $|D| \leq p$, then this is dominating set for $\mathcal{G}^3$. To see this, observe that in G_1 all nodes $v \in V$ are within distance 2 of some node in I_1 because I_1 is a maximal independent set

Input: A pair of graphs $\mathcal{G} = \{G_1, G_2\}$ of graphs on a set of nodes V and a number p.

1. Find I_1 and I_2, maximal independent sets in G_1^2 and G_2^2, respectively.
2. Form an instance SC of *Set Cover* as follows:
   - Let $I = I_1 \cup I_2$ be the elements to be covered.
   - Associate a set S_v with each node $v \in V$.
     Place into S_v any: $i_1 \in I_1$ adjacent to v in G_1; $i_2 \in I_2$ adjacent to v in G_2.
3. Find a minimal cover of SC.
   Let D be the set of vertices corresponding to this cover.
4. If $|D| > p$
   then output "no" /* **Test**($\mathcal{G}$) = "no"; no dominating set for $\mathcal{G}$ */
   else output D /* **Test**($\mathcal{G}$) = "yes"; dominating set exists for $\mathcal{G}^3$ */

Figure 1: **Test**($\mathcal{G}$) used by **Bottleneck** for k-*Network* p-*Center* for $k = 2$.

for G_1^2; because all nodes in I_1 are within distance 1 of D, it follows that all nodes are within distance 3 of D in G_1. Similarly, for G_2. Thus, D is a dominating set for $\mathcal{G}^3$, and the answer to the test is "yes".

Given that *Set Cover* with at most 2 elements per set can be solved in polynomial time, it is easy to see that all of the given operations can be carried out in polynomial time.

We have thus proved the following:

Theorem 1.1 *There exists a polynomial time* 3*-approximation algorithm for the* Δ*-inequality version of k-Network p-Center when* $k = 2$.

A description of the "test" as discussed previously, to be used within the **Bottleneck** procedure, is given in Figure 1.

2 Remarks on Approximation Algorithms for Δ k-*Network* p-*Center*

In this paper we have considered the k-*Network* p-*Center* problem, a generalization of p-*Center* in which a set of centers are to be selected to optimize an objective function involving a series of underlying networks. Because k-*Network* p-*Center* is a generalization of p-*Center*, which is known to be NP-hard even when the triangle inequality holds, we have concentrated on developing polynomial time approximation algorithms for Δ k-*Network* p-*Center* (finding polynomial time approximation algorithms for the general problem is also NP-hard).

A summary regarding the approximability of Δ k-*Network* p-*Center*:

$k = 1$: Several 2-approximation algorithms are known for Δ p-*Center*. Such algorithms are "best possible" in that it is known that finding a polynomial time algorithm with a better constant factor guarantee is NP-hard.

$k = 2$: We have provided in this paper a 3-approximation algorithm. It is not known whether this is "best possible".

$k \geq 3$: As we comment below, [BGKS96] have proved that approximating the problem for more than 2 periods is NP-hard.

As an alternative to developing approximation algorithms of the type discussed (in which p centers are required and we try to find a solution within a certain factor of the optimal), it may be satisfactory to provide solutions that have *more than* p centers. Along these lines, it can be observed that the impediment to extending the 3-approximation algorithm to other $k > 2$ is that we encounter an instance of *Set Cover* with up to k elements per set. While this can be optimally solved for $k = 2$, it is NP-hard for $k \geq 3$ (see [GJ79]). However, one can find a cover within a factor of

$$H(k) - \frac{1}{6}$$

of the optimal, where $H(k)$ is the k'th Harmonic number, [OGY93]. ($H(d) = \sum_{i=1}^{d} \frac{1}{i}$.) This implies that we can find a solution to k-*Network* p-*Center* in polynomial time that has cost within a factor of 3 of the optimal but that also uses up to $(H(k) - \frac{1}{6}) \times p$ centers. For $k = 3$ for instance, we get a 3-approximation using $\frac{5}{3}p$ centers.

In cases where the dynamically changing network is changing in some proportional manner, say with all link costs increasing by a factor in a certain range in each period, the problem may be much easier to solve. If the factor is the same for all edges then the problem is simply the p-*Center* problem.

It recently came to our attention that Bhatia, Guha, Khuller and Sussmann [BGKS96] worked independently on the k-*Network* p-*Center* problem and came up with the same approximation result for $k = 2$. Referring to the factor mentioned above of the largest ratio between largest and smallest edge costs across all networks as β, they proved the impossibility of any approximation better than β for k-*Network* p-*Center* for $k \geq 3$.

3 The Concept of "Multiple Underlying Networks"

The concept of "multiple underlying networks", with the objective being to optimize a function that combines the cost of the solution for each underlying network, may be of general interest.

Other Combinatorial Problems: Just as k-*Network* p-*Center* generalizes p-*Center* from a *single* to a *multiple* underlying network problem, so to it may be interesting to consider multiple underlying network versions of other problems. Consider for instance k-*Network Shortest Path*, the *Shortest Path* problem on k networks. Here we have to choose a path and use it in all k periods so the maximum length of that path is minimum. Two natural approaches that come to mind for solving the problem do not work: one approach is to find the shortest path in each network and take the best of these; a second approach is to find in $G_{\max}$ the shortest path. In fact, k-*Network Shortest Path* is NP-hard even for $k = 2$. In the appendix, we establish that multiple underlying network versions of *Shortest Path*, *Minimum Spanning Tree*, *Minimum Weight Perfect Matching*, and *Minimum*

Cut are all NP-hard; such a version of *Sorting*, however, is not, and a polynomial algorithm is provided. It appears that the concept of multiple underlying networks increases substantially problem complexity.

One observation, which may be useful for k-*Network* p-*Center* or for other problems with multiple underlying networks in which the objective is to optimize the underlying network that is least optimally served, is that if the graphs in $\mathcal{G}$ satisfy the triangle inequality then so does the graph in which each edge has weight equal to the maximum of the weights from the k original graphs. Thus, if the optimal solution to $G_{\max}$ can be related to the optimal solution for $\mathcal{G}$, results (complexity or approximation algorithms) achieved for $G_{\max}$ may imply results for $\mathcal{G}$.

Lemma 3.1 *Consider $\mathcal{G}$ as in the definition of k-Network p-Center. Let $G_{\max}$ be the complete graph on node set V with edge e having weight $w_e = \max\{w_{e_1}, \ldots, w_{e_k}\}$. Then, the edge weights in $G_{\max}$ satisfy the triangle inequality.*

Proof: Let $w_{x_j}, w_{y_j}, w_{z_j}$ be the weights in the j'th network for edges x, y, z, respectively. Then, by the triangle inequality holding for the edges in each network,

$$w_{x_j} + w_{y_j} \geq w_{z_j} \quad \text{for} \quad j = 1, 2, \ldots, k \ .$$

Therefore,

$$w_x + w_y \geq w_{z_j} \quad \text{for} \quad j = 1, 2, \ldots, k \ .$$

This implies that $w_x + w_y \geq w_z$; that is, the triangle inequality holds in $G_{\max}$. ∎

Alternative Objective Functions: In k-*Network* p-*Center* we have considered an objective function of a bottleneck nature. That is, we try to minimize the cost of the maximum cost network. Some problems on multiple underlying networks, including such versions of p-*Center*, may for certain applications be more appropriately modeled with an objective function that rolls up the cost of each underlying network in some different way.

A practical objective is to combine solutions of different periods with associated discount factors. Another approach is to simply sum up the objectives during different periods. This however would be uninteresting for problems like the shortest path or matching, in the sense that we can replace the edge costs by the sum of edge costs. For the k-*Network* p-*Center* problem this would be an non-trivial objective to explore.

Appendix

A Multiple Underlying Network Versions of Other Problems

We first show that multiple underlying network versions of *Shortest Path*, *Minimum Spanning Tree*, *Minimum Weight Perfect Matching*, and *Minimum Cut* are all NP-hard, even for $k = 2$. All of the complexity proofs rely on a reduction from *Set Partition*, which is known to be NP-hard (see [GJ79]).

Problem Name: *Set Partition*

Instance: Given positive integers $a_1, a_2, \ldots, a_n$.

Objective: Partition the integers into two subsets so as to minimize the weight of the subset with larger weight.

After proving that each of the four problems listed above is NP-hard in a multiple period setting despite being polynomial for $k = 1$, we show that *Sorting* remains polynomial in this setting. Finally, because the recognition version of *Set Partition* is only *weakly* NP-complete, it follows that there may exist *pseudo-polynomial* algorithms for the problems shown in this appendix to be NP-hard; we provide such an algorithm for k-*Network Shortest Path.*

A.1 Shortest Path

Theorem A.1 *k-Network Shortest Path is NP-hard for any fixed $k > 1$.*

Proof: Given an instance of *Set Partition*, we construct an instance of k-*Network Shortest Path* with $k = 2$ underlying networks in which an optimal s-t path will correspond to an optimal set partition. Between $s = x_0$ and $t = x_n$ are *junction* nodes $x_1, x_2, \ldots, x_{n-1}$. There are two routes, a top and bottom route, to get from x_{i-1} to x_i (for $i = 1, \ldots, n$): the top route goes through t_i and the bottom route goes through b_i. The cost of edge $\{x_{i-1}, t_i\}$ is a_i and 0 in networks 1 and 2, respectively; the cost of edge $\{x_{i-1}, b_i\}$ is 0 and a_i in networks 1 and 2, respectively. (The edges connecting t_i and b_i to x_i are free.)

Observe that any s-t path corresponds to a selection of top or bottom routes for each pair x_i, x_{i+1}. The cost of the path in network 1 is the sum of top route costs, and the cost of the path in network 2 is the sum of bottom route costs; the overall cost of the path overall is the maximum of these two. Now, each s-t path can be seen to correspond to a set partition with the same objective. Thus, an optimal s-t path optimally solves the *Set Partition* instance. ■

A.2 Minimum Spanning Tree

The same reduction given for k-*Network Shortest Path* can also be used to establish that k-*Network MST*, the multiple underlying network version of the minimum spanning tree problem, is NP-hard.

Theorem A.2 *k-Network MST is NP-hard for any fixed $k > 1$.*

Proof: Using the reduction from Theorem A.1, we can see that an optimal spanning tree produces an optimal shortest path which, in turn, provides an optimal set partition. ■

A.3 Minimum Weight Perfect Matching

Here the objective is to select a matching so as to minimize the weight of the network whose matching edge weights have maximum sum.

Theorem A.3 *k-Network Minimum Weight Perfect Matching is NP-hard for any fixed $k > 1$.*

Proof: Given an instance of *Set Partition*, we construct an instance of k-*Network Minimum Weight Perfect Matching* with $k = 2$ underlying networks in which an optimal matching will correspond to an optimal set partition. Construct a a graph G made up of n components, where the i'th component, for $i = 1, \ldots, n$, contains vertices w_i, x_i, y_i, and z_i. Between w_i and x_i is an edge with weight a_i in network 1 and 0 in network 2; between w_i and y_i is an edge with weight 0 in network 1 and a_i in network 2; between z_i and each of x_i and y is an edge that has 0 cost in both networks.

Observe that any perfect matching corresponds to the weight a_i, for each $i = 1, \ldots, n$, either contributing to the weight of the matching in network 1 or in network 2, but not both. It follows that every perfect matching corresponds to a set partition of equal weight. Thus, an optimal perfect matching solves the *Set Partition* instance. ■

Note that the construction in the above theorem is of perfect matching in a bipartite graph. Thus the theorem can be strengthened to state that k-*Network Minimum Weight Perfect Matching* is NP-hard for any fixed $k > 1$, even on bipartite graphs.

A.4 Minimum Cut

Theorem A.4 *k-Network Minimum s-t Cut is NP-hard for any fixed $k > 1$.*

Proof: Given an instance of *Set Partition*, we construct an instance of k-*Network Minimum s-t Cut* with $k = 2$ underlying networks in which an optimal cut will correspond to an optimal set partition. Let the nodes in the network be s (source), t (sink), and $v_1, \ldots, v_n$. For each $i = 1, \ldots, n$, there exists an arc (s, v_i) with weight a_i in network 1 and weight 0 in network 2, and an arc (v_i, t) with weight 0 in network 1 and weight a_i in network 2.

Observe that in any s-t cut, the weight a_i, for each $i = 1, \ldots, n$, either contributes to the weight of the cut in network 1 or in network 2, but not both. It follows that every s-t cut corresponds to a set partition of equal weight. Thus, an optimal s-t cut solves the *Set Partition* instance. ■

A.5 Sorting

Consider a k-metric (sorting does not take place on a network) version of sorting in which each element in the group has k different measures measures. The objective in k-*Metric Sorting* is to "optimally" sort the elements so as to minimize the metric that is most poorly sorted on; the quality of a given sort for a given metric is defined to be the maximum *distance* in position for an element in the sort from its optimal position for the given metric, where any reasonably defined *distance* measure (in particular, the distance of an element from itself is 0 and all distances are non-negative) may be used. For $k = 1$, standard sorting solves the problem optimally, with objective value 0.

We show that k-*Metric Sorting* can be optimally solved by formulating it as a bottleneck version of the assignment problem, which can in turn be solved through a series of assignment problems. Given an instance of k-*Metric Sorting* on n elements, construct an instance of the bottleneck assignment problem in which the n elements are to be assigned to n positions such that the element with the highest cost assignment is minimized. The cost of assigning any particular element to a particular position is equal to the maximum cost, among the k metrics, of placing the element in this position.

This problem can be solved by finding the minimum weight such that there exists an assignment of elements to positions such that no pairing has a cost exceeding the weight. As the optimal solution is one of the element/position costs, and the check for each weight can be carried out by solving a bipartite matching problem, an optimal assignment can be found via $O(\log n)$ calls to bipartite matching. As a bipartite matching problem on n nodes and m edges can be solved in $O(\sqrt{n}m)$ time (see [ET75]), and because $m = O(n^2)$, the overall running time is $O(n^{\frac{5}{2}} \log n)$.

A.6 Shortest Path Revisited

We provide a pseudo-polynomial time algorithm for k-*Network Shortest Path*, in which the objective is to find optimal paths from a single source s. Let V be the set of nodes in the directed graph $G = (V, A)$. Let d^i_{xy} be the distance on the arc from x to y in the i'th network; all distances are assumed to be non-negative. Let D be the maximum arc distance in any network.

Define

$$f(x; c_1, ..., c_k)$$

to be a boolean-valued function that indicates whether or not there exists an s-x path of cost c_1 in network 1, c_2 in network 2, etc. The cost of the optimal path between s and any node x is then found by scanning all $f(x; \ldots) = 1$ and finding the one with the smallest $\max\{c_1, \ldots, c_k\}$.

Values of f can be calculated recursively:

$$f(y; c_1, ..., c_k) \;=\; \bigvee_{(x,y)\in A} f(x; c_1 - d^1_{xy}, \ldots, c_k - d^k_{xy}) \;.$$

$f(x; c_1, ..., c_k)$ is defined to be 0 if any c_i is negative. Observe that we need not consider any cost indices for f that exceed nD, where $|V| = n$, as an optimal path from s to any other node is guaranteed not to exceed this value. Also observe that it is not difficult to keep track of the path(s) associated with any set of indices for which f is true.

We now show that the above procedure solves k-*Network Shortest Path* in pseudo-polynomial time. $O(n)$ for each recursive function calculation; this includes the work required in the final scan to determine the optimal path to a given vertex. As there are $O(n)$ vertices, and because the cost index is bound by nD in each network, there are $O(n(nD)^k)$ indices at which the function f is to be calculated for the all-pairs problem. This implies an overall $O(n^{k+2}D^k)$ running time. For k fixed this running time is pseudo-polynomial.

References

[BGKS96] R. Bhatia, S. Guha, S. Khuller and Y. Sussmann. Facility Location with Dynamic Distance Functions. Unpublished manuscript, University of Maryland, 1996.

[Das78] M. Daskin. *Location, Dispatching and Routing Models for Emergency Services with Stochastic Travel Times*, pages 224–265. Spatial Analysis and Location-Allocation Models. Van Nostrand Reinhold Co., New York, 1978. A. Gosh and G. Rushton (eds.).

[ET75] S. Even and R. E. Tarjan. Network Flow and Testing Graph Connectivity. *SIAM Journal on Computing*, 4:507–518, 1975.

[GJ79] M.R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, 1979.

[HS85] D. S. Hochbaum and D. Shmoys. A best possible heuristic for the k-center problem. *Mathematics of Operations Research*, 10(2):180–184, May 1985.

[HS86] D. S. Hochbaum and D. Shmoys. A Unified Approach to Approximation Algorithms for Bottleneck Problems. *Journal of the Association for Computing Machinery*, 33(3):533–550, July 1986.

[KH79] O. Kariv and S. L. Hakimi. An Algorithmic Approach to Network Location Problems: The p-Centers. *SIAM Journal of Applied Mathematics*, 3:513–538, 1979.

[MO79] P. Mirchandani and A. Odoni. Locations on Medians on Stochastic Networks. *Transportation Science*, 13:85–97, 1979.

[OGY93] D. S. Hochbaum O. Goldschmidt and G. Yu. A Modified Greedy Heuristic for the Set Covering Problem with Improved Worst Case Bound. *Information Processing Letters*, 48:305–310, 1993.

[SM96] D. Serra and V. Marianov. The Location of Emergency Services in a Changing Network: The Case of Barcelona. Edmonton, Canada, June 1996. ISOLDE VII.

[SR96] D. Serra and C. ReVelle. The Maximum Capture Problem with Uncertainty. *Environment and Planning B*, 62:49–59, 1996.

[WC83] J. Weaver and R. Church. Computational Procedures for Location Problems on Stochastic Networks. *Transportation Science*, 17:168–180, 1983.