Upgrading Shortest Paths in Networks

Bistra Dilkina, Katherine J. Lai, and Carla P. Gomes

Computer Science Department, Cornell University
{bistra,klai,gomes}@cs.cornell.edu

Abstract. We introduce the Upgrading Shortest Paths Problem, a new combinatorial problem for improving network connectivity with a wide range of applications from multicast communication to wildlife habitat conservation. We define the problem in terms of a network with node delays and a set of node upgrade actions, each associated with a cost and an upgraded (reduced) node delay. The goal is to choose a set of upgrade actions to minimize the shortest delay paths between demand pairs of terminals in the network, subject to a budget constraint. We show that this problem is NP-hard. We describe and test two greedy algorithms against an exact algorithm on synthetic data and on a real-world instance from wildlife habitat conservation. While the greedy algorithms can do arbitrarily poorly in the worst case, they perform fairly well in practice. For most of the instances, taking the better of the two greedy solutions accomplishes within 5% of optimal on our benchmarks.

1 Introduction

Many applications in areas as diverse as VLSI circuit design, QoS routing, and traffic engineering involve designing networks under constrained shortest paths and budget limits. For example, in a transportation network, a key goal is to connect major cities via short routes to better serve the bulk of the traffic. In a multicast communication setting where a single node is broadcasting to a set of subscribers, it is important to minimize the latency, or the shortest path delays between the source node and all the subscribers. In wildlife conservation, our motivating application from computational sustainability [8], the landscape connectivity between important habitat patches is measured as the length of the shortest path in terms of landscape resistance to animal movement. Maintaining good landscape connectivity, i.e. short resistance paths, is key to resilient wildlife populations in an increasingly fragmented habitat matrix.

In this work, we introduce a new general network improvement problem relevant in such settings. The problem is defined with respect to a network with node delays where the delay between a pair of nodes is the shortest path delay in the network, while the overall delay of the network is measured as the average delay among a designated set of node pairs. Given a set of node upgrade actions with respective costs and upgraded node delays, we seek to choose the best possible upgrade strategy in terms of minimizing total upgrade cost and resulting overall

T. Achterberg and J.C. Beck (Eds.): CPAIOR 2011, LNCS 6697, pp. 76–91, 2011.

network delay. We refer to this problem as the *Upgrading Shortest Paths Problem*. We consider two optimization settings. In the budget-constrained setting, the goal is to find an upgrade strategy such that the total upgrade cost does not exceed a given budget B, and the resulting upgraded network has minimum overall delay over all possible strategies which obey the budget constraint. On the other hand, in the delay-constrained setting, the goal is to find a minimum-cost set of nodes to be upgraded so that the overall delay in the resulting network meets a given bound D.

Some network improvement problems have been studied previously. In particular, most of the previous work has concentrated on the edge-delay variant where either edges can be upgraded directly, or nodes are to be upgraded, effectively upgrading all the edges incident to the upgraded nodes. Many of the studies assume a particular relationship between the delays and the upgraded delays. For example, if a node v is upgraded, then the delay of each edge incident to v reduces by a factor x where $0 \leq x < 1$, and if both endpoints of an edge are upgraded, then its delay reduces by a factor of x^2. Paik et al. [15] introduce several NP-hard network improvement problems under this upgrade model and unit costs, including the minimum-cost network improvement problem subject to a maximum delay constraint over all pairs of nodes in graph. Krumke [11] studies a similar network improvement problem but the constraint is on the total delay of the minimum spanning tree of the resulted upgraded network.

Although the edge-delay setting has been studied more than its node counterpart, placing the delays on the nodes can be more appropriate in certain applications. For example, in telecommunications, expensive equipment such as routers and switches are at the nodes of the underlying network. Unfortunately, while one can easily reduce an edge-weighted version to a node-weighted version, the reverse does not usually hold for undirected graphs, and hence it is desirable to work directly on node-weighted problems in undirected graphs. In this work, we address the more general node-weighted variant.

Landscape Connectivity

Although the network optimization problem considered here is very general, the main motivating application for our work is in Conservation Planning.

Habitat fragmentation is one of the principal threats to biodiversity. The focus of much ecology research is to quantify *landscape connectivity* [17], a measure of the degree to which the landscape facilitates or impedes movement among habitat patches. The landscape is represented as a set of small parcels or pixels, each of which has a resistance value that gives the species-specific cost of moving through particular landscape features. Resistance models are inferred by relating landscape characteristics to genetic distance between individuals at different locations [4] or to radio-collar movement data. Under the Least-Cost Path model, the connectivity between two designated habitat patches is measured as the length of the shortest resistance-weighted path between them [16].

Preserving and restoring connectivity for broad-scale ecological processes, such as dispersal and gene flow, has become a major conservation priority [3]. While conservation biology has historically set conservation objectives and plans irrespective of their cost, multiple studies in recent years have shown that systematic conservation planning is the right approach. It is possible to achieve conservation objectives at a fraction of the cost (or achieve higher targets for the same cost) if the conservation and management costs are formally considered at the outset of the planning process [13, 10]. Decision-support tools to design efficient budget-constrained conservation strategies are needed and yet still largely lacking.

By reducing the problem of maximizing landscape connectivity to the Upgrading Shortest Paths problem, we provide conservation planners with a tool to evaluate trade offs between costs and connectivity benefits as well as generate conservation plans with formal optimality guarantees. In particular, we can model the pixels or parcels of land as nodes in the graph, and edges are drawn between parcels that share boundaries. The resistance of each parcel is the corresponding node delay, and its upgraded delay is the predicted effective resistance of the parcel if it were under conservation management. Given pairs of important habitat patches (i.e. existing conserved areas or subpopulations), solving the combinatorial optimization problem designs a conservation strategy that maximizes the resulting landscape connectivity.

Recently, the related problem of Wildlife Corridor Design was studied in [2, 9, 5]. In the optimization model used for designing wildlife corridors, the goal is to maximize the total utility of the set of bought parcels while ensuring that the parcels connect a designated set of reserves and that the total cost does not exceed a specified budget. By enforcing connectivity of the purchased parcels, it in effect pessimistically assumes that any land parcel that is not bought for the wildlife corridor is no longer usable by the wildlife. In reality, choosing not to buy a piece of land may not significantly impact whether wildlife will still be able to use the land. In the landscape connectivity conservation problem discussed in this work, each land parcel may contribute to the connectivity of the terminals, whether or not it has been bought. The benefit of buying a piece of land is reflected by decreasing the land's effective resistance.

Our Contributions

In this paper, we introduce the Upgrading Shortest Paths problem, a new combinatorial problem for improving network connectivity in many real-world applications. We show that this problem is NP-hard. We give a formulation of the problem as a multicommodity flow mixed integer program for solving it to optimality, as well as two fast greedy algorithms. We tested these approaches on various synthetically generated planar graph instances and a real-world instance from conservation planning, and we found that our MIP formulation scales surprisingly well to instances with hundreds of nodes. While the greedy algorithms can perform arbitrarily badly even in planar graphs, they performed fairly well,

coming within 5% of optimal on most of these test instances. One interesting phenomenon we observed is that the hardness of the instances is very much correlated with the nature and magnitude of the generated upgraded delay values for the nodes. Changes in node delays that were large in magnitude and varying greatly from node to node resulted in longer running times for the MIP and larger optimality gaps for the greedy algorithms.

The paper is organized as follows. First, we formally define the Upgrading Shortest Paths Problem. Second, we characterize its computational complexity. Third, we describe the three solution approaches. Finally, in the experiments section we study their typical case behavior on a synthetic dataset and present results for an instance derived from a real conservation planning setting.

2 The Upgrading Shortest Paths Problem

2.1 Problem Definition

We can define an instance of the decision version of the Upgrading Shortest Paths (USP) problem as follows.

Definition 1 (The Upgrading Shortest Paths Problem)

Given: *an undirected graph $G = (V, E)$, a set of terminal pairs $P \subseteq V \times V$, a cost function on the nodes $c : V \to \mathbb{R}^+$, a delay function $d : V \to \mathbb{R}^+$, a delay function $d' : V \to \mathbb{R}^+$ where $d'(v) \leq d(v)$ for all $v \in V$, a budget value $B \geq 0$, and a delay value $D \geq 0$.*

Find: *a set of nodes $V' \subseteq V$ such that $\sum_{v \in V'} c_v \leq B$, and the average shortest path for pairs in P is at most D when evaluated under the effective delays:*

$$\hat{d}(v) = \begin{cases} d'(v) & \text{if } v \in V' \\ d(v) & \text{otherwise} \end{cases} \tag{1}$$

For convenience, we also define T to be the set of all terminals, or set of nodes that appear in at least one pair $p \in P$. We can also define the following two variations of the USP problem.

Definition 2 (Budget-constrained USP Problem). *The delay value D is not given as an input, and the objective is to find a set of nodes $V' \subseteq V$ such that $\sum_{v \in V'} c_v \leq B$, and the average shortest path for pairs in P is minimized.*

Definition 3 (Delay-constrained USP Problem). *The budget value B is not given as an input, and the objective is to find a set of nodes $V' \subseteq V$ such that the average shortest path for pairs in P is at most D, and the total cost $\sum_{v \in V'} c_v$ is minimized.*

2.2 Computational Complexity

We now show that the two variants of the USP problem are NP-hard.

Theorem 1. *The budget-constrained Upgrading Shortest Paths Problem is NP-hard.*

Proof. To show that budget-constrained USP is NP-hard, we use a reduction from the knapsack problem which is NP-hard [7]. In a knapsack instance, we are given items indexed $\{1, \ldots, n\}$ with sizes $\{c_1, \ldots, c_n\}$ and values $\{d_1, \ldots, d_n\}$. The goal is to find some subset S that maximizes $\sum_{i \in S} d_i$ subject to the constraint that $\sum_{i \in S} c_i \leq B$, where B is the capacity of the knapsack.

Let G be a path graph with endpoints s and t, and n interior points v_i, one for each item in the knapsack instance. Note that the only shortest path between s and t is the entire path. We can now construct a USP instance with the graph G, one terminal pair (s, t), and a budget value of B. The nodes s and t have zero cost and delay. Each intermediate node v_i has cost c_i, delay d_i, and upgraded delay of 0. We can now map a set of items S exactly to the set of nodes in G that represent them, and this set of nodes has total cost $\sum_{i \in S} c_i$ and improves the total shortest path length by $\sum_{i \in S} d_i$ when bought. Since the optimal solution to the USP instance minimizes the shortest path length while satisfying the budget constraint, it in effect finds the set of nodes with the maximum total decrease in delay, thus exactly solving the knapsack instance. Therefore, the budget-constrained USP is NP-hard. Instances that involve more complicated graphs than a simple path graph can be viewed as having multiple knapsack instances to choose from, and instances with more than one demand pair may have overlapping knapsack instances where items are bought once but may contribute to multiple knapsacks.

Theorem 2. *The delay-constrained USP problem is NP-hard and can only be approximated within an $\Omega(\log |V|)$ factor unless P=NP.*

Proof. To show this result directly, we use a reduction from set cover which has the same hardness results [1]. In a set cover instance, we are given a universe of elements $U = \{1, \ldots, n\}$, a family $\mathcal{S}$ of candidate sets S_j each of which has a cost c_j. The goal is to find a family of sets $\mathcal{C} \subseteq \mathcal{S}$ such that they cover all of the elements, i.e. $\cup_{S \in \mathcal{C}} S = U$, and such that the total cost of the sets in $\mathcal{C}$ is minimized. We can construct a USP instance where there is a zero-delay node v_i for each element i in U, and the terminal pairs set P is composed of all pairs of these nodes. Each set S_j is similarly represented by a node u_j with delay 1, cost c_j and upgraded delay 0. Each node u_j is connected to the nodes v_i for which $i \in S_j$ as well as every other node u_k. Thus the shortest path delay between any two distinct nodes is at least 1, and upgrading a set of nodes u_j such that every node v_i is adjacent to at least one of these nodes decreases all of the delays to 0. Thus if we set the target average delay D to 0 and minimize the cost necessary

to achieve this delay, we are exactly solving the set cover problem, and the cost of the nodes in the optimal solution is equal to the cost of the sets in the set cover instance.

3 Solution Methods

In this section, we present an exact method for solving the two variations of the USP problem using a MIP formulation as well as two greedy algorithms for the budget-constrained variation. To evaluate the quality of an approximation algorithm or a heuristic, it is standard to calculate the optimality gap of a solution by taking the difference between the approximate and exact solutions and dividing this result by the optimal value. However, this is a problematic and uninformative measure for the budget-constrained problem. For example, if the best upgraded shortest paths all have delay 0 and a heuristic finds a nearly-optimal solution of average delay ϵ, the heuristic still has an infinite optimality gap. For the sake of evaluating the performance of our solution methods, for the rest of this paper we will regard the objective function for the budget-constrained problem as maximizing the improvement in the average shortest path delay.

For all of the methods we present, we can prune the search space by eliminating all nodes $v \in V$ for which upgrading the node will never improve the delay between any terminal pair in P. To find these nodes, we first calculate single-source shortest paths from each terminal to the rest of the nodes in both the graph with no upgrades and the graph with all nodes upgraded. A node v will never improve the delay for a terminal pair p if the shortest path for p with no upgrades is shorter than the shortest possible path passing through v in the fully upgraded graph. If this condition holds for all terminal pairs, then under no upgrade strategy would v ever improve the objective, and hence we can safely prune it.

3.1 Mixed Integer Programming

We can solve the Upgrading Shortest Paths problem exactly by formulating it as a mixed integer program (MIP). We use a multicommodity flow formulation for computing the shortest delay path for each terminal pair $p = (s, t) \in P$. For this formulation, we transform the given undirected graph G to a directed graph G' where delays now appear on the edges instead of the nodes. Each node v in the graph is replaced by two nodes, the "in" node v^- and the "out" node v^+, that are then connected with two parallel edges directed from v^- to v^+. We refer to these edges as the "original node edge" e_v and the "upgraded node edge" e'_v, and their delays are set to the original and upgraded delays of the node v, respectively. Each undirected edge $\{u, v\}$ in the graph becomes two directed edges (u^+, v^-) and (v^+, u^-) with delay 0. See Figure 1 for an example of an edge in G and its corresponding subgraph in G'. We can now state the flow formulation for the constructed graph G'.

We now describe the variables used in our formulation:

- x_v: binary variable indicating whether node $v \in V$ is to be upgraded.
- *cost*: the total cost of all upgraded nodes.
- f_{pe}: continuous variable indicating the flow of commodity p on edge e, i.e. whether edge e is chosen to be on the shortest path for the terminal pair p.
- f_{pv}: continuous variable indicating the flow of commodity p on edge e_v. In an integral solution, this indicates whether the original node v is chosen to be on the shortest path for the terminal pair p.
- f'_{pv}: continuous variable indicating the flow of commodity p on edge e'_v. In an integral solution, this indicates whether the upgraded node v is chosen to be on the shortest path for the terminal pair p.
- $delay_p$: variable for the effective shortest path delay for terminal pair p.
- *avgdelay*: variable for the delay over all terminal pairs.

The full MIP for the budget-constrained problem is shown in Equations (2)-(18). The delay-constrained MIP is a simple modification of this MIP where the objective function minimizes *cost* instead, and Constraint (16) is replaced by the constraint $avgdelay \leq D$. We use Constraints (3)-(10) to model each terminal pair's shortest delay path as a multicommodity flow problem. For each terminal pair (s, t), Constraints (3)-(8) force the nodes s and t to be the source and sink of one unit of flow, respectively. We use $\delta^-(v^-)$ to indicate the set of incoming edges to the node v^- and $\delta^+(v^+)$ to indicate the set of outgoing edges from node v^+. The next constraints (9)-(10) enforce flow conservation through the rest of the nodes in the graph. The total delay for a terminal pair p is equal to the sum of delays of each edge e, scaled by the flow f_{pe} going through it (Constraint (13)).

Constraints (11)-(12) ensure that if a node v is chosen to be upgraded, only the upgraded node edge e'_v can carry flow; the original node edge e_v is not to be used. Similarly, if a node v is not chosen to be upgraded, only the original node edge e_v can be used to carry flow. Constraints (14) and (15) compute the total cost of the upgraded nodes and the average delay of all terminal pairs, respectively. Constraint (17) enforces that the upgrade decision variables are binary, and Constraint (18) enforces that the flow variables are all non-negative.

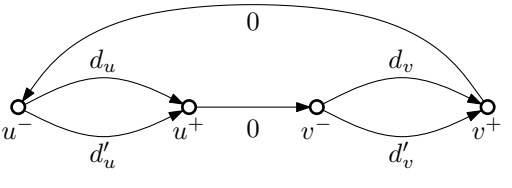


Fig. 1. The representation of nodes u, v, and an undirected edge between them in the new directed graph G'. The delay of each edge is labeled.

$$\min \quad avgdelay \tag{2}$$

$$\text{s.t. } f_{ps} + f'_{ps} = 1 \qquad \forall p = (s,t) \in P \tag{3}$$

$$\sum_{e \in \delta^-(s^-)} f_{pe} = 0 \qquad \forall p = (s,t) \in P \tag{4}$$

$$f_{ps} + f'_{ps} = \sum_{e \in \delta^+(s^+)} f_{pe} \qquad \forall p = (s,t) \in P \tag{5}$$

$$f_{pt} + f'_{pt} = 1 \qquad \forall p = (s,t) \in P \tag{6}$$

$$\sum_{e \in \delta^-(t^-)} f_{pe} = f_{pt} + f'_{pt} \qquad \forall p = (s,t) \in P \tag{7}$$

$$0 = \sum_{e \in \delta^+(t^+)} f_{pe} \qquad \forall p = (s,t) \in P \tag{8}$$

$$\sum_{e \in \delta^-(v^-)} f_{pe} = f_{pv} + f'_{pv} \qquad \forall p = (s,t) \in P, \forall v \neq s,t \in V \tag{9}$$

$$f_{pv} + f'_{pv} = \sum_{e \in \delta^+(v^+)} f_{pe} \qquad \forall p = (s,t) \in P, \forall v \neq s,t \in V \tag{10}$$

$$f'_{pv} \leq x_v \qquad \forall p = (s,t) \in P, \forall v \neq s,t \in V \tag{11}$$

$$f_{pv} \leq 1 - x_v \qquad \forall p = (s,t) \in P, \forall v \neq s,t \in V \tag{12}$$

$$delay_p = \sum_{v \in V} [d(v) f_{pv} + d'(v) f'_{pv}] \qquad \forall p \in P \tag{13}$$

$$cost = \sum_{v \in V} c(v) x_v \tag{14}$$

$$avgdelay = \frac{1}{|P|} \sum_{p \in P} delay_p \tag{15}$$

$$cost \leq B \tag{16}$$

$$x_v \in \{0,1\} \qquad \forall v \in V \tag{17}$$

$$f_{pe}, f_{pv}, f'_{pv} \geq 0 \qquad \forall p \in P, e \in E, v \in V \tag{18}$$

For both of the minimization problems, we implement pruning by setting $x_v = 0$ for all nodes $v \in V$ for which upgrading the node will never improve the delay between any terminal pair in P. For each node v that will never improve the delay for some particular pair $p \in P$, we add the constraint $f'_{pv} = 0$.

3.2 A Naive Greedy Algorithm

One naive approach for the budget-constrained USP problem is to take the current shortest paths between terminal pairs and upgrade them as much as possible. This cuts down on the search space a great deal. Intuitively, the greedy algorithm sorts the nodes in decreasing order by their heuristic *value* and attempts to upgrade each node in the list with what is left of the budget. To define the value for each node, the greedy algorithm first sets the values of every

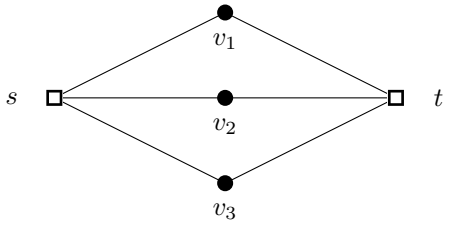


Fig. 2. In this example, the naive greedy algorithm will only examine the nodes v_1 and v_3 since they are part of the current and best possible paths. Under a budget constraint of B, the naive greedy algorithm will make no improvement to the delay even though upgrading v_2 would decrease it to an arbitrarily small $\epsilon > 0$.

node to 0. Then, for each pair of terminals $p = (s, t)$, it adds $(d(v) - d'(v))/c(v)$ to the value of each node v on the shortest path between s and t. The total running time is that of running Dijkstra's shortest paths algorithm from each terminal, sorting the eligible nodes, and adding them in linear time. In total this algorithm takes $O(|T|(|E| + |V| \log |V|))$ time, where $T = \{t : \exists(s, t) \in P\}$ is the set of terminals that show up in some terminal pair.

A similar alternative to the way this heuristic cuts down on its search space is to consider only the nodes on the shortest paths that would exist if the entire graph were upgraded; these are the best possible paths if the budget were infinite. It is a simple matter to run both heuristics and take the better result; we will call this combined approach the *Naive Greedy* algorithm. As with many heuristics, this algorithm does not have a provable guarantee. In fact, it can do arbitrarily poorly as shown in the example in Figure 2.

3.3 An Iterative Greedy Algorithm

Further improvement on the naive greedy algorithm can be gained by considering nodes that are not considered by the naive greedy algorithm. After pruning and eliminating the nodes that could never improve the delay for any terminal pair (as described earlier in Section 3.1), we again assign a heuristic value to each eligible node. Here, we redefine a node's value to be the change in the average shortest path delay if we were to upgrade that one node, divided by its cost. The new greedy algorithm iteratively upgrades the node with the highest value and recomputes the remaining nodes' values before upgrading the next node. After the algorithm exhausts the budget, it is possible that some of the nodes it chose to upgrade no longer improve the solution. As such, it removes these unnecessary nodes from the solution and starts over again but with the current set of upgraded nodes and the leftover budget. We can repeat this process until there is no longer any improvement made on the objective function, or we can set a limit to the number of times that this is run. We will call this the *Iterative Greedy* algorithm, and more detailed pseudocode is outlined in Algorithm 1.

The time complexity of this greedy algorithm is dominated by calls to the function `CalcShPaths()`. Calculating single-source shortest paths for all of the

Algorithm 1. The Iterative Greedy Algorithm

Input: input of the USP instance, a parameter NumIters, and the subroutines
- `CalcShPaths(`G, T, d, d', V'`)`: shortest path delays from the nodes $t \in T$ to all other nodes $v \in V$ assuming the nodes in V' have been upgraded
- `CalcAvgDelay(pathDists,` P`)`: average delay for pairs in P
- `CalcImpr(pathDists,` P, v`)`: improvement in average delay for P if v is upgraded

Output: A set $\mathsf{V}' \subseteq V$ to upgrade

```
1  V' ← ∅
2  spent ← 0
3  for i ← 1 to NumIters do
4      Q ← V − V'
5      pathDists ← CalcShPaths(G, T, d, d', V')
6      startAvgSP ← CalcAvgDelay(pathDists, P)
7      while Q ≠ ∅ do
8          foreach v ∈ Q do
9              if spent + c(v) ≤ B then value(v) ← CalcImpr(pathDists,P,v)/c(v)
10             else Q ← Q − {v}
11         if Q ≠ ∅ then
12             Let v ∈ Q be the node for which value(v) is maximum
13             V' ← V' + {v}
14             Q ← Q − {v}
15             spent ← spent + c(v)
16             pathDists ← CalcShPaths(G, T, d, d', V')
17     deleted ← false
18     avgSP ← CalcAvgDelay(pathDists, P)
19     foreach v ∈ V' do
20         if avgSP = CalcAvgDelay(CalcShPaths(G, T, d, d', V' − {v}), P)
           then
21             V' ← V' − {v}
22             spent ← spent − c(v)
23             deleted ← true
24     if deleted = false or avgSP = startAvgSP then return V'
25 return V'
```

terminals is implemented by running Dijkstra's algorithm $|T|$ times using Fibonacci heaps, which takes a total of $O(|T|(|E| + |V|\log|V|))$ time. In each iteration of the greedy algorithm, i.e. each iteration of the for loop starting at Line 3, this function is called $O(|V'|)$ times. Having computed the shortest paths, the function `CalcAvgDelay()` takes $O(|P|)$ time to look up the shortest path delay for each terminal pair. The function `CalcImpr()` needs to calculate the upgraded delays of the shortest paths that must pass through v. This can be done in $O(1)$ time for each terminal pair (for a total of $O(|P|)$ time for the function) by adding up the shortest path delays from the node v to the two terminals, removing the delay $d(v)$ from both paths, and adding the upgraded delay $d'(v)$. Since the running times of these other functions and the various loops are all dominated by the running time of the shortest-paths computations, the total running time for each iteration of the greedy algorithm can be bounded above by $O(|V||T|(|E| + |V|\log|V|))$.

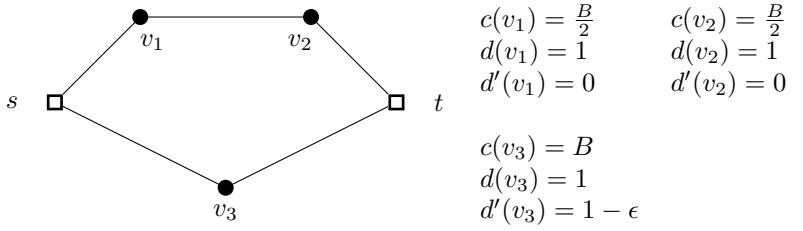


Fig. 3. In this example, there is one terminal pair $p = (s, t)$, and the initial shortest path length is 1 via node v_3. Under a budget constraint of B, the greedy algorithm ignores both nodes v_1 and v_2 in favor of v_3.

In terms of performance guarantees, this greedy algorithm can also perform arbitrarily poorly. Greedy algorithms occasionally have provable guarantees in some problems such as maximizing submodular functions [14, 6, 12]. Submodular functions capture settings where the payoff for choosing some set of items exhibits diminishing returns, i.e. they are functions f that satisfy $f(A \cup B) \leq f(A) + f(B) - f(A \cap B)$. In a recent result, Lin and Bilmes [12] show that it is possible to use a modification of our iterative greedy approach to get a constant approximation when maximizing a submodular function subject to a budget constraint. Their algorithm takes the better of the two solutions of a) performing the greedy algorithm and b) choosing the single element x for which $c(x) \leq B$ and $f(x)$ is maximized. Unfortunately, their small modification fails to work here because the budget-constrained USP problem, when posed as a maximization problem under the choice of V', is not submodular. We give an example in Figure 3. Buying either of the nodes v_1 or v_2 alone does not decrease the shortest path length, but buying both of them decreases it by 1 (and is in fact the optimal solution). The iterative greedy algorithm would preferentially add nodes that immediately improve the objective function, so it would choose v_3 and use up the entire budget in the process. Choosing the one element that improves the objective the most also gives the same result. Since this is only an improvement of ϵ to the optimal improvement of 1, the performance of the greedy algorithm can be made arbitrarily worse by setting ϵ to be an arbitrarily small but positive value.

4 Experimental Results

We implemented and tested the greedy algorithms against the exact solution provided by using a MIP solver. To test these algorithms, we created synthetic problem instances on square grid graphs. The initial delay and cost of each node was chosen uniformly at random from the range [50, 1000]. Three terminal pairs were chosen from a set of four randomly chosen terminals (two set in opposite corners of the graph) by taking the edges in the all-pairs shortest-paths minimum spanning tree on the terminals. Three approaches were used to model the upgraded delay function:

scaled. Each upgraded delay value is some scalar factor times the original delay value, i.e. $d'(v) = cd(v)$ for some $c \in [0, 1]$.

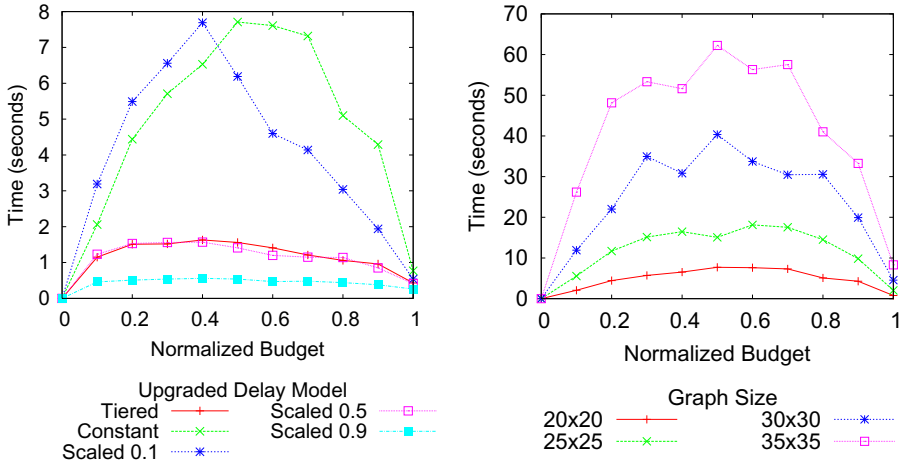
constant. Each upgraded delay is equal to the same constant 50.

tiered. For nodes with delay value $d(v)$ in the range (500, 1000), the upgraded delay value is 500, and those with $d(v)$ in the range [100, 500], $d'(v)$ is set to 75.

We first tested the performance of solving the MIP encoding exactly by using IBM ILOG CPLEX 11 on 100 instances of 20 by 20 grid graphs (400 nodes) with pruning as described earlier. We varied the budget value B between 0 and $B_{\max}$, the total budget necessary to achieve the shortest possible delays. Each value $B_{\max}$ is specific to the instance and is calculated by solving the budget minimization MIP. As shown in Figure 4a, the problem exhibits easy-hard-easy behavior as the budget is increased. It is notable on these instances that the easy-hard-easy trend is most pronounced for the **constant** model and the **scaled** model for $c = 0.1$. As a general trend, instances where the change in node delays are larger and vary a great deal are harder than instances where the new node delays represent very little change. The MIP scaled surprisingly well for larger grid graphs, as can be seen in Figure 4b.

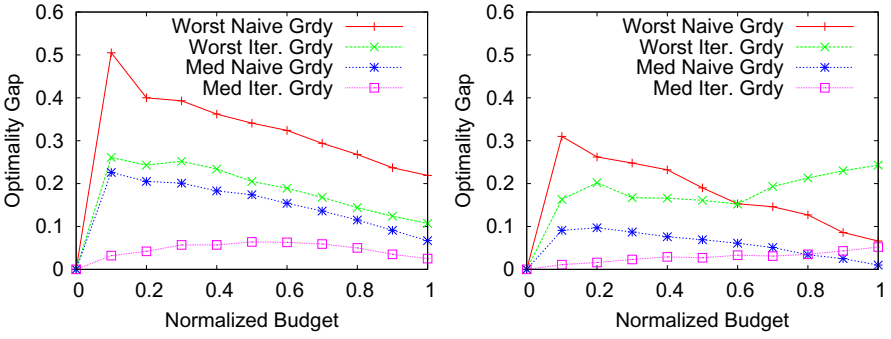
4.1 Greedy Algorithm Performance

The naive greedy algorithm does not always perform very well, though it sometimes outperforms the iterative greedy algorithm when the budget nears $B_{\max}$. The iterative greedy algorithm performed very well on these randomly generated



(a) All models (b) **constant**

Fig. 4. (a) Median MIP running times for different upgraded delay models on 100 instances of 20x20 grid graphs with 3 terminal pairs. The budget ranges from 0 to 100% of the maximum budget necessary for the instance. (b) Median MIP running times for different grid graph sizes under the **constant** upgraded delay model.



(a) **constant** (b) **tiered**

Fig. 5. The worst and median performances of the greedy algorithms are given from running on 100 instances of 20x20 grid graphs on two of the upgraded delay models

instances. Both the median and mean performance of the algorithm on 100 samples were within 5% of optimal for all of the upgraded delay models. As expected, there were occasionally instances where the algorithm did poorly, though given the nature of our synthetic instances, this did not occur for many instances, nor was the result ever found to be worse than 60% of optimal. Figure 5 shows the average and worst case performances of the greedy algorithm on our data set.

By the heuristic nature of the greedy algorithms, both of the greedy implementations were very fast. In our experiments, the naive greedy algorithm finished in at most 0.02 seconds, and the iterative greedy algorithm finished in at most 0.5 seconds.

4.2 Results on Grizzly Bear Data

We apply our solution approaches to data derived from a real conservation setting. We use the data for the grizzly bear corridor design problem studied in [2]. The goal in this work was to ensure connectivity between three major national conservation parks with existing grizzly populations. The data was compiled by Dr. Jordan Suter and is given in terms of habitat suitability, or *utility*, values and costs for different land parcels in the geographical area surrounding the three wildlife reserves. For each land parcel, we generated landscape resistance values that were inversely correlated with their utility values. In many ecological studies, habitat suitability and resistance are treated as complementary values. Hence, we compute the resistance of nodes on the same scale as the utilities $resist(v) = \min_{u \in V} util(u) + \max_{u \in V} util(u) - util(v)$.

At a 10 by 10 km pixel resolution, the resulting network has 3299 parcels (nodes) and 3 terminal pairs (connecting the three reserves). We solved the 10km grizzly instance for different resistance models for both the budget-constrained and delay-constrained formulations. Figure 6 presents results from the **scaled** model for $c = 0.1$. The other resistance models behaved qualitatively similarly, although this was the most computationally demanding setting for the MIP formulation. The graph on the left plots the Pareto frontier between cost and

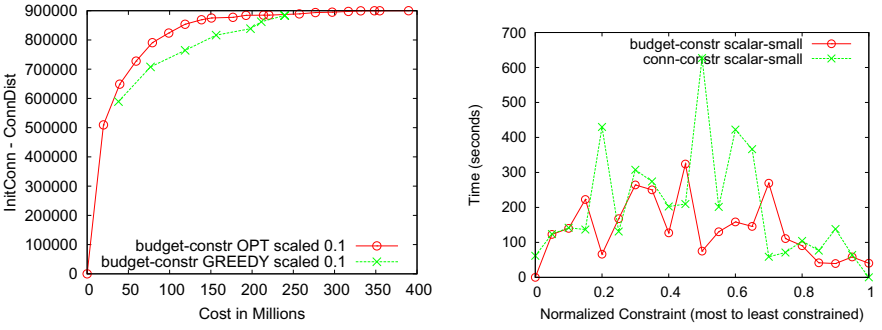


Fig. 6. Results for the 10km grizzly instance for the scaled 0.1 resistance model. Left plot shows the tradeoff between the cost spend and the improvement in delay achieved by the optimal as well as the iterative greedy solutions. Right plot shows the running time of both the budget-constrained and delay-constrained formulations as a function of the tightness of the respective constraint.

delay, i.e. the tradeoff curve of improvement in average terminal pair delay as we increase the budget allowed for upgrades. Such analysis can provide important insight for conservation planning as a small fraction of the maximum budget is enough to achieve more than half of the connectivity improvement. The graph on the right plots the computation time versus the normalized constraint for both constrained variants of the problem. For all resistance models, the minimum cost delay-constrained problem usually required more time to solve to optimality than the minimum delay budget-constrained variant. While the wildlife corridor design problem cannot be solved to optimality within hours for this instance [5], the respective Upgrading Shortest Paths problem on the same graph is solvable to optimality in a practical time frame.

5 Conclusions and Future Work

In this paper, we introduced the USP problem, a new NP-hard combinatorial problem for improving network connectivity in real-world applications. We also provided a MIP formulation that scaled very well with the size of the graph. This was a surprisingly positive result given the bad scaling behavior of MIP formulations for many other combinatorial network design problems. This is also a very practical result because in the context of conservation planning, problem instances are usually quite large, on the order of thousands of nodes. The greedy algorithms provided very high quality solutions in practice and can be used for extremely large instances.

The introduction of the USP problem also opens up several interesting open problems. Our greedy algorithms perform well but can do arbitrarily poorly. An open research direction is to design approximation algorithms with provable performance guarantees. It would also be interesting to study exactly why this MIP scales so well as compared to other combinatorial network design problems.

In the context of conservation planning, our model can also be generalized to capture other features such as multiple species of wildlife that have different resistance values for the same land parcel. We can also study the generalized model where each node can have different upgraded delay values available at different costs. This would model more fine-tuned applications where there is a discrete spectrum of actions that can be taken to decrease the inherent delay or resistance of a node.

Acknowledgments

This research was supported by NSF Expeditions in Computing award for Computational Sustainability (Grant 0832782) and by the USDA Forest Service, Rocky Mountain Research Station (10-JV-11221635-24). Katherine J. Lai is supported by a Graduate Research Fellowship from the National Science Foundation.

References

1. Alon, N., Moshkovitz, D., Safra, S.: Algorithmic construction of sets for k-restrictions. ACM Trans. Algorithms 2, 153–177 (2006)
2. Conrad, J., Gomes, C.P., van Hoeve, W.-J., Sabharwal, A., Suter, J.: Connections in networks: Hardness of feasibility versus optimality. In: Van Hentenryck, P., Wolsey, L.A. (eds.) CPAIOR 2007. LNCS, vol. 4510, pp. 16–28. Springer, Heidelberg (2007)
3. Crooks, K.R., Sanjayan, M. (eds.): Connectivity Conservation. Cambridge University Press, Cambridge (2006)
4. Cushman, S.A., McKelvey, K.S., Schwartz, M.K.: Use of empirically derived source-destination models to map regional conservation corridors.. Conservation Biology 23(2), 368–376 (2009)
5. Dilkina, B., Gomes, C.P.: Solving connected subgraph problems in wildlife conservation. In: Lodi, A., Milano, M., Toth, P. (eds.) CPAIOR 2010. LNCS, vol. 6140, pp. 102–116. Springer, Heidelberg (2010)
6. Feige, U., Mirrokni, V.S.: Maximizing non-monotone submodular functions. In: FOCS, pp. 461–471 (2007)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York (1979)
8. Gomes, C.P.: Computational Sustainability: Computational methods for a sustainable environment, economy, and society. The Bridge, National Academy of Engineering 39(4) (Winter 2009)
9. Gomes, C.P., van Hoeve, W.-J., Sabharwal, A.: Connections in networks: A hybrid approach. In: Trick, M.A. (ed.) CPAIOR 2008. LNCS, vol. 5015, pp. 303–307. Springer, Heidelberg (2008)
10. Joseph, L.N., Maloney, R.F., Possingham, H.P.: Optimal allocation of resources among threatened species: a project prioritization protocol. Conservation Biology 23(2), 328–338 (2009)
11. Krumke, S.: Improving Minimum Cost Spanning Trees by Upgrading Nodes. Journal of Algorithms 33(1), 92–111 (1999)

12. Lin, H., Bilmes, J.: Multi-document summarization via budgeted maximization of submodular functions. In: Human Language Technology Conference, NAACL/HLT (2010)
13. Naidoo, R., Balmford, A., Ferraro, P.J., Polasky, S., Ricketts, T.H., Rouget, M.: Integrating economic costs into conservation planning. Trends in Ecology & Evolution 21(12), 681–687 (2006)
14. Nemhauser, G.L., Wolsey, L.A., Fisher, M.L.: An analysis of approximations for maximizing submodular set functions-I. Mathematical Programming 14(1), 265–294 (1978)
15. Paik, D., Sahni, S.: Network upgrading problems. Networks 26(1), 45–58 (1995)
16. Singleton, P.H., Gaines, W.L., Lehmkuhl, J.F.: Landscape permeability for large carnivores in washington: a geographic information system weighted-distance and least-cost corridor assessment. Res. Pap. PNW-RP-549: U.S. Dept. of Agric., Forest Service, Pacific Northwest Research Station (2002)
17. Taylor, P.D., Fahrig, L., Henein, K., Merriam, G.: Connectivity is a vital element of landscape structure. Oikos 73, 43–48 (1993)