Optimizing Resilience in Large Scale Networks

Xiaojian Wu[1] **Daniel Sheldon**[1,2] **Shlomo Zilberstein**[1]

[1] College of Information and Computer Sciences, University of Massachusetts, Amherst, MA 01003, USA
[2] Department of Computer Science, Mount Holyoke College, South Hadley, MA 01075, USA
{xiaojian,sheldon,shlomo}@cs.umass.edu

Abstract

We propose a decision making framework to optimize the resilience of road networks to natural disasters such as floods. Our model generalizes an existing one for this problem by allowing roads with a broad class of stochastic delay models. We then present a fast algorithm based on the sample average approximation (SAA) method and network design techniques to solve this problem approximately. On a small existing benchmark, our algorithm produces near-optimal solutions and the SAA method converges quickly with a small number of samples. We then apply our algorithm to a large real-world problem to optimize the resilience of a road network to failures of stream crossing structures to minimize travel times of emergency medical service vehicles. On medium-sized networks, our algorithm obtains solutions of comparable quality to a greedy baseline method but is 30–60 times faster. Our algorithm is the only existing algorithm that can scale to the full network, which has many thousands of edges.

Introduction

We study the problem of optimizing the resilience of networks to various types of failures. Our approach applies to both digital networks (e.g., communication networks) and physical networks (e.g., road networks). In this paper, we focus on the latter, and describe a method to help decision makers use a limited budget to reinforce roads and make them more resistant to damage from floods. According to a recent report (Hallegatte et al. 2013), flood damages for the 136 largest coastal cities around the globe could cost $1 trillion per year by 2050 without taking protective measures. Developing computational tools to help combat this problem is therefore an important task.

This challenge has been tackled in the past, most notably as the Pre-disaster Transportation Network Preparation (PTNP) problem (Peeta et al. 2010; Schichl and Sellmann 2015). In this problem, an undirected graph is given where edges represent road segments and vertices represent intersections of multiple roads. Each edge has a fixed travel time (or length) and can survive the disaster with a certain probability. If a road segment fails, it becomes impassable. Pre-disaster investment can increase the survival probability of a road segment. Given a budget limit, the goal is to select a subset of edges to invest in so that the total expected travel time between a set of origin/destination (o/d) pairs is minimized. Optimization algorithms for PTNP have been developed and applied to practical problems.

Previous works, however, suffer from two significant limitations. First, the assumption that an edge has only two states—passable or impassable—is limiting. For example, a malfunctioning culvert may not disconnect the road completely, but can cause traffic to experience a certain amount of delay. Hence, edge lengths can take on multiple different values. Second, although existing algorithms can produce optimal solutions, they can only scale up to networks of 1000 edges and 5 o/d pairs (Schichl and Sellmann 2015). In our application, the network contains 55687 edges and 5504 pairs. Existing algorithms do not scale to networks of this size and cannot handle edges with multiple possible lengths.

To address the above limitations, we first formulate a more general problem called Generalized Predisaster Transportation Network Preparation (GPTNP). Unlike PTNP, the problem is defined on *directed* graphs, which are a better model for road networks. The length of an edge is a random variable with domain $[0, \infty]$ where ∞ means that the edge is impassable. The investment on an edge makes its length *stochastically shorter*. This general definition allows us to model richer and more practical situations.

We then create a very fast algorithm, based on the sample average approximation (SAA) (Sheldon et al. 2010; Wu, Sheldon, and Zilberstein 2015) and network design algorithms, to compute a high quality solution for the GPTNP problem in large-scale networks. In our algorithm, we recast the GPTNP problem as a deterministic optimization problem using the SAA method and a novel sampling procedure. With enough samples, the optimal solution to the deterministic problem approaches the optimal solution of the GPTNP problem (Kleywegt, Shapiro, and Homem-de Mello 2002). Then, the deterministic problem is formulated as a novel network design problem called Budget Set Weighted Shortest Path Steiner Graph (BSW-SPSG), composed of a directed graph and a budget limit. The goal is to purchase sets of edges to minimize the total shortest path length between a number of o/d pairs, subject to a budget constraint.

The BSW-SPSG problem is NP-hard and, more importantly, the size of the constructed problem is usually very large. Consequently, even a greedy heuristic is too slow

to tackle our target application. Hence, we propose a fast primal-dual algorithm to solve the BSW-SPSG problem approximately. The key idea is to first use Lagrangian relaxation (Jain and Vazirani 2001; Kumar, Wu, and Zilberstein 2012) to fold the budget constraint into the objective, parameterized by a Lagrange multiplier β. The resulting problem is called Prize-Collecting Set Weighted Shortest Path Steiner Graph (PCSW-SPSG). A bisection procedure (Wu, Sheldon, and Zilberstein 2015) is then used to find a value of β for which the near optimal solution of PCSW-SPSG is also a near optimal solution of the BSW-SPSG problem. Finally, we derive a primal-dual algorithm (Williamson and Shmoys 2011) to solve the PCSW-SPSG problem approximately.

We demonstrate experimentally on a small existing benchmark that our our algorithm produces near optimal solutions to the BSW-SPSG problem and the SAA method converges quickly to the optimal solution of the original stochastic optimization problem. On a network of medium size, our algorithm performs as well as a greedy baseline, but it is 30–60 times faster. When applied to a full real-world road network, our algorithm is the only one that finds good solutions within a reasonable amount of time. We also show a BSW-SPSG problem where the greedy baseline performs poorly, but our algorithm finds near-optimal solutions.

Another contribution of the paper is that our algorithm, with minor modifications, can solve two additional variants of the BSW-SPSG problem: the Quota Set Weighted Shortest Path Steiner Graph (QST-SPSG) problem and the PCSW-SPSG problem. The prefixes *quota*, *budget*, and *prize-collecting* describe standard variants of network design problems (Johnson, Minkoff, and Phillips 2000). The problems share the same basic setup, but have different constraints and objectives. In the QST-SPSG problem, the goal is to minimize cost while keeping the total shortest path length below a certain level. In the PCSW-SPSG problem, both total shortest path length and cost are in the objective, weighted by a tradeoff parameter β.

We begin with a formal description of the GPTNP problem in Section 2. Section 3 introduces the SAA method and our sampling procedure. Section 4 defines the BSW-SPSG problem. Section 5 presents our solution method. Section 6 contains the experimental results and analysis.

Problem Statement

An instance of GPTNP includes a directed graph $G{=}(V, E)$, where each edge e is associated with a *travel time* or *length* that is stochastically distributed within $[0, \infty]$ (∞ indicates the edge is not passable). The length can be decreased stochastically at some cost c_e. We are also given a set of pairs of vertices $\Theta = \{(o_1, d_1), ..., (o_T, d_T)\}$ $(o_i, d_i \in V)$ and a budget limit $\mathcal{B}$. The goal is to select a subset $\pi \subseteq E$ of edges, also called a *policy*, to invest subject to the budget limit $\mathcal{B}$, so that the total expected shortest path length from o to d for all $(o, d) \in \Theta$ is minimized.

The length of edge e is a random variable $L_e(\pi)$ whose distribution depends on π. We are given two cumulative distribution functions (CDF): $H_e : [0, \infty] \rightarrow [0, 1]$ and $F_e : [0, \infty] \rightarrow [0, 1]$ for the cases ($e \notin \pi$) and ($e \in \pi$) respectively. With investment, the length is *stochastically shorter* than the length without. This is modeled by the constraints $F_e(l) \geq H_e(l)$ for any $l \in [0, \infty]$. Let $L(\pi)$ denote the random vector consisting of $L_e(\pi)$ for all $e \in E$. The GPTNP problem can be written as a stochastic optimization problem:

$$\min_{\pi \subseteq E} \sum_{(o,d)\in\Theta} \mathbb{E}_{L(\pi)} \left[SPL(o, d, L(\pi); G) \right] \; s.t. \sum_{e\in\pi} c_e \leq \mathcal{B} \quad (1)$$

where $SPL(o, d, L(\pi); G)$ represents the shortest path length from o to d when edge lengths are $L(\pi)$. We also define a penalty $M_{o,d}$ for a pair (o, d). If the shortest path length is ∞, we set $SPL(o, d, L(\pi); G) = M_{o,d}$.

In the above definition, we assume that the length of each edge is independently distributed, but our solution method can be modiflied slightly to deal with more complex settings in which lengths of multiple edges are correlated or one investment can affect multiple edges simultaneously. To extend our algorithm to these cases, only the sampling procedure needs to be modified; the bisection procedure and the primal-dual algorithm remain the same.

Connection to the PTNP Problem The major difference between the PTNP (Peeta et al. 2010) and GPTNP problems is that in PTNP, the length of each edge is defined by a binary random variable, that is, with probability p_e, the edge has length l_e and with probability $1 - p_e$, the edge is impassable or has length ∞. Another difference is that PTNP is defined on an undirected graph while GPTNP is defined on a directed graph. In fact, our model is more general because an undirected edge is equivalent to two directed edges of equal length that share the same investment. As GPTNP is a more general problem, two complexity results established for the PTNP problem (Schichl and Sellmann 2015) immediately apply to the GPTNP problem.

Theorem 1. *The GPTNP problem is APX-hard and is neither sub- nor super-modular.*

We note that the GPTNP problem is also related to the continuous-time influence maximization problem (Rodriguez and Schölkopf 2012; Du et al. 2013). The time of a vertex being infected can be considered as the shortest path length from the source to this vertex in our problem. Our problem suggests novel variants of influence maximization where it is possible to accelerate propagation by manipulating certain edges in addition to selecting diffusion sources (Khalil, Dilkina, and Song 2014).

Sample Average Approximation

We use the SAA method to recast the stochastic optimization problem (1) as a deterministic analogue using N scenarios that are generated by sampling from the underlying distribution. It has been shown (Kleywegt, Shapiro, and Homem-de Mello 2002) that as N goes to infinity, the optimal solution of the deterministic problem converges to the optimal solution of the stochastic optimization problem. However, we can't directly sample $L(\pi)$ as its distribution depends on π.

Here, we introduce a new sampling method. The basic idea is that we define a new random graph $\boldsymbol{G'} = (V, E', \boldsymbol{\xi})$ where $\boldsymbol{\xi} = \{\boldsymbol{\xi}_e | e \in E'\}$ and $\boldsymbol{\xi}_e$ is a random variable representing the length of edge $e \in E'$. The distribution of $\boldsymbol{\xi}_e$

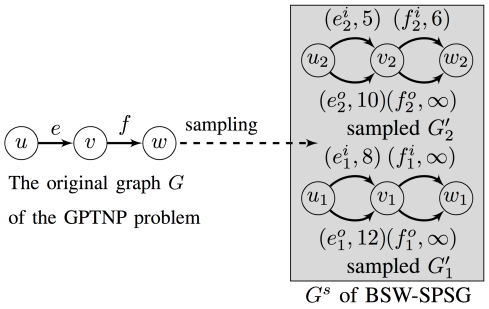


Figure 1: An example of creating the BSW-SPSG problem. The tuple $(e_2^i, 5)$ means the edge e_2^i has sampled length 5. The graph G^s contains all vertices and edges in gray. We have edge sets $E_e = \{e_1^i, e_2^i\}$, $E_f = \{f_2^i\}$ and $E_0 = \{e_1^o, e_2^o\}$ with $c_0 = 0$ in the BSW-SPSG problem. If the o/d pair in the GPTNP is $\{u, w\}$, in BSW-SPSG, $\Theta = \{(u_1, w_1), (u_2, w_2)\}$

does not depend on any policy, so the sample of $\boldsymbol{G'}$ can be drawn before applying policies. Then, we create the deterministic optimization problem using N samples of $\boldsymbol{G'}$.

More specifically, in $\boldsymbol{G'}$, there are two parallel edges e^o (*original* edge) and e^i (*invested* edge) in E' for each edge $e \in E$, with lengths $\boldsymbol{\xi}_{e^o}$ and $\boldsymbol{\xi}_{e^i}$ respectively. We further define a random graph $\boldsymbol{G'}(\pi) = \{V, E'(\pi), \boldsymbol{\xi}\}$ parameterized by policy π. This graph always includes the original edge (length ξ_{e^o}) and will include the invested edge (length ξ_{e^i}) if and only if $e = (u, v) \in \pi$. If $e \notin \pi$, the distance from u to v is ξ_{e^o}; we want this to have CDF H_e. If $e \in \pi$, the distance from u to v is $\min\{\xi_{e^o}, \xi_{e^i}\}$; we want this to have CDF F_e. To achieve this, for each edge $e \in E$, we define a uniform random variable U_e in the range $[0, 1]$ and use it to define:

$$\boldsymbol{\xi}_{e^o} = \min_{l: H_e(l) = U_e} l, \qquad \boldsymbol{\xi}_{e^i} = \min_{l: F_e(l) = U_e} l \tag{2}$$

With this definition, we can sample the values of $\boldsymbol{\xi}_{e^o}$ and $\boldsymbol{\xi}_{e^i}$ independently of any policy by inverse-transform sampling (Devroye 1986). First, a value of U_e is sampled to be the cumulative probability. Then, we pick the smallest lengths that have cumulative probability U_e from H_e and F_e respectively. If the function H_e or F_e is strictly increasing, we have $\boldsymbol{\xi}_{e^o} = H_e^{-1}(U_e)$ or $\boldsymbol{\xi}_{e^i} = F_e^{-1}(U_e)$. Also, if H_e or F_e is a discrete CDF, the probability of any sampled value is nonzero. Now, we claim the following result.

Theorem 2. *For any fixed policy $\pi \subseteq E$ and two vertices $o, d \in V$, the expected shortest path length from o to d in $G'(\pi)$ equals to $\mathbb{E}_{L(\pi)}\left[SPL(o, d, L(\pi); G)\right]$ in (1).*

To summarize our SAA procedure: first, we generate N samples of $\boldsymbol{G'}$, $\{G_1', ..., G_N'\}$, by sampling the value of U_e's and applying (2). In G_k', edges have fixed lengths. Then, we form the following deterministic optimization problem:

$$\min_{\pi \subseteq E} \frac{1}{N} \sum_{(o,d) \in \Theta} \sum_{k=1}^{N} SPL(o, d, G_k'(\pi)) \; s.t. \sum_{e \in \pi} c_e \leq \mathcal{B} \tag{3}$$

$$\min \sum_{(o,d) \in \Theta^s} \Big(\sum_{e \in E^s} d_e x_e^{od} \Big) + M_{o,d} z^{od} \tag{1}$$

$$s.t. \sum_{e \in \delta^+(S)} x_e^{od} + z^{od} \geq 1 \quad \forall (o,d) \in \Theta^s, S \in \mathcal{S}_o^{od} \tag{2}$$

$$\sum_{i: e \in E_i} y_i \geq x_e^{od} \quad \forall (o,d) \in \Theta^s, \forall e \in E^s \tag{3}$$

$$\sum_{E_i \in \mathcal{E}} c_i y_i \leq \mathcal{B} \tag{4}$$

$$x_e^{od} \in [0,1] \qquad \forall (o,d) \in \Theta^s \;\; \forall e \in E^s \tag{5}$$

$$y_i \in \{0,1\} \; \forall E_i \in \mathcal{E}, \;\; z^{od} \in \{0,1\} \; \forall (o,d) \in \Theta^s \tag{6}$$

Figure 2: MIP of the budget BSW-SPSG problem

where $SPL(o, d, G_k'(\pi))$ is the shortest path length from o to d in the sample $G_k'(\pi)$. By theorem 2, for any policy, the objective of (3) converges to the objective of (1) as N goes to infinity. In the next two sections, we formally define the problem (3) and give a fast algorithm for solving it.

Set Weighted Shortest Path Steiner Graph

Problem (3) can be formulated as a new network design problem called Budget Set Weighted Shortest Path Steiner Graph (BSW-SPSG). The input of a BSW-SPSG problem consists of a directed graph $G^s = \{V^s, E^s\}$ where each edge has fixed length l_e and a set of o/d pairs $\Theta^s = \{(o_1, d_1), ..., (o_T, d_T)\}$ $(o_i, d_i \in V^s)$ where each pair is associated with a penalty $M_{o,d}$. We are also given a collection of edge sets $\mathcal{E} = \{E_1, ..., E_S\}$ where $E_i \subseteq E^s$ and each E_i is associated with a cost c_i. A subset $\mathcal{A} \subseteq \mathcal{E}$ corresponds to a subgraph $G^s(\mathcal{A}) = \{V^s, E^s(\mathcal{A})\}$ with $E^s(\mathcal{A}) = \cup_{E_i \in \mathcal{A}} E_i$. The goal is to purchase edge sets $\mathcal{A}$, subject to a budget limit $\mathcal{B}$, such that in $G^s(\mathcal{A})$, the total shortest path length from o to d for all $(o, d) \in \Theta^s$ is minimized. That is,

$$\min_{\mathcal{A} \subseteq \mathcal{E}} \sum_{(o,d) \in \Theta^s} SPL(o, d, G^s(\mathcal{A})) \;\; s.t. \sum_{E_i \in \mathcal{E}} c_i \leq \mathcal{B} \tag{4}$$

where $SPL(o, d, G^s(\mathcal{A}))$ is the shortest length path from o to d in $G^s(\mathcal{A})$, set to be $M_{o,d}$ if there is no path from o to d.

To write problem (3) as a BSW-SPSG problem, we combine N sampled graphs $\{G_1', ..., G_N'\}$ of $\boldsymbol{G'}$ into a single graph G^s with appropriate edge sets $\mathcal{E}$. The vertex set of G^s is the union of those from the samples $\{G_i'\}$. The edges of G^s include original edges $\{e_k^o\}$ and invested edges $\{e_k^i\}$ of all samples. For each edge e in G, the invested edges of finite length (from all samples) are grouped into an edge set E_e with cost c_e. The unimproved edges of finite length for *all* edges in G are grouped into an additional edge set E_0 of zero cost. Finally, let set Θ^s contain o/d pairs in all sampled graphs. An example is shown in Fig. 1.

The BSW-SPSG problem can be formulated as a Mixed Integer Program (MIP) shown in Fig. 2. A binary variable x_e^{od} is defined for each edge $e \in E$ and o/d pair indicating whether e is on (=1) the shortest path from o to d or not

Prize-Collecting Problem / Primal Problem:

$$L(\beta) = \min \left(\sum_{(o,d)\in\Theta^s} (\sum_{e\in E} d_e x_e^{od}) + M_{o,d} z^{od} \right) + \beta \left(\sum_{E_i \in \mathcal{E}} c_i y_i - \mathcal{B} \right) \quad (1)$$

$$s.t.\ \ x, y \text{ satisfy constraints (2), (3), (5), (6) in Fig. 2} \quad (2)$$

Dual Problem:

$$\max \sum_{(o,d)\in\Theta^s, S\in\mathcal{S}_o^{od}} \mu_S^{od} \quad (3)$$

$$s.t. \sum_{S: e\in\delta^+(S), S\in\mathcal{S}_s^{od}} \mu_S^{od} - \lambda_e^{od} \le d_e \quad \forall (o,d)\in\Theta^s, \forall e\in E \quad (4)$$

$$\sum_{S: S\in\mathcal{S}_s^{od}} \mu_S^{od} \le M_{o,d} \quad \forall (o,d)\in\Theta^s \quad (5)$$

$$\sum_{(o,d)\in\Theta^s} \sum_{e\in E_i} \lambda_e^{od} \le \beta c_i \quad \forall E_i \in \mathcal{E} \quad (6)$$

Figure 3: Primal and dual of the prize-collecting problem

(=0). A binary decision variable y_i is defined for each edge set $E_i \in \mathcal{E}$ indicating whether the set is purchased (=1) or not (=0). All y_is define a set $\mathcal{A}$. A binary variable z^{od} is defined for each o/d pair indicating whether the pair is penalized/disconnected (=1) or not/connected (=0). Let set $\mathcal{S}_o^{od}$ consist of all subsets of V that contain o but not d, $\mathcal{S}_o^{od} = \{S | S \subseteq V \wedge o \in S \wedge d \notin S\}$. Let set $\delta^+(S)$ contain all outgoing cut edges of S, that is, $\delta^+(S) = \{(u,v) | (u,v) \in E \wedge u \in S \wedge v \notin S\}$. With constraint (2), if d is connected to o ($z^{od} = 0$), at least one cut edge e for any set in $\mathcal{S}_o^{od}$ should have value $x_e^{od} = 1$. With constraint (3), an edge is on the shortest path from o to d only if it is purchased. Constraint (4) is the budget constraint. It is easy to show that relaxing binary variable x into continuous variable in $[0, 1]$ will not change the value of the optimal solutions.

Proposition 1. *The BSW-SPSG problem is NP-hard and its objective function is neither sub- nor super-modular.*

Our Algorithm for the BSW-SPSG problem

In this section, we present the algorithms to solve both BSW-SPSG and the prize-collecting problem (PCSW-SPSG).

To solve BSW-SPSG problem, we first create a PCSW-SPSG problem by folding the budget constraint into the objective with a β based on Lagrangian relaxation method (Kumar, Wu, and Zilberstein 2012; Jain and Vazirani 2001). The MIP of PCSW-SPSG is shown in Fig. 3. Then, we use the **bisection procedure** to find a β such that by solving the prize-collecting problem with β, we obtain a high-quality solution of the budget problem. In the bisection procedure, we start with a large interval of β and halve it iteratively until narrow enough. At each iteration, a solution is found by solving the prize-collecting problem with β. If the cost of the solution is greater than $\mathcal{B}$, the lower half of the interval is abandoned. Otherwise, the higher half is abandoned. The details of this method are in (Wu, Sheldon, and Zilberstein 2015).

Algorithm 1 Primal-Dual Algorithm

```
1: function PRIMALDUAL(G^s, E, β)
2:     Set y_i ← 0 ∀E_i ∈ E and z^od ← 0 ∀(o,d) ∈ Θ^s
3:     F ← φ
4:     For each pair (o,d), the set C^od contains all vertices reachable from o in (V^s, F)
5:     Initially, C^od = {o} for each (o,d)
6:     Mark all o/d pairs as active
7:     while there exists some active pairs do
8:         Increase μ^od_{C^od} of all active pairs simultaneously. If (4) in Fig. 3 becomes tight for edge e, increase λ^od_e to maintain feasibility until
9:         if (5) in Fig. 3 becomes tight for (o,d) then,
10:            Mark (o,d) abandoned and set z^od = 1
11:        else                          ▷ (6) in Fig. 3 becomes tight for E_i
12:            set y_i = 1 and F ← F ∪ e
13:            for each active (o,d) do
14:                expand C^od by e
15:            end for
16:            if C^od contains d for some (o,d) then
17:                Mark (o,d) connected
18:            end if
19:        end if
20:    end while
21:    For each connected (o,d), find the shortest path in (V^s, F)
22:    for each E_i with y_i = 1 do
23:        if there are no shortest paths using edges in E_i then
24:            y_i ← 0
25:        end if
26:    end for
27:    return y
28: end function
```

Solving the Prize-Collecting Problem

To scale up to large networks, we derive a fast primal-dual algorithm, shown in Algorithm 1, to solve the PCSW-SPSG problem approximately. Fig. 3 gives the dual formulation of the LP relaxation of the prize-collecting problem.

The algorithm borrows ideas from the primal-dual algorithms of the single pair shortest path problem and the generalized Steiner tree problem (Williamson and Shmoys 2011). The basic idea is to repeatedly increase the dual objective by increasing the value of dual variables and simultaneously construct a feasible primal solution based on the primal complementary slackness condition (Vazirani 2003).

Definition 1. ***Primal Complementary Slackness Condition*** *$x_e^{od} = 1$, $z^{od} = 1$ and $y_i = 1$ imply equality in constaints (4), (5) and (6) in Fig. 3 respectively.*

Lines 2-20 of Algorithm 1 are the major primal-dual procedure. We start with an infeasible primal solution where all x, y, z variables are 0, an empty graph (V^s, F) where $F = \phi$ and a feasible dual solution where all μ and λ are 0. The graph (V^s, F) represents the primal solution: F contains e if $x_e^{od} = 1$ for any o/d pair. The loop (lines 7-20) proceeds until each (o,d) is either abandoned ($z^{od} = 1$) or connected

where we get a feasible primal solution by satisfying (2) in Fig. 2. Note that (3) in Fig. 2 is always satisfied as we proceed. A pair is *active* meaning that constraint (2) of this pair is violated for some S. To satisfy (2) for all $S \in \mathcal{S}_o^{od}$, at each iteration, we only increase the dual variable of the set $S \in \mathcal{S}_o^{od}$ with the smallest number of vertices for which the constraint (2) is violated. This smallest set is C^{od} defined at line 4. At each iteration, we increase the dual variable of C^{od} for all (o, d) simultaneously (line 8). If the constraint (4) in Fig. 3 becomes tight for some e, by the slackness condition, we want to add e in to F (set $x_e^{od} = 1$), but e may not be purchased yet. So, we continue increasing both $\mu_{C^{od}}^{od}$ and λ_e^{st} with the same speed until following cases happen. If (5) in Fig. 3 becomes tight for some (o, d), by the slackness condition, we set $z^{od} = 1$ and never consider this pair again because all constraints (2) in Fig. 2 that involve this pair are satisfied (line 10). If constraint (6) in Fig. 3 becomes tight for some E_i, we set $y_i = 1$. Also, we know e is purchased and can be added into F (line 12). At line 14, we update C^{od} with the newly added edge. At line 16, $d \in C^{od}$ means the pair is connected. In lines 21-30, the unused edge sets are removed. Only the edge sets that are used by the shortest path of *connected* pairs are purchased.

Proposition 2. *The running time of Algorithm 1 is bounded by* $O\big(\ (|E^s| + |\mathcal{E}| + |\Theta^s|)^2 + |\Theta^s|\ |E^s|\ \log|V^s|\ \big)$.

To analyze the running time of solving GPTNP by reducing it to BSW-SPSG, let N be number of samples, and K be the number of iterations in the bisection procedure. Then, in the constructed BSW-SPSG problem, there are at most $2N{\cdot}|E|$ edges (i.e., $|E^s| \leq 2N{\cdot}|E|$), $|\Theta^s| = N{\cdot}|\Theta|$ *o/d* pairs, and $|\mathcal{E}| \leq |E|$. As the primal-dual algorithm is invoked K times by the bisection procedure, by plugging the above terms into the bound in Proposition 2, we have:

Proposition 3. *The running time of solving a GPTNP problem is* $O\big(K\ N^2\ (|E| + |\Theta|)^2 + K\ N\ |\Theta|\ |E|\ \log\ |V|\ \big)$.

To analyze the complexity of the natural greedy algorithm, let K' be the number of edges selected for investment by the greedy algorithm. Then, we have

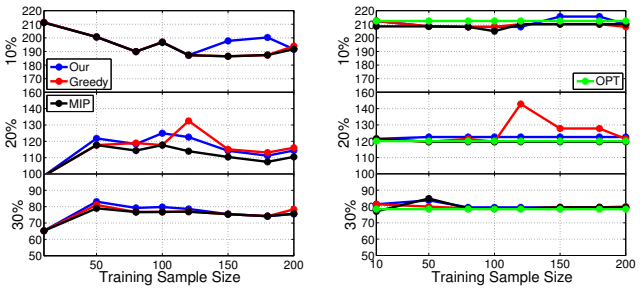
Proposition 4. *The runtime of a greedy algorithm is bounded by* $O\big(\ K'\ N\ |\mathcal{E}|\ |\Theta|\ |E|\ \log|V|\ \big)$.

In the experiments, we observed that a small number of samples (e.g., $N \leq 30$) is sufficient for convergence. If we assume that N is a constant, $|E| \geq |\Theta|$, and $|\mathcal{E}| = \Theta(|E|)$, the running time for our algorithm is $O(K\ |E|^2\ \log|V|)$, and the running time for the greedy algorithm is $O\big(K'\ |\Theta|\ |E|^2\ \log\ |V|\big)$, implying that our algorithm is asymptotically $O(\frac{K'\,|\Theta|}{K})$ times faster than the greedy algorithm. On our largest network with $|\Theta| = 5504$, a small value of K (e.g., a number between 20 and 30) is sufficient to make the range of β narrow enough in our algorithm, and in the greedy algorithm, K' is a number between 200 and 300. We observed that our algorithm is about 400 times faster than the greedy algorithm.

Experimental Results

Theoretically, the convergence of the SAA method is guaranteed as the number of samples N goes to infinity. In practice, a small number of samples may be enough for convergence. In our experiments, we treat our optimization algorithm as the training step and the samples used by SAA as training samples. We use training samples to create a BSW-SPSG problem and compare the performance of our algorithm in optimizing this BSW-SPSG problem against other existing algorithms. However, an optimal solution to the training samples may perform poorly in minimizing the expected travel time when N is not big enough. To evaluate the actual performance of policies produced in the training step and also determine how many training samples are enough for convergence, we conduct a testing step. In testing, we draw another group of samples as testing samples and calculate, for a policy, the average travel time as a testing value. The testing value of a policy is an unbiased estimate of its expected travel time. As N increases, the convergence of the testing value is a good indicator that SAA converges.

We experimented on two different domains, using a 2.2GHz Intel Core i7 CPU with 16GB of RAM.



(a) Training (b) Testing

Figure 4: "Istanbul Prep" benchmark. Y-axis: path lengths.

Istanbul Earthquake Preparation

We first used a small benchmark called Istanbul Earthquake Preparation (Peeta et al. 2010). The goal is to minimize the total shortest path length between 5 o/d pairs. The network contains 30 undirected edges. Each edge has binary length distribution. The survival probability can be raised to 1.0 with investment. We used the basic settings described by Peeta et al. (2010), with $M_{o,d} = 120$. On this small problem, we only report solution quality. In Fig. 4, we compare our algorithm, the greedy algorithm and the mixed integer program (MIP) with three budget sizes (10%, 20% and 30%) where MIP is an optimal solver. In testing, we use 10000 samples to evaluate policies produced in the training step. We also add the optimal expected values (green) as comparisons, which are reported by (Schichl and Sellmann 2015).

In training, "Our" and "Greedy" produce near optimal solutions in most of the cases. In testing, except for several cases for "Greedy", all algorithms produce near-optimal testing values. For the 10% budget case, the testing value is better than the optimal expected value; this indicates a slight discrepancy in problem setting from previous work. The training value of each algorithm is always smaller than the testing value due to overfitting. Also, varying the sample

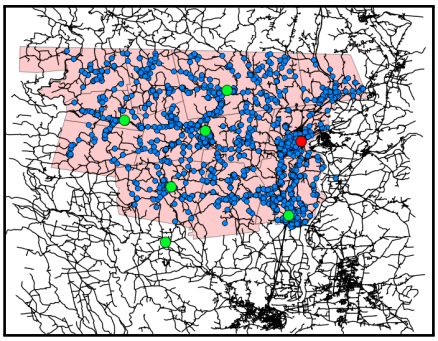
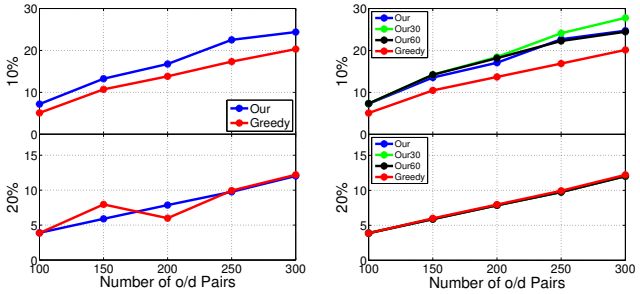


Figure 5: The road network showing patient locations (blue), hospital (red), and ambulance dispatch locations (green).



(a) Training (b) Testing

Figure 6: Performance on the sub-network for flood preparation. Y-axis represents total travel time ($\times 10^4$ seconds).

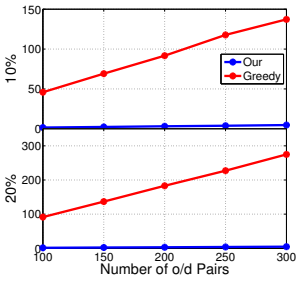


Figure 7: Training time (mins) on the sub-network.

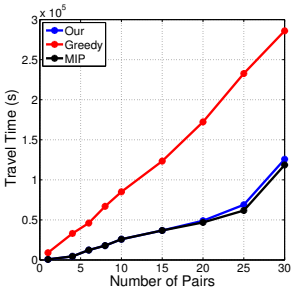


Figure 8: Network with survival probabilities 0.

size from 10 to 200 barely impacts the testing value, which implies that 10 training samples are enough for convergence.

Flood Preparation for EMS

Roadway stream crossing structures, such as culverts, become vulnerable to floods as climate changes. The failure of crossing structures causes road segments to flood or wash out, which causes traffic delay or even make roads impassable. Money can be invested to increase the resilience of crossings. The goal is to decide which crossings to invest in, subject to a budget limit, so that the total travel time of certain o/d pairs is minimized. In this problem, we focus only on travel time of emergency medical services (EMS). We obtained relevant data for the road network of the Deerfield river watershed in Massachusetts. In our dataset, an o/d pair represents a request of ambulance from o (ambulance center/patient address) to d (patient address/hospital). We obtain such pairs from actual EMS calls recorded over the past 5 years. The road network, shown in Fig. 5, contains 55687 edges, 1366 crossings (not shown) and 5504 o/d pairs.

To conduct the experiments, we used the following assumptions. The length of an edge corresponds to travel time and is calculated by $l_e = \frac{road\ length}{speed\ limit}$. Each crossing has a survival probability p_e in the range $[0.2 - 0.4]$. An edge, if associated with a crossing, has length l_e with probability p_e and has length ∞ with probability $1 - p_e$. p_e is raised to 1.0 if the corresponding crossing is fixed. We used a constant investment cost for all crossings. The penalty $M_{o,d}$ of each disconnected pair was set to be 15 times the shortest travel time from o to d when no crossings fail.

Sub-Network We compare SAA and the greedy baseline on a sub-network with 10037 edges and 248 crossings. MIP failed to finish within a reasonable amount of time for this dataset. The results are shown in Fig. 6. In training, we use 10 samples, two budget sizes and various numbers of o/d pairs. For 10% budget, "Greedy" performs slightly better, but the value of our algorithm is within 1.3 times the value of "Greedy". For 20% budget, both algorithms perform the same in most cases. In terms of runtime, shown in Fig. 7, our algorithm is significantly faster. In testing, we use 100 testing samples and evaluate the policies produced by different number of training samples. For example, "Our30" represents the policy trained on 30 samples by our algorithm. The policy of "Greedy" is trained on 10 samples due to its limited scalability. Again, for 10% budget, the policy produced by "Greedy" gives slightly better testing value. For 20% budget, all policies basically perform the same, implying that 10 samples are enough for convergence. Overall, the results show that our algorithm performs similarly well or a little worse in some cases compared to the greedy algorithm, but it is significantly faster and can scale up to larger problems.

Complete Network We also tested on the complete road network using 10 training samples and 100 testing samples. In this case, it is not feasible to run the greedy algorithm; one iteration takes more than 10 hours. Instead, we compared with two other methods. The results are shown in Table 1. The "Random" method selects a policy by randomly picking crossings to fix until the budget is exhausted. We let the algorithm repeatedly generate and test policies over 10 hours and report the best one. The "M-Greedy" method is the same as the greedy algorithm except that, at each iteration, it only re-evaluates the top 10 crossings that gave the best travel time reduction in the previous iteration. We only show the result of "M-Greedy" for 10% budget, which already takes 46 hours. By the table, our algorithm runs faster and produces the best training and testing values. To the best of our knowledge, our algorithm is the only known method that can solve this problem and produce good solutions.

Experiments in More Challenging Settings As shown above, the greedy algorithm performs quite well in training

Algorithm	Budget	Train ($\times 10^6$s)	Time (h)	Test ($\times 10^6$s)
Our	10%	9.8	6.1	9.8
Random	10%	24.5	13	24.4
M-Greedy	10%	11.2	46.2	11.4
Our	20%	3.6	7.3	3.6
Random	20%	19.7	16.3	19.8

Table 1: Experiments on the complete road network of the Deerfield river watershed

in terms of solution quality. We believe one reason for this is that many o/d pairs are close to each other and the survival probabilities of crossings are relatively high, so, with high probability, at most one or two crossings fail on any o/d path. This means that coordinated repairs of multiple crossings are not necessary to realize improvements, and the greedy algorithm can achieve good performance by repairing crossings incrementally in a myopic way. Intuitively, the greedy strategy will fail when o/d paths experience multiple failures, so repairing any single crossing provides no benefit. To model this, we designed a BSW-SPSG problem where all survival probabilities start out equal to zero. We used a small problem for which it is possible to find optimal solutions by solving a MIP. The results in Fig. 8 show that the greedy algorithm performs poorly in this case, while our algorithm produces near optimal solutions.

Conclusion

We propose a general decision making framework and a fast approximate algorithm for optimizing the resilience of networks to various failures. Compared to previous work, our framework allows for richer problems to be represented, with arbitrary length distributions over edges. We construct the approximation method by deriving a sampling method and a primal-dual algorithm. Empirically, our algorithm produces near optimal solutions on a benchmark problem. It perform as well as a greedy baseline on a mid-size network, but is significantly faster. Most importantly, our algorithm scales well and can solve a larger practical problem that no other existing algorithm can solve. It is also robust to varying network conditions and produces a near optimal solution for a challenging problem on which the baseline greedy algorithm performs poorly. Finally, our work offers a good foundation for exploring several related problems, particularly the continuous time influence maximization problem that we plan to study in future work.

Acknowledgements

We thank Joshua Shanley for providing the EMS call records and Scott Jackson for providing the culvert data and helping with the modeling and interpretation of the data. This project was funded in part by the Massachusetts Department of Transportation through ISA 114-118531 for grant No. 83226.

References

Devroye, L. 1986. *Non-uniform random variate generation*. Springer.

Du, N.; Song, L.; Gomez-Rodriguez, M.; and Zha, H. 2013. Scalable influence estimation in continuous-time diffusion networks. In *Advances in Neural Information Processing Systems (NIPS)*, 3147–3155.

Hallegatte, S.; Green, C.; Nicholls, R. J.; and Corfee-Morlot, J. 2013. Future flood losses in major coastal cities. *Nature Climate Change* 3:802–806.

Jain, K., and Vazirani, V. V. 2001. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation. *Journal of the ACM (JACM)* 48(2):274–296.

Johnson, D. S.; Minkoff, M.; and Phillips, S. 2000. The prize collecting Steiner tree problem: Theory and practice. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, 760–769.

Khalil, E. B.; Dilkina, B.; and Song, L. 2014. Scalable diffusion-aware optimization of network topology. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 1226–1235. ACM.

Kleywegt, A. J.; Shapiro, A.; and Homem-de Mello, T. 2002. The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization* 12:479–502.

Kumar, A.; Wu, X.; and Zilberstein, S. 2012. Lagrangian relaxation techniques for scalable spatial conservation planning. In *Proceedings of the 26th Conference on Artificial Intelligence (AAAI)*, 309–315.

Peeta, S.; Sibel Salman, F.; Gunnec, D.; and Viswanath, K. 2010. Pre-disaster investment decisions for strengthening a highway network. *Computers and Operations Research* 37(10):1708–1719.

Rodriguez, M. G., and Schölkopf, B. 2012. Influence maximization in continuous time diffusion networks. In Langford, J., and Pineau, J., eds., *Proceedings of the 29th International Conference on Machine Learning (ICML)*, 313–320.

Schichl, H., and Sellmann, M. 2015. Predisaster preparation of transportation networks. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI)*, 709–715.

Sheldon, D.; Dilkina, B.; Elmachtoub, A.; Finseth, R.; Sabharwal, A.; Conrad, J.; Gomes, C.; Shmoys, D.; Allen, W.; Amundsen, O.; and Vaughan, W. 2010. Maximizing the spread of cascades using network design. In *Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence (UAI)*, 517–526.

Vazirani, V. 2003. *Approximation Algorithms*. Springer.

Williamson, D. P., and Shmoys, D. B. 2011. *The design of approximation algorithms*. Cambridge University Press.

Wu, X.; Sheldon, D.; and Zilberstein, S. 2015. Fast combinatorial algorithm for optimizing the spread of cascades. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, 2655–2661.

Appendix

Theorem 2. *For any fixed policy $\pi \subseteq E$ and two vertices $o, d \in V$, the expected shortest path length from o to d in $G'(\pi)$ equals to $\mathbb{E}_{L(\pi)}[SPL(o, d, L(\pi); G)]$ in (1).*

Proof. First, let's consider an arbitrary policy π and an edge $e \in E$. For a fixed length l, we claim that

$$Pr(L_e(\pi) \leq l) = \begin{cases} Pr(\boldsymbol{\xi}_{e^o} \leq l) & \text{if } e \notin \pi \\ Pr(\min\{\boldsymbol{\xi}_{e^o}, \boldsymbol{\xi}_{e^i}\} \leq l) & \text{if } e \in \pi \end{cases} \tag{5}$$

Remember that we defined

$$\boldsymbol{\xi}_{e^o} = \min_{l:H_e(l)=U_e} l, \qquad \boldsymbol{\xi}_{e^i} = \min_{l:F_e(l)=U_e} l$$

and both H_e and F_e are nondecreasing functions.

- Case 1: $e \notin \pi$. Only edge e^o exists in $\boldsymbol{G}'(\pi)$. In G, $L_e(\pi)$ has CDF H_e. Then we have

$$\begin{aligned} Pr(\boldsymbol{\xi}_{e^o} \leq l) &= Pr(\min_{l':H_e(l')=U_e} l' \leq l) \\ &= Pr(U_e \leq H_e(l)) = H_e(l) \end{aligned}$$

  To explain the second last equation, let $l'' = \min_{l':H_e(l')=U_e} l'$. Since $l'' \leq l$, we have $H_e(l'') = U_e \leq H_e(l)$.
- Case 2: $e \in \pi$, both e^o and e^i are present in $\boldsymbol{G}'(\pi)$. In G, $L_e(\pi)$ has CDF F_e.
  Remember that we define the concept of *stochastically shorter* as $F_e(l') \geq H_e(l')$ for any l'. So, we claim

$$\min_{l':H_e(l')=U_e} l' \geq \min_{l':F_e(l')=U_e} l'$$

  To prove this claim, we let $l_1 = \min_{l':H_e(l')=U_e} l'$ and $l_2 = \min_{l':F_e(l')=U_e} l'$. We have $H_e(l_1) = F_e(l_2) = U_e$. If $l_1 < l_2$, since l_2 is the smallest length with $F_e(l_2) = U_e$, we have $H_e(l_1) \leq F_e(l_1) < U_e$ which contradicts $H_e(l_1) = U_e$. Therefore, $l_1 \geq l_2$ and the claim holds.
  Then, we have

$$\begin{aligned} &Pr(\min\{\boldsymbol{\xi}_{e^o}, \boldsymbol{\xi}_{e^i}\} \leq l) \\ =&Pr(\min\{\min_{l':H_e(l')=U_e} l', \min_{l':F_e(l')=U_e} l'\} \leq l) \\ =&Pr(\min_{l':F_e(l')=U_e} l' \leq l\}) = Pr(U_e \leq F_e(l)) = F_e(l) \end{aligned}$$

Now, we prove the theorem. First, we prove the easy case where there are no parallel edges in G and we only consider a single path $p = \{v_1, v_2, v_3, ..., v_n\}$ in G. In $\boldsymbol{G}'(\pi)$, we consider edges $(v_i, v_{i+1})^o$, and also $(v_i, v_{i+1})^i$ only if $(v_i, v_{i+1}) \in \pi$ for any i. We ignore all other edges in both graphs for now. Then, for any shortest path length l in G, the probability that l is achieved by G with policy π is the same as the probability that l is achieved by $\boldsymbol{G}'(\pi)$ based on our previous claims. Thus, basically, these two graphs give the same probablity for any fixed path and any fixed shortest path length for this path. So, the claim of the theorem holds. □

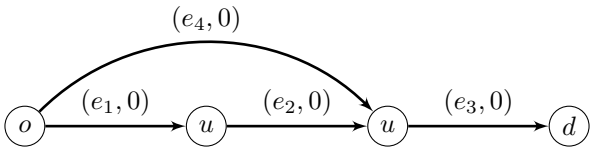


Figure 9: An example

Proposition 1. *The BSW-SPSG problem is NP-hard and its objective function is neither sub- nor super-modular.*

Proof. Given an instance of Knapsack problem, we show it is a special case of the BSW-SPSG problem. The Knapsack problem is defined by

$$\max \sum_{i=1}^{N} p_i x_i \quad s.t. \quad \sum_{i=1}^{N} c_i x_i \leq \mathcal{B}$$

(where p_i is profit and c_i is cost) which can be rewritten as

$$\min \sum_{i=1}^{N} M_i x_i \quad s.t. \quad \sum_{i=1}^{N} c_i x_i \leq \mathcal{B}$$

where $M_i = M - p_i > 0$ and M is the upper bound of all p_is.

To build the BSW-SPSG problem for this Knapsack problem, we define the graph (V, E). V contains $N + 1$ vertices or $V = \{o\} \cup \{d_i : i = 1 : N\}$. E contains N edges $E = \{(o, d_i) : i = 1 : N\}$. The length of each edge is 0. Define N edge sets, each one as $E_i = \{(o, d_i)\}$ with the cost c_i. Let $\Theta = \{(o, d_i) : i = 1 : N\}$ and (o, d_i) has penalty M_i. It is easy to see that the constructed BSW-SPSG problem is equivalent to this instance of the Knapsack problem.

To show that the objective function is neither submodular nor supermodular, we use the simple graph as shown in Fig. 9 with $V = \{o, u, v\}$ and $E = \{e_1, e_2, e_3, e_4\}$. Lengths of all edges are 0. Let the unique o/d pairs be (o, d) with the penalty $M_{o,d} > 0$ and the collection of edge sets be $\mathcal{E} = \{\{e_1\}, \{e_2\}, \{e_3\}, \{e_4\}\}$. Let $f(\cdot)$ be the objective function defined based on (4). Given a set of edges being purchased, if there is a path from o to d, the value of the objective function is 0 because the shortest path length is 0. Otherwise, the objective function equals to the penalty $M_{o,d}$ because the shortest path from d to o doesn't exist. For example,

$$\begin{aligned} &f(\phi) = M_{o,d} \\ &f(\{e_3\}) = M_{o,d} \\ &f(\{e_1, e_2\}) = M_{o,d} \\ &f(\{e_1, e_2, e_3\}) = 0 \\ &f(\{e_4, e_3\}) = 0 \end{aligned}$$

If we let $A = \phi$, $B = \{e_1, e_2\}$ and $x = \{e_3\}$ satifying $A \subseteq B$ and $x \notin B$, we have

$$f(A \cup \{x\}) - f(A) = 0 > f(B \cup \{x\}) - f(B) = -M_{o,d}$$

Thus, the objective function is not supermodular.

If we let $A = \{e_2, e_3\}$, $B = \{e_2, e_3, e_4\}$ and $x = \{e_1\}$ with $A \subseteq B$ and $x \notin B$, we have

$$f(A \cup \{x\}) - f(A) = -M_{o,d} < f(B \cup \{x\}) - f(B) = 0$$

Thus, the objective function is not submodular.

□

Proposition 2. *The running time of Algorithm 1 is bounded by* $O\big(\ (|E^s| + |\mathcal{E}| + |\Theta^s|)^2 + |\Theta^s|\ |E^s|\ \log|V^s|\ \big)$.

Proof. The algorithm takes at most $(|E^s| + |\Theta^s|)$ iterations, because at each iteration, either an edge is added into F or a pair becomes inactive (abandoned or connected). Note Θ^s is the o/d pair set for the BSW-SPSG problem. When all edges are in F, the loop terminates too. In line 8, we calculate the minimium amount that the variable μ of all active C^{od} can increase. To do this, in the worst case, at each iteration, we need to check constraint (4) in Fig. 3 for all edges (note that all o/d pairs increasing uniformly), constraint (5) in Fig. 3 for all pairs, and constraint (6) in Fig. 3 for all edge sets. It takes time $O(\ |E^s| + |\Theta^s| + |\mathcal{E}|\ )$. In total, the line 8 takes $O(\ (|E^s| + |\mathcal{E}| + |\Theta^s|)^2\ )$. Line 14 will only take $O(|E^s||\Theta^s|)$ in total because each edge will be expanded at most once for each pair. The shortest path calculation in line 21 takes time $O\big(|\Theta||E^s|\log|V^s|\big)$ in total for all pairs if we use the Dijkstra's algorithm for each pair, which takes time $|E^s|\log\ |V^s|$. Except these lines, other lines take insignificant runtime. Thus, the total runtime is bounded by $O(\ (|E^s| + |\mathcal{E}| + |\Theta^s|)^2 + |\Theta||E^s|\log|V^s|\ )$.

□

Proposition 3. *The running time of solving a GPTNP problem is* $O\big(K\ N^2\ (|E| + |\Theta|)^2 + K\ N\ |\Theta|\ |E|\ \log\ |V|\ \big)$.

Proof. To solve the given GPTNP problem, we first use the sampling procedure to construct a BSW-SPSG problem with N samples. The time of sampling is insignificant and can be ignored. To solve the BSW-SPSG problem, we call the bisection procedure that invokes the Algorithm 1 once at each iteration.

First, let's analyze the running time of the Algorithm 1 in the constructed BSW-SPSG problem where $|E^s| \leq 2N{\cdot}|E|$, $|V^s| = N{\cdot}|V|$ and $|\Theta^s| = N{\cdot}|\Theta|$. Also, we know $|\mathcal{E}| \leq |E|$. For this constructed BSW-SPSG problem, the runtime of the Algorithm 1 changes as follows. In line 8, the running time becomes $O(\ (N\ |E| + N\ |\Theta|)^2\ )$. In line 14, the total runtime becomes $O(N^2\ |E|\ |\Theta|)$. In line 21, for an o/d pair, to calculate its shortest path length, we only need to consider the sample graph that the pair belongs to, so the Dijsktra's algorithm takes time at most $O\big(|E|\ \log\ |V|\big)$ while a sample graph has $|V|$ vertices and at most $2|E|$ edges. The total running time of this step is $O\big(N\ |\Theta|\ |E|\ \log\ |V|\big)$ since we have $N\ |\Theta|$ pairs. Therefore, one call of the Algorithm 1 takes time

$$O\ \big(\ N^2\ (|E| + |\Theta|)^2 + N\ |\Theta|\ |E|\ \log\ |V|\ \big)$$

Let K be the number of iterations in the bisection procedure. The total runnning time for solving the GPTNP problem is bounded by

$$O\ \big(\ K\ N^2\ (|E| + |\Theta|)^2 + K\ N\ |\Theta|\ |E|\ \log\ |V|\ \big)$$

□

Proposition 4. *The runtime of a greedy algorithm is bounded by* $O\big(\ K'\ N\ |\mathcal{E}|\ |\Theta|\ |E|\ \log|V|\ \big)$.

Proof. The greedy algorithm has K' iterations, each one selecting an edge to invest in. At i^{th} iteration, we check $|\mathcal{E}| - i$ edge sets. By adding each one of these edge sets into the current graph, we check how much the total travel time decreases by calculating the shortest path length for all o/d pairs. In the worst case, none of these pairs share the same vertex, so we need to calculate shortest path lengths for all pairs. To calculate the shortest path length for an pair, we run the Dijkstra algorithm but terminates as long as the shortest path length between this pair is calculated. Usually, only a small portion of vertices in the sample graph, which the pair belongs to, are visited but in the worst case, all vertices and edges are visited, which takes time $O\big(|E|\ \log\ |V|\big)$ while a sample graph has $|V|$ vertices and at most $2\ |E|$ edges. There are $N|\Theta|$ pairs. Therefore, the total running time is bounded by $O\big(K'\ N\ |\mathcal{E}|\ |\Theta|\ |E|\ \log|V|\big)$

□