November 2016

Stochastic Network Design: Models and Scalable Algorithms

Xiaojian Wu

STOCHASTIC NETWORK DESIGN: MODELS AND SCALABLE ALGORITHMS

A Dissertation Presented

by

XIAOJIAN WU

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2016

College of Information and Computer Sciences

STOCHASTIC NETWORK DESIGN: MODELS AND SCALABLE ALGORITHMS

A Dissertation Presented

by

XIAOJIAN WU

Approved as to style and content by:

Daniel Sheldon, Co-chair

Shlomo Zilberstein, Co-chair

Don Towsley, Member

Scott Jackson, Member

James Allan, Chair
College of Information and Computer Sciences

DEDICATION

To my family and parents

ACKNOWLEDGMENTS

First of all, I want to thank my advisors Daniel Sheldon and Shlomo Zilberstein. In my Ph.D study, I received lots of help and great advice on research, scientific writing, academic presentation, and career planning. Their advice, deep insight into research, and encouragement are important factors to the completion of this thesis.

I also thank my two committee members. Don Towsley provided insightful comments on my thesis. Scott Jackson provided guidance on conducting multidisciplinary research.

I give my special thank to my wife, Jing Liu, who encouraged and supported me all the time during my Ph.D study.

I am very lucky to work in the RBR lab. Thanks Akshat for inspiring my initial interests in network optimization and computational sustainability, Rick for those wonderful chat during nights in the lab, Haochong for introducing the philosophy of research, Luis and Kyle for being travel companions, Yi for playing basketball together, Yoonheui for having interesting chats.

I also want to thank my friends for giving me an enjoyable life while at UMass. Thanks Bo, Kun for swimming together. Thanks Pengyu, Shaofang, Tao, Yanqin, Weixuan, Juhong, Bo, Weize, Chang, Tian, Yahan for Chinese festival celebration, parties and games during spare time. Thanks many of my friends for playing basketball. Thanks Steven for sharing research experience and helping to solve technical problems. These lists are not exhausted.

At last, I want to thank all other faculty and staff members at the College of Information and Computer Sciences and, particularly, Leeanne and Micheal for their help.

ABSTRACT

STOCHASTIC NETWORK DESIGN: MODELS AND SCALABLE ALGORITHMS

SEPTEMBER 2016

XIAOJIAN WU

B.E., WUHAN UNIVERSITY, CHINA

M.S., MISSISSPPI STATE UNIVERSITY

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Daniel Sheldon and Professor Shlomo Zilberstein

Many natural and social phenomena occur in networks. Examples include the spread of information, ideas, and opinions through a social network, the propagation of an infectious disease among people, and the spread of species within an interconnected habitat network. The ability to modify a phenomenon towards some desired outcomes has widely recognized benefits to our society and the economy. The outcome of a phenomenon is largely determined by the topology or properties of its underlying network. A decision maker can take *management actions* to modify a network and change the outcome of the phenomenon. A management action is an activity that changes the topology or other properties of a network. For example, species that live in a small area may expand their population and gradually spread into an interconnected habitat network. However, human development of various structures such as highways and factories may destroy natural habitats or block paths connecting different habitat patches, which results in a population decline. To facilitate the

dispersal of species and help the population recover, artificial corridors (e.g., a wildlife highway crossing) can be built to restore connectivity of isolated habitats, and conservation areas can be established to restore historical habitats of species, both of which are examples of management actions. The set of management actions that can be taken is restricted by a budget, so we must find cost-effective allocations of limited funding resources.

In the thesis, the problem of finding the (nearly) optimal set of management actions is formulated as a discrete and stochastic optimization problem. Specifically, a general decision-making framework called *stochastic network design* is defined to model a broad range of similar real-world problems. The framework is defined upon a stochastic network, in which edges are either present or absent with certain probabilities. It defines several metrics to measure the outcome of the underlying phenomenon and a set of management actions that modify the network or its parameters in specific ways. The goal is to select a subset of management actions, subject to a budget constraint, to maximize a specified metric.

The major contribution of the thesis is to develop scalable algorithms to find high-quality solutions for different problems within the framework. In general, these problems are NP-hard, and their objective functions are neither submodular nor super-modular. Existing algorithms, such as greedy algorithms and heuristic search algorithms, either lack theoretical guarantees or have limited scalability. In the thesis, fast approximate algorithms are developed under three different settings that are gradually more general. The most restricted setting is when a network is tree-structured. For this case, fully polynomial-time approximation schemes (FPTAS) are developed using dynamic programming algorithms and rounding techniques. A more general setting is when networks are general directed graphs. We use a sampling technique to convert the original stochastic optimization problem into a deterministic optimization problem and develop a primal-dual algorithm to solve it efficiently. In the previous two problem settings, the goal is to maximize connectivity of networks. In the most general setting, the goal is to maximize the number of nodes being

connected and minimize the distance between these connected nodes. For example, we do not only want the species to reach a large number of habitat areas but also want them to be able to get there within a reasonable amount of time. The scalable algorithms for this setting combine a fast primal-dual algorithm and a sampling procedure.

Three real-world problems from the areas of computational sustainability and emergency response are used to evaluate these algorithms. They are the barrier removal problem aimed to determine which instream barriers to remove to help fish access their historical habitats in a river network, the spatial conservation planning problem to determine which habitat units to set as conservation areas to encourage the dispersal of endangered species in a landscape, and the pre-disaster preparation problem aimed to minimize the disruption of emergency medical services by natural disasters. In these three problems, the developed algorithms are much more scalable than the existing state-of-the-arts and produce high-quality solutions.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

Many natural and social phenomena occur in networks. For example, information spreads among people [18, 42] through a social network. In a social network, people are represented by nodes, and an edge may represent the friendship between two people. If an individual receives a piece of information sent by their friends, she/he may continue to send it to her/his other friends. Over time, more and more people will receive the information, forming a so-called cascading process [42]. Another example is the spread of species within a landscape [33, 97]. Historical habitats within a landscape are divided into a finite number of habitat areas, each of which is represented by a node, and are interconnected by natural paths that species can travel along, each of which is represented by an edge. Species that live in one habitat area may expand their population and spread into a nearby habitat area along the natural paths. Over time, the population of the species increases, and a large number of habitat areas are occupied by species. This process can be modeled as a diffusion process in an interconnected habitat network.

Management actions can be taken to modify the underlying networks and reshape the phenomena toward some desired objective. For example, a company can provide rewards as one kind of management actions to inspire people to share information about a product through a social network to a large number of people—an advertising method called viral marketing [18, 57]. Ecologists want to facilitate the dispersal of endangered species of one kind by constructing corridors, as management actions, to connect fragmented habitats [7, 66, 67]. Habitats are fragmented by human constructed barriers, such as highways and railroads, that cut off the natural paths for species to move between habitats. Other

examples of management actions can be to improve the reliability of telecommunication networks [90], to control the spreading of epidemics [70], to minimize the disruption of emergency services by a natural disaster [60, 72].

A decision maker wants to know how to select a good set of management actions from a large number of candidate actions to steer the outcome of a phenomenon towards a certain desired objective. For example, among millions of people in a social network, a company may need to decide who to provide the rewards, or an environmental agency may need to select among hundreds or thousands of damaged habitat areas which ones should be recovered or set as conservation areas. Broadly, this question asks how to allocate resources in an efficient manner to maximally achieve certain desired objective. In most cases, the effect of one action relies on the execution of several other actions, meaning that selecting actions greedily by maximizing the marginal gain of an objective function does not provide the optimal, or even a good, strategy. The decision is often made subject to a budget limit. The purpose of this thesis is to develop tools to help a decision maker find cost-efficient management actions by solving an optimization problem.

Typical network design problems are defined to model similar network optimization problems. Example network design problems include the Steiner tree problem [37, 75] and the maximum spanning tree problem [51, 74]. These problems have been formulated and extensively studied [30, 40] and used successfully in many real-world applications, such as the design of telecommunication, traffic networks, and VLSI chips. The basic structure of a typical network design problem is that we are given a graph and a set of extra edges that can be purchased and added to the graph and are required to design an algorithm to find the least costly set of edges so that the graph augmented with the purchased edges accomplishes some desired objectives. For the *Steiner tree problem*, the input consists of an undirected graph, a subset of specified nodes called terminal nodes, which, for example, can represent locations that a truck needs to reach, and a set of edges that can be purchased and added to the graph. The desired output is the cheapest set of edges with which all terminal nodes are

interconnected in the augmented graph. The Steiner tree problem can model a real-world problem in which we want to build a road network to connect several important locations, represented by terminal nodes, by building roads, represented by edges, in an existing road network.

However, typical network design problems fail to capture the uncertainty or the stochasticity that exists in a natural or social phenomenon such as the spread of information over a social network and the dispersal of species within a landscape. Consider the same example of adding edges to a road network to connect several important terminal nodes. By solving a Steiner tree problem, we can find the least costly set of edges if all these edges or roads guarantee to work. However, in reality, some of these edges may malfunction due to various reasons such as natural disasters. For example, if a flood or an earthquake happens, some roads may be destroyed, and become impossible for traffic to pass. To guarantee the connectivity of a pair of nodes, one way is to either build two parallel edges to connect them or build a reinforced edge that is resistant to natural disasters such as floods and earthquakes. The question asks which pairs of nodes we should invest money to connect. While making the decision, we want to take into account which edge will be under attack by a natural disaster, and how severe the damage will be. Although the perfect prediction for these events is unavailable, we could treat them as random events and build a predictive probabilistic model to describe them. Therefore, a better decision-making framework can be built by combining the Steiner tree problem with a probabilistic model. Another generalized version of the Steiner tree problem called *survivable network design* deals with uncertainty from a different perspective. In a survivable network design problem, multiple disjoint paths between two terminal nodes are required to be constructed so that the connectivity between them remains valid even although one path is destroyed by a natural disaster. This method doesn't explicitly model the probabilities of edge failures, but chooses a safe way by building multiple paths beforehand with the assumption that the likelihood that all these paths fail simultaneously is very low. This method is related to an active research

area called *robust optimization* [5] where the decisions are made to optimize the worst outcomes. Robust optimization is widely used when a probabilistic model of the involved random events is not available. Otherwise, it is too pessimistic to consider the worst outcomes, and, usually, the solution produced by solving a robust optimization problem turns out to be too conservative and not cost-efficient. A better way is to take advantage of the available probabilistic model to formulate the decision-making problem. For example, if, statistically, floods are very unlikely to severely damage any roads within a certain area, the priority of building multiple roads to connect two locations to guarantee connectivity should be low. The ability to model the uncertainty of a phenomenon is the first step to produce a high-quality strategy to modify the phenomenon towards our desired direction.

The major contribution of the thesis is to provide scalable algorithms to compute high-quality solutions for a range of decision-making problems including the examples discussed earlier.

First, I propose a framework called *stochastic network design* to model a range of network design problems by taking into account the uncertainty existing in the underlying phenomenon. The way to model the uncertainty is to use a *stochastic network*. A stochastic network is a directed graph in which the presence or absence of an edge is a random event. The absence of an edge means that the edge suffers from a failure; a road, for example, is destroyed by a natural disaster. Management actions can be taken to change the probabilities of presence. An evaluation function is also defined to measure the performance a set of taken actions, and a budget constraint that limits the set of actions that can be taken. For example, in the influence maximization problem [42], the evaluation function measures the expected number of people who will receive a piece of information if the information initially starts from a source and gradually spreads out among people over time. An evaluation function maps a subset of actions into a real value. The goal of the framework is to select a subset of actions that satisfy the budget constraint to produce the maximum (or minimum) value of the evaluation function. The framework can model a

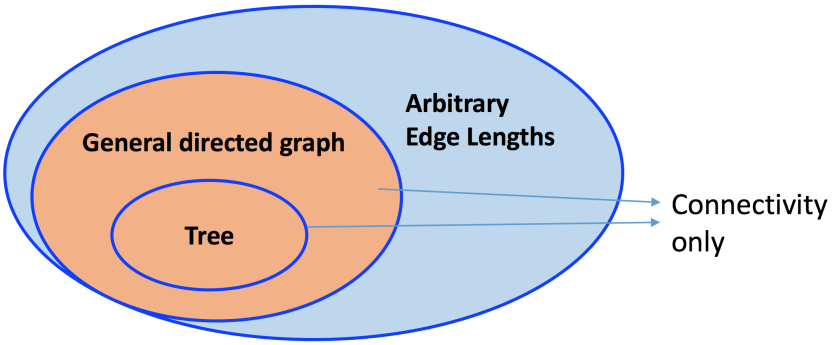


Figure 1.1: Subclasses of stochastic network design problems

range of real-world decision-making problems if the presence probabilities, management actions, and the evaluation function are defined properly. Both the information spreading problem and the species spreading problem discussed earlier can be modeled by the framework.

Second, I develop methodologies and scalable algorithms to solve problems within the stochastic network design framework under three different settings, as shown in Fig 1.1, that are gradually more general. In the most restricted setting, the underlying network is assumed to be tree-structured. A real-world problem that satisfies this assumption is the so-called *barrier removal problem* that has been widely studied by researchers from environmental science and operations research [65, 63]. In a barrier removal problem, the goal is to maintain the connectivity of a river network by removing river barriers, such as dams and culverts that prevent fish from moving upstream, so that fish once enter the river from the ocean can access their historical river habitats and produce next generation. The modeling works well because a river network can be treated as a tree [65]. In the slightly more general setting, it is assumed that the underlying network is a general directed graph. This setting can be used to model a *conservation planning problem* [87], of which the goal is to conserve the population of an endangered species called red-cocked woodpecker (RCW). Human constructions and activities destroyed the natural habitats of RCWs and resulted in a decline of their population. In this problem, we want to decide which disrupted habitats,

which are either currently occupied or will potentially be occupied, to set as conservation areas to encourage the dispersal of RCWs. The influence maximization problem [42] can be modeled by this setting as well. In the first two settings, the focus is only on the connectivity of a network but not the distance between nodes. For the information spreading problem, under these two settings, we only maximize the number of people that the information can eventually reach but not worry how long it takes for those people to receive the information. The last setting, which is more general, is to maximize the connectivity and also minimize the distance. To adapt the stochastic network design framework to this new setting, each edge has a length that is randomly distributed in a range $[0, \infty]$. That is, we do not only want the information to reach a large number of people in the network but also want it to reach those people within a reasonable amount of time (e.g., within a week). A decision-making problem under this last setting is related to a well-known problem called *continuous-time influence maximization problem* in the area of computational social network [77]. In this thesis, a real-world problem called *pre-disaster preparation* is formulated using the stochastic network design framework under this setting. The goal of the problem is to selectively reinforce several road segments before a natural disaster, such as a flood, happens, such that when a disaster indeed happens, the response time of Emergency Medical Services (EMS), the time that an ambulance takes to travel from an ambulance center reach a patient who made the call, is minimized.

A problem,in general, for any one of these three settings is NP-hard, so the focus of the thesis is to develop approximate algorithms. When the underlying network is a tree, approximation algorithms called fully polynomial-time approximation schema (FPTAS) are developed to produce near optimal solutions within a reasonable amount of time. When applying to the barrier removal problem [65], the algorithm runs around 60 times faster than a state-of-the-art algorithm and produces a solution with the value within 99% optimal value. When the underlying network is a general directed graph, an FPTAS becomes unavailable, but I develop a fast approximate algorithm by combining a sample average ap-

proximation method and a primal-dual algorithm. As demonstrated later, the algorithm is faster than a greedy baseline and produces high-quality solutions for the conservation planning problem [87]. For the most general setting where the lengths of edges are considered, all previous algorithms are not applicable, but a new approximate algorithm is proposed to deal with problems of large sizes, which a greedy algorithm takes more than 100 hours to solve. As shown by experiments, this new algorithm can produce near optimal solutions and much faster than several existing algorithms.

Several other algorithms and optimization techniques are relevant but cannot solve stochastic network design problems with satisfiable performance and scalabilities. Approximation algorithm design techniques have been widely studied to solve typical network design problems such as the Steiner tree problem [37, 75], the maximum spanning tree problem [51, 74] and the facility location problem [39]. But, since these network design problems fail to model the stochasticity that exists in the problem, these algorithms can't be applied directly. Local search algorithms can give empirically good solutions for many NP-hard problems [26, 36, 56, 61, 71]. A local search algorithm is usually fast and can stop at any time to produce a feasible solution but does not give any theoretical guarantee on the quality of the computed solution. Also, a local search algorithm is domain-dependent, that is, an algorithm that can produce a high-quality solution for one problem is hardly to be transported to solve another problem even although these two problems share several common properties, because the performance of a local search algorithm largely relies on the way that the neighborhood of a solution is defined and how a starting point of search is selected, both of which are designed based on heuristic and are hardly to generalize to a different problem. More discussion of the performance of local search algorithms is beyond the scope of this thesis. In comparison, the algorithms provided in the thesis are very general to solve any problems within the framework, and for a tree-structured network, the theoretical result is given to bound the quality of a computed solution. Moreover, as shown in the experiments, these algorithms can be hundreds times faster than a greedy algorithm

that is one type of fast local search algorithms, that is, they are superior to local search algorithms also in terms of time efficiency. Heuristic search algorithms [48, 49, 79] are widely used to find the optimal solution or a near optimal solution [32] for a NP-hard problem. With a good heuristic, a heuristic search algorithm can run very fast because a large portion of search space is pruned and is never explored by the algorithm. The problem is that it is hard to design a good heuristic function in many domains, and in some cases, computing a good heuristic value takes almost the same amount of time as solving the original optimization problem. And, a carefully designed heuristic function for one problem is hardly useful to solve a different problem. In this thesis, the interest is to develop several general algorithms that can solve a range of decision-making problems. These problems can be modeled by a unified framework, but are quite different. It is unclear how to design a general heuristic function that can work for all these problems. Also, since the number of candidate actions is usually proportional to the number of edges in a network, which can be thousands or millions, the search tree of a heuristic search algorithm can be very deep such that even with a fairly good heuristic, the algorithm may take a long time to finish. As demonstrated later, the heuristic search algorithm can find optimal solutions for a pre-disaster preparation problem [83] but can only scale up to a network with 1000 edges while my algorithms can deal with a network with $50,000$ edges for that problem. Other problems that are tested can be millions of candidate actions, of which a heuristic search algorithm is obviously not viable. Greedy algorithms are fast, of which the runtime are usually quadratic the number of candidate actions, but don't have any guarantee on the solution quality for most of NP-hard problems. A constrained discrete optimization problem that a greedy algorithm has theoretical guarantee on needs to satisfy certain properties [21, 24, 28, 50] such as the objective function being submodular and nondecreasing and having a knapsack constraint [91]. For problems that don't have these properties, the greedy algorithm can perform arbitrarily bad. As shown later, a stochastic network design problem under any one of three settings doesn't satisfy these properties. For example, the

objective function is neither submodular or supermodular. In this sense, greedy algorithms are ideal algorithm of solving stochastic network design problems. In experiments shown in other chapters, a greedy algorithm is used as a baseline algorithm to compare with other algorithms. Also, for some real-world problems such as a pre-disaster preparation problem, a greedy algorithm is unable to compute a solution within 100 hours. For the same problem, my algorithm only takes 6 hours. In total, approximation algorithm design, local search, heuristic search, and greedy search are useful techniques to deal with NP-hard problems, but my fast approximate algorithms are better options in aspects of generality and scalabilities.

In summary, the thesis studies one type of decision-making problems that we want to find a good set of management actions to steer a phenomenon that occurs in network towards certain desired goal. A general framework called stochastic network design is defined to model a range of such problems, and several scalable algorithms are developed to produce high-quality solutions. These algorithms are superior to several existing techniques that are widely used to deal with NP-hard problems regarding generality and scalability. In next section, several relevant problems studied in other fields are connected to the problem studied in this thesis. The similarities and differences between them are discussed.

1.1 Related Works

Several research areas are related to our work.

1.1.1 Network Reliability

In telecommunication networks, *network reliability* describes the ability of a network to continue network services in the case of component failures [3]. In the basic model, each edge or node has an independent failure probability, and different metrics are used to measure the reliability, such as *all-terminal reliability* (the probability that the graph remains connected) and *s-t reliability* (the probability that there is a path from s to t).

However, most widely used metrics are intractable ($\#P$-hard) to compute exactly [3, 11], and polynomial-time algorithms exist only for very restricted cases (e.g. series-parallel graphs) [12, 80]. The reliability network design problem is to find a network topology maximizing the reliability for a given cost constraint or minimizing the design cost for a given reliability constraint [47]. As the reliability is hard to calculate, most of the existing methods are only scalable on small networks. For example, a branch-bound algorithm, a genetic algorithm, and a neural network approach have been used but can only scale up to hundreds of edges [15, 78, 89].

The stochastic network design framework can be specialized to model several common metrics of reliability, such as s-t reliability and all-terminal reliability, so the stochastic network design algorithms can be applied to reliability network design problems. They have a better scalability, for example up to hundreds of thousands of nodes, and can produce high-quality solutions with theoretical and empirical evidence.

1.1.2 Influence Maximization Problem

Influence maximization problem recently becomes a hot topic in the area of computational social network. The spread of information among people is modeled as a cascading process through a social network, and several well-known cascade models, such as the Independent Cascade (IC) model [42], the Linear Threshold (LT) model [42], and the Continuous-Time Independent Cascade (CTIC) model [19, 77], are used to describe such a process. Each of these models defines a local rule and specifies how information spreads through a network following this local rule. For example, a set of nodes called *sources* are *infected* or *active* at the beginning of the process. Recursively, an infected node can infect its neighboring nodes, and the local rule, such as a probability distribution, describes how each neighboring node gets infected. After a certain number of such *cascading* cycles, a large number of nodes become infected. For an influence maximization problem, the goal is to maximize the (expected) number of nodes that become infected eventually by taking

management actions. One type of influence maximization problems is the *source selection* problem, motivated by an active research area called *viral marketing* [57], where a decision maker wants to pick K nodes, also called sources, to make them infected at the beginning of the process to maximize the number of nodes infected at the end of the process [18]. This problem has a submodular objective function [42] and, a natural greedy algorithm can produce a solution with the value within 63% of the optimal value. A lot of following work has been done to improve the greedy algorithm to scale up to networks of millions of nodes [8, 9, 10], but it hard to extend these algorithms to problems of other types of management actions than selecting sources. Khalil *et al.* [44] study a problem of modifying the topology of a network, such as adding edges, deleting edges, adding nodes and deleting nodes, to maximize/minimize the influence under the LT model. The objective function has shown to be either sub-modular or super-modular, and the technique of submodular function optimization is used to develop algorithms producing high-quality solutions. Sheldon *et al.* [87] study a problem of maximizing the spread of species in a habitat network by adding nodes. The optimization problem doesn't have a sub-modular or super-modular objective function. A *sample average approximation* technique [45] is used to formulate the problem into a mixed integer program (MIP) and a standard solver is used to solve the MIP. As shown by experimental results, the algorithm produces high-quality solutions, but only works when the underlying network is a *acyclic directed graph* (DAG). In addition, the algorithm can only scale up to networks of several hundreds of nodes. Instead of maximizing (minimizing) the spread of influence directly, some other methods optimize a static property of the network, in the hope of optimizing diffusion. For instance, Schneider *et al.* [84] suggest *betweenness centrality* as a heuristic for immunizing nodes or removing edges under the SIR model, while *degree centrality* is adopted by Gao *et al.* [25] to protect against virus propagation in email networks.

All these influence maximization problems with actions including selecting sources, adding (removing) nodes and edges, under the IC, LT and CTIC models are special cases of

the stochastic network design framework. Besides modifying the topology of the network, the framework also models richer management actions, such as increasing (decreasing) the probability that a node infects its neighbors, which has correspondence in reality. For example, to maximize the number of purchases by a viral marketing strategy, a company will offer a person rewards if her friends purchase the product due to her recommendation [57]. The rewards, as one type of management actions, can increase the chance that a person sends the recommendation of one product to her friends, which raises the chance that a friend will also purchase the product. To efficiently allocate rewards, we want to determine who are the good candidates to provide the rewards and who should get more rewards than others. Also, my algorithm can solve the problem introduced by Sheldon *et al.* [87], deal with a general directed graph, and runs much faster than their algorithm.

1.1.3 Conservation Planning

Conservation planning that has recently been taken up in the emerging field of computational sustainability [29]. One branch of conservation planning problems is known as reserve design or corridor design in which a decision maker selects management actions, such as purchasing or improving habitats and constructing a wildlife corridor connecting two fragmented habitats, to optimize certain metric or the connectivity of landscape to facilitate species dispersal [7, 66, 67, 95]. A number of recent works connect the reserve design problems to typical network design [13, 14, 16, 17, 54, 55]. In most of these works, the goal is to construct, by purchasing nodes or edges, a subgraph minimizing the design cost for a given connectivity constraint, such as connecting all terminals, or maximizing certain connectivity metric for a given cost constraint, such as the total design cost less than a budget limit. However, as mentioned earlier, these network design problems don't explicitly model the stochasticity of the activity of species within a landscape. A separate group of works incorporate the stochasticity into an evaluation function that measures how good a conservation strategy is [69, 81, 85]. For example, a well-known method, adopted by ecol-

ogists, to evaluate a conservation policy is called *probability of connectivity* (PC) [81], that is, the probability that one node is connected to other nodes. For a given policy, calculation of its PC score is very time-consuming, so finding the policy to maximize PC score is a complex stochastic optimization problem. Heuristic based methods and greedy algorithms are used by ecologists to solve this problem [92, 59, 62], which, as mentioned earlier, do not have any theoretical guarantee on the quality of the computed solution. The conservation planning problem with the PC evaluation function can be formulated as a stochastic network design problem, and my algorithms are capable of solving the problem.

1.2 Organization of the Thesis

The rest of this thesis is organized as follows. Chapter 2 defines the stochastic network design framework and its extensions. Chapter 3 and 4 introduce the algorithms for tree-structured networks. Chapter 5 introduces the algorithms for general directed graphs. Chapter 6 introduces the algorithms for the setting that edges have arbitrary lengths.

CHAPTER 2

STOCHASTIC NETWORK DESIGN

This chapter introduces the stochastic network design framework. First, the basic setting of the stochastic network design framework is given. To help understand the motivation of the framework, two real-world problems are introduced, and the way to formulate these two problems as stochastic network design problems is explained. Then, several ways to build more complex settings by relaxing some of the assumptions of the basic setting are discussed. At last, the quota problem and the prize-collecting problem of a stochastic network design problem are briefly mentioned.

2.1 Basic Setting

This chapter introduces the mathematical formulation of the stochastic network design framework. It starts with the basic setting of the framework and discusses several ways to extend the framework. Then, several real-world problems are introduced and formulated as stochastic network design problems.

2.1.1 Stochastic Network

A stochastic network consists of a directed graph $G = (V, E)$, where V is a set of nodes and E is a set of directed edges, and a probability mass function $p : E \to [0, 1]$ that defines the probability that an edge doesn't fail. That is, each edge is in one of two exclusive states: present (survival) or absent (failed). The state of an edge is a random event following a Bernoulli distribution. The probability of an edge $(u, v) \in E$ being present is defined by $p(u, v)$, and the probability of the edge being absent is $1 - p(u, v)$.

The probability function $p(\cdot)$ is also called the *survival probability function* while $p(u, v)$ is called the *survival probability* of the edge (u, v), and $1 - p(u, v)$ is called the *failure probability*. To make the presentation clear, let's for now assume that the state of each edge is independently distributed, which is a reasonable assumption for some real-world problems as shown later in section 2.2. This assumption is relaxed while discussing the ways to extend this basic setting in section 2.4. The stochastic network, in fact, defines a probability distribution over a set Φ_G that contains all directed graphs $G^s = (V, E^s)$ with $E^s \subseteq E$, that is, $\Phi_G = \{G^s = (V, E^s) : E^s \subseteq E\}$. Each directed graph in Φ_G is called a *sample network*. The probability of a sample network G^s is defined to be the probability that all edges in E^s are present and all edges not in E^s are absent, which equals to the product of the survival probabilities of all edges in E^s and the failure probabilities of all edges not in E^s. That is

$$Pr(G^s) = \prod_{(u,v)\in E^s} p(u, v) \prod_{(u,v)\notin E^s} (1 - p(u, v)) \tag{2.1}$$

It is easy to show that probabilities of all sample networks in Φ_G sum to one. To sample from the distribution defined by a stochastic network, a sample network is constructed by flipping a biased coin with probability $p(u, v)$ to determine the state of each edge (u, v).

The stochastic network can model many social and natural phenomena. For the influence maximization problem, the probability that a node u is connected to another node v in a stochastic network represents the probability that information propagates from u to v in a social network. More details on how to formulate the influence maximization problem as a stochastic network design problem is gave later in section 2.2.

2.1.2 Management Actions

Management actions can be taken to change the survival probabilities of edges. For each edge (u, v) in a stochastic network, there is a management action a_{uv}, with cost c_{uv}, available to modify its survival probability. To describe the survival probabilities of edges

before and after actions are taken, two probability mass functions are defined. A probability mass function $p^o : E \to [0, 1]$ defines the *original* (by superscript o) survival probabilities of edges, and a different probability mass function $p^m : E \to [0, 1]$ defines the *modified* (by superscript m) survival probabilities after actions are taken. That is, if a_{uv} is not taken, $p(u, v) = p^o(u, v)$ (p^o_{uv} for short) and if a_{uv} is taken, $p(u, v) = p^m(u, v)$ (p^m_{uv} for short). Let the set of all available actions denoted by $\mathbb{A} = \{a_{uv}|(u, v) \in E\}$ and define a policy π to be a subset of $\mathbb{A}$. The survival probability function is re-defined to be conditional on π as

$$p(u, v; \pi) = \begin{cases} p^o_{uv} & \text{if } a_{uv} \notin \pi \\ p^m_{uv} & \text{if } a_{uv} \in \pi \end{cases} \tag{2.2}$$

The definition of management actions is very general. For an influence maximization problem, a management action models the activity that rewards are provided to a person A to increase the chance that a person A sends the information to a person B. For a conservation planning problem, an action models the activity of building a corridor to connect two fragmented habitat areas A and B. The corridor increases the chance (modeled as the survival probability) that species can spread from A to B. Each action is associated with a cost, which is defined by a cost function $c : E \to \mathbb{R}$. The cost of an edge (u, v) is written as c_{uv} for short. For now, it is assumed that each edge has only one candidate action, which is reasonable from some real-world problems as shown later. This simple assumption makes the presentation clear and is relaxed later in section 2.4. If no candidate action is available, we can simply set $p^o_{uv} = p^m_{uv}$ and $c_{uv} = 0$.

2.1.3 Objective Function

Given a stochastic network G, we need to have an evaluation function to quantify how good a policy π is applied to G.

First, a function $r : V \times V \to \mathbb{R}$ is defined to quantify the benefit that we can obtained by connecting one pair of nodes. For example, if $r(s, t) > r(u, v)$, it benefits us more to

have the pair (s,t) connected than the pair (u,v). The function is also called the *reward function*, that is, $r(v,u)$ represents the reward that we can obtain if v is connected to u. With a reward defined for each node, the *reward of a sample network* is calculated as

$$r(G^s) = \sum_{(s,t)\in V\times V} r_{st} \cdot \mathbb{I}(\text{there is a path from } s \text{ to } t \text{ in } G^s) \tag{2.3}$$

where $\mathbb{I}$ is an indicator function which returns 1 if the specified event is true and returns 0 otherwise. This is to say that for a sample network, a reward r_{st} is collected for each node pair (s,t) if there is a path from s to t, and the reward of the sample network is the total rewards collected.

Then, a *resilience* function is defined to quantify the outcome of a policy π by the expected reward that can be obtained when π is applied. Mathematically,

$$R(\pi;G) = \sum_{G^s\in\Phi_G} Pr(G^s;\pi)r(G^s) \tag{2.4}$$

With a policy, the management action of each edge is determined. Then, the survival probability of each edge is fixed, so the probability of a sample network is calculated using the (2.1). Since the probability of a sample network also depends on the policy π taken, it is written as a function of both G^s and π as shown in (2.4). The name resilience in some way represents the robustness of the network's connectivity to random failures, which is represented by the expected total rewards in the stochastic network design framework. Other ways to represent the resilience are discussed later. The resilience function is the objective function that we want to maximize.

2.1.4 Decision Making Problem

Now, we can define the decision-making problem, a discrete optimization problem, for the stochastic network design framework. For a given policy $\pi \subseteq \mathbb{A}$, the cost of a policy is defined to be the total cost of actions taken, that is, $cost(\pi) = \sum_{a_{uv}\in\pi} c_{uv}$.

The inputs of the decision-making problem consist of a stochastic network $G = (V, E)$, an original survival probability function p^o (representing the probabilities before actions are taken), a modified survival probability function p^m (representing the probabilities after actions are taken), a reward function r that defines a resilience function, and a budget limit $\mathcal{B}$. The goal is to find a policy π maximizing the resilience function defined by (2.4) for a given cost constraint $cost(\pi) \leq \mathcal{B}$. Mathematically, the decision-making problem is written as

$$\max_{\pi \subseteq \mathbb{A}} R(\pi) \qquad \text{subject to} \quad cost(\pi) \leq \mathcal{B} \tag{2.5}$$

where the dependency of the resilience function on G is omitted without confusion.

2.2 Two Examples of Stochastic Network Design Problems

This section introduces two real-world problems and explains how they can be formulated as stochastic network design problems. Some of these problems have been mentioned and used as examples earlier.

2.2.1 Influence Maximization by Source Selection

The influence maximization problem is introduced in Section 1.1. Here, we formally define the source selection problem and explain how it can be formulated as a stochastic network design problem. In the source selection problem, the diffusion process is described by the Independent Cascade (IC) model [43] on a directed graph $G' = (V', E')$. Each node is either infected (also called active) or uninfected (also called inactive). Each edge $(u, v) \in E'$ is associated with an *infection probability* p'_{uv} to represent the probability that node v will get infected if node u becomes infected. A set of nodes called *sources* are infected by some external sources at the beginning of the process. The process unfolds over a finite number of time steps. At each step, once a node u becomes infected, it will attempt to infect each of its neighboring nodes v independently with probability p'_{uv}. After

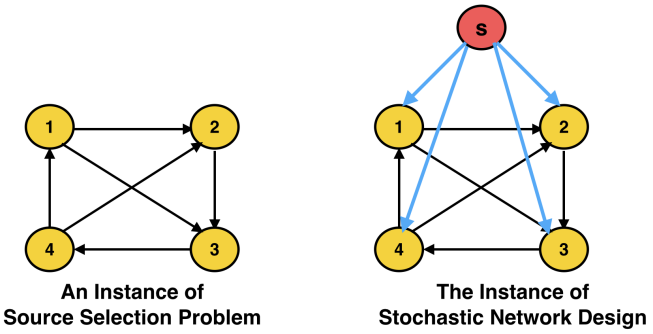


Figure 2.1: The instance of the stochastic network design problem equivalent to an instance of the source selection problem.

a certain number of time steps, a large number of nodes become infected in the network. The process terminates or becomes stable when no nodes will get infected in the future, that is, when all nodes have made their attempts. Since the diffusion of infection starts from sources, well-selected sources can make a large number of nodes infected at the end of the process while poorly selected sources, such as sources not connecting to any other nodes, can hardly spread out the infection. To facilitate the spread of infection, a decision maker wants to find the best set of sources. Since the decision maker also has a budget constraint, (e.g., at most K sources), in the *source selection problem*, the goal is to select K sources to maximize the expected number of nodes that will be infected at the end of the cascading process. The source selection problem can be written as follows.

$$\max_{\text{select } K \text{ sources from } V'} \mathbb{E}\left[\sum_{v \in V'} \mathbb{I}(v \text{ will be infected})\right]$$

where $\mathbb{I}(\cdot)$ is an indicator function.

Now, I show how to formulate the source selection problem as a stochastic network design problem. First, given an instance of the source selection problem with a social network represented by a directed graph G' and infection probabilities of all edges, a new graph $G = (V, E)$ is created where the new node set $V = \{s\} \cup V'$ contains an extra node s and the edge set $E = \{(s, v) | v \in V\} \cup E'$ contains a directed edge from s to each v in V.

An example is shown in Fig. 2.1. For each edge (u, v) in E', set the original and modified probability of the correspondent edge to be $p^o_{uv} = p^m_{uv} = p'_{uv}$ and set the cost of the action to be $c_{uv} = 0$ meaning that no actions can be taken to change the infection probabilities of edges in the social network. For each edge (s, v) with $v \in V - \{s\}$, set $p^o_{uv} = 0$, $p^m_{uv} = 1$ and $c_{uv} = 1$. That is, the action on (s, v) with one unit of cost can make node v infected with probability 1 or equivalently make v a source. Then, we set $\mathcal{B} = K$ meaning that at most K actions can be taken or equivalently at most K sources can be selected. Therefore, a policy of the stochastic network design problem maps to a set of sources being selected. To define the resilience function, we set the value of a policy to be the expected number of nodes being infected eventually. The way is to set $r(s, v) = 1$ for all $v \in V - \{s\}$ and the reward for other node pairs to be 0, that is, one unit of reward is collected for a node $v \in V - \{s\}$ if and only if the node v is connected to s. Now, it is easy to show that the stochastic network design problem encodes the source selection problem.

It is also easy to show that the problem of maximizing the influence by modifying the topology of the network [44] can also be formulated as a stochastic network design problem.

2.2.2 Barrier Removal Problem

Fish barrier removal problem is an important ecological sustainability problem proposed by [65] to combat dramatic population declines of wild fish over the past two centuries due to the presence of river barriers. Dams and other barriers such as culverts, floodgates, and weirs harm populations by preventing fish from accessing or moving between parts of their historical habitat. The way to address this problem is to retrofit existing barriers or to replace them by new instream structures that make it easier for fish to pass. The existing work formulated the optimization problem of selecting a subset of barriers to repair or remove so as to maximize the available *upstream* habitat for anadromous fish—species such as salmon that live part of the year in oceans but travel up rivers and streams to spawn.

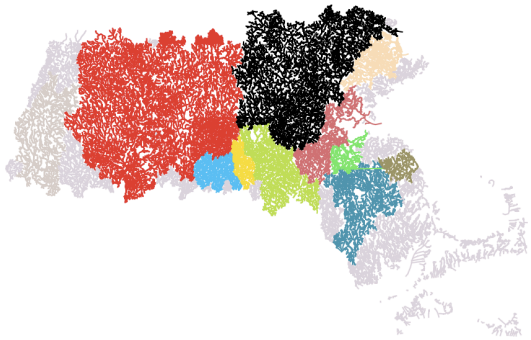


Figure 2.2: River networks in Massachusetts

Fig. 2.2 shows river systems of Massachusetts in which different color represents different river networks. Every year, fish start from a *root* (e.g., the ocean or some fixed location) and swim upstream to access their historical habitats. At a barrier, a randomly selected fish can pass the barrier with a certain probability, which is called the *passage probability* of the barrier. A river network is treated as a tree as justified by [65], that is, two different locations in the river is connected by a unique path. So, the probability that a fish can reach a designated location in a river from the root is the probability that the fish can pass all barriers on the unique path from the root to that location. To make the model simple, we assume that the passage of one barrier is independently distributed from the passage of any other barrier. The probability equals to the product of the passage probabilities of all barriers on the path. A barrier can be removed or repaired with a certain cost to increase its passage probability. For example, the passage probability of an old culvert is 0.5, meaning that with 50% chance a randomly picked fish, which, for example, is mature and has strong fins, can swim upstream to pass the culvert. If a new type of tunnel is built to replace the old culvert, some premature fish can pass the tunnel and reach upstream areas so that the passage probability may become 80%. Fish, after reaching a stream segment, can consume the food and use the natural resource available in that region. One stream segment can only support a certain number of fish to live there, so we say that each stream segment has a certain amount of habitat that the fish can use. The barrier removal problem is to find the

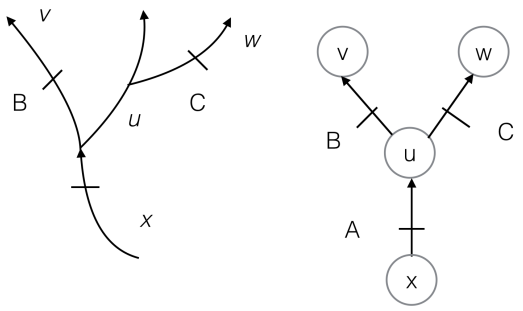


Figure 2.3: Left: a river network with barriers A, B, C and contiguous regions u, v, w, x. Right: corresponding bidirected stochastic tree.

best set of barriers to repair or remove to maximize the expected amount of habitat that fish can access.

The barrier removal problem can be formulated as a stochastic network design problem. As shown in Fig. 2.3, a river network (left) can be modeled as a directed rooted tree (right) denoted by $\mathcal{T} = (V, E)$ with a unique $root$. Each node $v \in V$ represents a contiguous region of the river network—i.e., a connected set of stream segments among which fish can move freely without passing any barriers—and the reward $r(root, v)$ equals to the total amount of habitat in that region (e.g., the total length of all segments if the amount of habitat in a segment is proportional to its length). Thus, the total habitat that fish can use equals to the total reward collected at nodes that are connected to $root$. Each barrier is represented by a directed edge that connects two regions for which p^o_{uv} is set to be the passage probability before the repair and p^m_{uv} is set to be the passage probability after the repair. c_{uv} is the cost of repairing the barrier on the edge (u, v). Multiple alternative actions can be defined for each edge as discussed in section 2.4. In this way, the barrier removal problem is written as a stochastic network design problem.

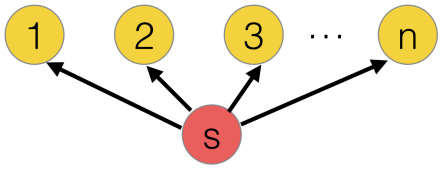


Figure 2.4: The instance of the stochastic network design framework representing the Knapsack problem

2.3 The Hardness of Stochastic Network Design

In general, a stochastic network design problem is NP-hard. This can be proved by showing that an instance of the Knapsack problem, which is a well-known NP-hard problem, can be written as a stochastic network design problem.

Theorem 2.3.1. *The stochastic network design problem, in general, is NP-hard.*

Proof. In the Knapsack problem, we are given n items, each of which has a weight and a value. Given a bag of capacity $\mathcal{B}$, the goal is to determine which items to pack to maximize the total value being packed while the total weight of packed items is no greater than the capacity of the bag. Mathematically, the Knapsack problem can be written as the following maximization problem.

$$\max \sum_i v_i \cdot \mathbb{I}(i \text{ is packed}) \qquad s.t. \quad \sum_i w_i \cdot \mathbb{I}(i \text{ is packed}) \leq \mathcal{B}$$

where v_i and w_i are the value and the weight of the i^{th} item and $\mathbb{I}(\cdot)$ is an indicator function that reflects how items are packed.

A Knapsack problem can be written as a stochastic network design problem shown in Fig. 2.4. In the network, V consists of n nodes $\{1, 2, ..., n\}$, each of which represents an item to be packed, and an extra node s. We set $p^o_{si} = 0$ and $p^m_{si} = 1$ for each node $i \in \{1, 2, ..., n\}$ and set the cost of the action on (s, i) as w_i. Also, we set the reward $r(s, i)$ to be v_i for each node $i \in \{1, 2, ..., n\}$ and the reward of other node pairs to be 0. Now, if we take the action on the edge (s, i), it is equivalent that we pack the item i.

Therefore, a set of actions taken maps to a set of items packed. The total value of packed items equals to the expected total reward. The total weight is equivalent to the cost of the policy. Therefore, two problems are equivalent. Since we know the Knapsack problem is NP-hard, the theorem holds. □

2.4 Extensions

In this section, several ways to extend the stochastic network design framework are discussed. For the ease of presentation, most of the algorithms in the thesis are presented in the basic setting of the stochastic network design framework discussed in section 2.1. These algorithms are also compatible with one or multiple these extensions. It is a future work to develop scalable algorithms to solve those stochastic network design problems with extensions of which no algorithms are given in the thesis.

2.4.1 Correlated Survival Probability

In the basic setting, the state of an edge is randomly distributed and is independent of the states of other edges, but, in a real-world problem, the states of multiple edges can be correlated. For example, in a pre-disaster preparation problem, an edge represents a road segment, and the survival probability represents the chance that the road segment will be damaged by a natural disaster such as a flood. It is very likelihood that when one road is damaged, all nearby roads are under the effect of the disaster too, implying that all these edges may fail simultaneously. To model this situation, the survival probabilities are defined jointly by all these edges instead of for each single edge. To draw a sample network from a correlated probability distribution, the states of multiple edges are sampled simultaneously from a joint probability distribution instead of independently from a Bernoulli distribution. For stochastic network design problems with correlated survival probabilities, the sampling based algorithms discussed in Chapter 5 and 6 are still applicable.

2.4.2 Effect of an Action on Multiple Edges

In the basic setting, each edge is associated with a candidate action, so the total number of actions equals to the number of edges. In reality, one action may change the survival probabilities of multiple edges. For the influence maximization problem, rewards are provided to a person (e.g., a node) to increase the chance that the person will share the information with her/his friends. The reward can target on nodes or edges. If rewards are given to a node, the survival probabilities of all outgoing edges can increase. For example, if rewards are given to encourage a person to share the information of a product to as many friends as possible, the chance of any friend of this person to adopt the product will increase. For a variant of the barrier removal problem, introduced in section 2.2.2, called *bidirected barrier removal problem* [99], fish don't start from a unique entrance, such as the ocean, but from multiple locations in the river network. These fish may pass a barrier from either upstream direction or downstream direction, each of which is modeled by a directed edge and has a different passage probabilities because of the difficulties of passing a barrier from two directions are different. Since the two directions are considered, an action will modify the survival probabilities of both edges (u, v) and (v, u) simultaneously. An algorithm to solve the *bidirected barrier removal problem* is given in Chapter 4. The algorithms in Chapter 5 and 6 that are based sampling methods are compatible with this extension too.

2.4.3 Multiple Candidate Actions

In the basic setting, it is assumed that one edge has only one candidate action, if taken, changing the survival probability from p^o to p^m. Sometime, it is required to have more than one candidate actions, each of which has a different cost and a different effect. For the barrier removal problem, there may be multiple ways, or multiple actions, to repair a barrier, each of which can raise the passage probability into a different level. To repair a dam, we may have two candidate actions. One is to remove the dam completely, which

is expensive but very effective such as raising the passage probability from 0.2 to 1.0. The other is to build a small fish passage tunnel, which is less expensive but may only be helpful to mature fish such as raising the passage probability from 0.2 to 0.5. More reparing strategies can be defined and set as candidate actions. The algorithms introduced in Chapter 3 and 4 are compatible with this extensions.

2.5 Quota, Budget, and Prize-collecting Problems

The goal of the stochastic network design framework defined by (2.5) is to find the policy to maximize the resilience function and also to satisfy a budget constraint. This type of problem is the *budget* problem. In contrast, two variant optimization problems of a budget problem are the quota problem and the prize-collecting problem, both of which have been widely studied in the community of approximation algorithm design [41]. Especially, many typical network design problems are quota problems. In a quota problem, the goal is to minimize the design cost to make a policy satisfy some constraint, which, in some sense, switches the budget constraint and the objective of a budget problem. An example problem is the Steiner tree problem [37], in which a policy needs to connect all terminal nodes and the goal is to minimize the design cost. A quota variant of a stochastic network design problem is to find a policy to minimize the cost and make the value of the policy equal or greater than a given value. Mathematically, it is written as

$$\min_{\pi \subset \mathbb{A}} \ cost(\pi) \quad s.t. \quad R(\pi) \geq value \tag{2.6}$$

All requirements that we want the policy to satisfy is summarized by a single inequality in the constraint in the quota problem. The left side of the inequality involves an expectation that makes the optimization problem very difficulty to solve. It is an open question on how to design the scalable algorithms to solve the quota variant of a stochastic network design problem.

A prize-collecting variant of a stochastic network design problem is to find a policy to maximize the difference between the value and the cost of the policy. Mathematically, it is written as

$$\max_{\pi \subset \mathbb{A}} \; R(\pi) - \beta \cdot cost(\pi) \tag{2.7}$$

The subtraction of the cost from the value is one kind of the penalty that we get while maximizing the value of the policy. In a stochastic network design problem, with more management actions, the value of the policy will increases or remain the same, but cost of the policy will also increase. The parameter β, which is a part of the problem definition, controls our tradeoff between the value and the cost. The larger the value of β, the more penalty we will get by taking more actions. A well-defined β helps people identify a cost-efficient policy so that more actions added to the policy can only increase the value very little and not worthy investing to. The prize-collecting Steiner tree problem and the algorithms to solve it are given in the book [96].

A prize-collecting problem appears to be easier to solve than its budget variant for many network design problems [39, 96]. For example, the facility location problem is one type of prize-collecting problem [39]. Its budget variant is the so-called k-median problem that is more difficulty to solve than the facility location problem.

2.5.1 Relationship between Budget and Prize-collecting Problems

A budget problem can be rewritten as a prize-collecting problem by taking its Lagrangian relaxation, that is, moving the budget constraint into the objective along with a parameter β called the Lagrangian multiplier. A budget problem is written as

$$\max_{\pi \subset \mathbb{A}} \min_{\beta} \; R(\pi) - \beta \cdot (cost(\pi) - \mathcal{B}) \tag{2.8}$$

If a policy violates the budget contraint or $cost(\pi) - \mathcal{B} > 0$, the inner minimization makes the objective negative infinity. For a policy that satisfie the contraint, the objective remain

bounded as long as value of any policy in the space is bounded. Therefore, problem (2.8) is equivalent to the stochastic network design problem defined by (2.5).

$\mathcal{B}$ is irrelavant to the policy and can be removed from (2.8). If we fix the value of β and eliminate the inner minimization problem, the problem becomes a prize-collecting problem if we fixed β. Algorithms have been developed to solve a budget problem by searching a good value of β and solving prize-collecting problems that are created by above procedure [39]. In Chapter 5 and 6, the same technique is used to solve stochastic network design problems in which a sequence of prize-collecting stochastic network design problems are solved.

2.6 Summary

This chapter gives the mathematical definition of the stochastic network design framework, which is defined under a general context and can model a range of decision-making problems. For illustrative purposes, it is explained how to formulate two real-world problems using the framework. A stochastic network design problem is one type of budget problem in which the goal is to find a policy to maximize the value of the policy that satisfies a budget constraint. The correspondent quota problem and prize-collecting problem are briefly discussed.

CHAPTER 3

STOCHASTIC NETWORK DESIGN FOR DIRECTED ROOTED TREES

This chapter and the next chapter introduce the algorithms for stochastic network design problems for trees. The algorithms are developed separately for two types of trees: the *directed rooted tree* and the *bidirected tree*. A *directed rooted tree* also called *arborescence* [1] is a directed graph with a unique node called $root$ in which there is a unique directed path from the root to every other node. A *bidirected tree* can be obtained from an undirected by converting each edge in the undirected tree into two directed edges with reversed directions. In this chapter, I introduce the stochastic network design problem for directed rooted trees and a scalable algorithm to solve it. The algorithm for bidirected trees is discussed in the next chapter. The structure of this chapter is as follows. First, the formal definition of the problem is given. A dynamic programming algorithm is given to solve the problem optimally, but the runtime complexity of the algorithm is exponential of the number of nodes in the tree. A rounding strategy is used to reduced the complexity of the algorithm but is still able to produce nearly optimal solutions. Then, experimental results are provided. At last, the quota and the prize-collecting versions of the problem are discussed.

3.1 Problem Statement

A *directed rooted tree* is a directed graph denoted by $\mathcal{T} = (V, E)$ with a unique node called $root \in V$. For every other node $v \in V - \{root\}$, there is a unique directed path from $root$ to v. An example is shown in Fig. 3.1

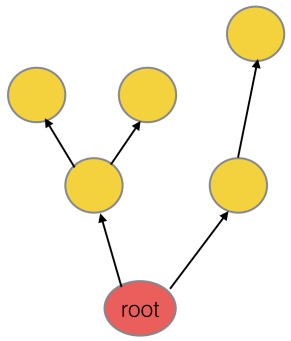


Figure 3.1: A directed rooted tree

The directed rooted tree can model a phenomenon or a process that spreads out from the root into the network. For example, the information spreads in a hierarchical structure where the root represents the information source, each level of the tree represents a level in the hierarchy and nodes pass information to their subordinates. Another example is the barrier removal problem that I introduce in section 2.2.2 where the river network is considered as a tree and all fish spread into the river network from the unique entrance of the network (e.g., the root).

Now, I will quickly remind you the barrier removal problem and how it is formulated as a stochastic network problem. In the barrier removal problem, each node represents a contiguous region in the river. The reward $r(root, v)$ represents the size of habitat that can be used by fish at a node v. Since all fish start from the root and spread into the network, so the reward for any other pair (u, v) with $u \neq root$ is 0. For simplicity, we denote the reward of a node as $r_v = r(root, v)$. The expected total reward defined by (2.4) is the sum of reward that we can collect at each node. Mathematically, the stochastic network design problem for rooted trees can be written as

$$\max_{\pi} R(\pi) = \sum_{v \in V} Pr(root \rightsquigarrow v; \pi) \cdot r_v \quad s.t. \quad cost(\pi) \leq \mathcal{B} \tag{3.1}$$

where $Pr(root \rightsquigarrow v)$ is the probability that all edges on the unique path from $root$ to v are present. Since we have a unique path from $root$ to v, we have

$$Pr(root \rightsquigarrow v; \pi) = \prod_{e \text{ on path from } root \text{ to } v} p(e; \pi)$$

The basic setting of the stochastic network design framework only defines one action for each edge, that is, the action can raise the survival probability from p^o_{uv} to p^m_{uv}. The algorithms introduced in this chapter and the next chapter are compatible with multiple candidate actions, so the problem definition is changed a little to accomodate multiple candidate actions. At each edge (u, v), there are a finite number of candidate actions A_{uv}. An action $a \in A_{uv}$ takes cost $c_{uv,a}$ to make the survival probability to be $p_{uv|a}$ or written as $p_{uv|\pi}$ for a given policy π.

Without loss of generality, I make the following assumption

Assumption 3.1.1. *Each node u in $\mathcal{T}$ has at most two children.*

Any problem instance can be converted to this form by replacing a node u with more than two children by a sequence of nodes with exactly two children to achieve the same overall branching factor so that a policy for the modified tree can be mapped to a unique policy for the original tree with the same expected reward.

For the purpose of presentation, $P_{u \rightsquigarrow v|\pi}$ denotes the probability that all edges on the unique path from u to v are present under π. Since the tree is a rooted tree, a subtree $\mathcal{T}_u$ can be defined at each node u. Let $z_u(\pi)$ denote the expected total reward (also called value for short) that we can collect within subtree $\mathcal{T}_u$ under a policy π. That is, $z_u(\pi) = \sum_{v \in \mathcal{T}_u} P_{u \rightsquigarrow v|\pi} r_u$, and $R(\pi) = z_{root}(\pi)$.

The basic idea of the algorithm to solve the problem (3.1) is as follows. First, a pseudo-polynomial time dynamic programming algorithm is developed to compute the optimal solutions. Then, a rounding strategy is developed to make the algorithm an fully polynomial-time approximation scheme (FPTAS). Let's see first how to evaluate a policy recursively.

3.2 Policy Evaluation

The value of each subtree can be calculated recursively from leaf nodes to the root by a simple recurrence

$$z_u(\pi) = p_{uv|\pi} z_v(\pi) + p_{uw|\pi} z_v(\pi) + r_u \tag{3.2}$$

where v and w are two children of u. At the root node, we have the expected total reward of the whole tree.

3.3 Dynamic Programming Algorithm

The dynamic programming (DP) algorithm computes a dynamic programming table for each subtree recursively from leaf nodes to the root of the tree. Let subpolicy π_u be the part of the full policy that defines actions within $\mathcal{T}_u$. The DP table of each subtree $\mathcal{T}_u$ contains a list of z values that are reachable by some subpolicies, and each value is associated with a least-cost subpolicy, that is, each z in the table is associated with the policy $\pi_u^* \in \arg\min_{\{\pi_u | z_u(\pi_u)=z\}} c(\pi_u)$ where $c(\pi_u)$ represents the cost of a subpolicy π_u.

We recursively generate the list of reachable z values for subtree u and the associated least-cost subpolicies using the z values and their associated subpolicies in the tables of subtree v and subtree w. To do this, for each z_v, z_w, we first extract the corresponding π_v^* and π_w^*. Then, using these two least-cost subpolicies of the children, for each $a \in A_{uv}$ and $a' \in A_{uw}$, a new subpolicy π_u is constructed for $\mathcal{T}_u$ with cost $c(\pi_u) = c_{uv,a} + c_{uw,a'} + c(\pi_v^*) + c(\pi_w^*)$.

Using Eqs. (3.2), the value $z_u(\pi_u)$ of π_u is calculated. If $z_u(\pi_u)$ already exists in the list (i.e., $z_u(\pi_u)$ was created by some other previously constructed subpolicies), we update the associated subpolicy such that only the minimum cost subpolicy is kept. If not, we add this new value $z_u(\pi_u)$ and subpolicy π_u into the list.

To initialize the recurrence, the table of a leaf subtree contains only a single value of $z_u = r_u$ associated with an empty subpolicy. Once the table of $\mathcal{T}_{root}$ is calculated, we scan

the table to pick a pair (z^*_{root}, π^*) such that $z^*_{root} = \max_{\{(z_{root},\pi)|c(\pi)\leq b\}} z_{root}$. Finally, π^* is the returned optimal policy and z^*_{root} is the optimal expected reward.

3.4 Rounded Dynamic Programming

As shown in Theorem 2.4, a stochastic network design problem is NP-hard even for a directed rooted tree. The DP algorithm is not a polynomial-time algorithm because the number of reachable values of z increases exponentially as we approach the $root$, but it can be made into an FPTAS algorithm with a rounding strategy. The basic idea is to discretize the continuous space of z_u at each subtree such that there only exists a polynomial number of different values. To do this, the one-dimensional value space of z_u for $\mathcal{T}_u$ is discretized by a granularity factor K^z_u into a finite number of values $\{0, K^z_u, 2 * K^z_u, ...\}$ in which the difference between any two subsequent values is K^z_u.

For any subpolicy π_u of subtree u, a rounded value $\hat{z}_u(\pi_u)$ in the discretized space is used to underestimate the true value $z_u(\pi_u)$ of π_u. To evaluate $\hat{z}_u(\pi_u)$, we use a similar recurrence as (3.2), but rounding each intermediate value into a value in the discretized space. The new recurrence is as follow:

$$\hat{z}_u(\pi_u) = K^z_u \left\lfloor \frac{p_{uv|\pi}\hat{z}_v(\pi) + p_{uw|\pi}\hat{z}_v(\pi) + r_u}{K^z_u} \right\rfloor \tag{3.3}$$

The modified algorithm—rounded dynamic programming (RDP)—is the same as the DP algorithm, except that it works in the discretized space. Specifically, each node maintains a list of reachable rounded values $\hat{z}_u$s, each one associated with a least costly subpolicy that can achieve $\hat{z}_u$ by formula (3.3), that is, $\pi^*_u \in \arg\min_{\{\pi_u|\hat{z}_u(\pi_u)=\hat{z}_u\}} c(\pi_u)$. Similar to the DP algorithm, at each subtree $\mathcal{T}_u$, we generate the list of reachable rounded values using the lists of rounded values of its children. The difference is that to calculate the rounded values of a new subpolicy we use the recurrence (3.3).

3.5 Theoretical Analysis of the Algorithm

This section provides the theoretical analysis of the performance of the RDP algorithm. The main result is as follows.

Theorem 3.5.1. *The Rounded Dynamic Programming (RDP) algorithm is an FPTAS. Specifically, let OPT be the value of the optimal policy. By assigning the scaling factors* $\{K_v\}$ *in a certain way (described below), the RDP algorithm computes a policy with value at least* $(1-\epsilon)$*OPT and runs in time* $O(\frac{n^2}{\epsilon^2})$ *in the worst case.*

The remaining of this section proves Theorem 3.5.1 by showing the approximate guarantee and analyzing the running time.

3.5.1 Approximation Guarantee

Let π^* be the optimal policy and let π' be the policy returned by RDP. We wish to bound the value loss $z(\pi^*) - z(\pi')$.

The idea is to first bound the difference between the true objective value $z(\pi)$ and the RDP objective value $\hat{z}(\pi)$ for an arbitrary policy π, which can be done by analyzing the error incurred by the rounding operations in the recurrence of Eq. 3.3. By showing that the rounded objective function $\hat{z}$ is uniformly close to z for all policies π, it is straightforward to show that optimizing with respect to $\hat{z}$ provides a nearly-optimal policy with respect to z.

To analyze the error introduced by rounding, fix a policy π and let $\Delta_u(\pi) \in [0, 1)$ be the fractional part of the quantity that is rounded in Eq. 3.3, so that $K_u\Delta_u(\pi)$ is the total loss due to rounding when computing the recurrence for node u. Then it is straightforward to show that the total error is equal to the sum of the rounding errors at each node u weighted by the accessibility of the parent of node u under policy π.

Lemma 3.5.1. *For any policy* π*, the difference between the original and rounded objective functions is*

$$z(\pi) - \hat{z}(\pi) = \sum_{u \in V} Pr(root \rightsquigarrow u; \pi) K_u \Delta_u(\pi) \tag{3.4}$$

Now, to bound the optimality gap $z(\pi^*) - z(\pi')$, note that $z(\pi') \geq \hat{z}(\pi') \geq \hat{z}(\pi^*)$, where the first inequality holds because the rounded policy value always underestimates the true policy value, and the second inequality holds because π' is optimal with respect to $\hat{z}$. Thus we have

$$z(\pi^*) - z(\pi') \leq z(\pi^*) - \hat{z}(\pi^*), \tag{3.5}$$

so it suffices to bound the gap between the original and rounded objective on the optimal policy π^* using Lemma 3.5.1. We have

$$z(\pi^*) - z(\pi') \leq \sum_{u \in V} Pr(root \rightsquigarrow u; \pi^*) K_u \Delta_u(\pi^*) \tag{3.6}$$

Lemma 3.5.2. *The RDP policy π' has value at least $(1-\epsilon)$OPT if the following condition on the scaling factors $\{K_u\}$ holds:*

$$\sum_{u \in V} Pr(root \rightsquigarrow u; \pi^*) K_u \leq \epsilon z(\pi^*) \tag{3.7}$$

Proof. The left side of Eq. (3.7) is an upper bound on $z(\pi^*) - z(\pi')$ because of (3.6) and $\Delta_u(\pi^*) \leq 1$. Then, we have $z(\pi') \geq (1-\epsilon) z(\pi^*)$ which proves the lemma. □

Lemma 3.5.2 is useful as a generic condition on the scaling factors $\{K_u\}$ for obtaining a $(1-\epsilon)$-optimal policy. There are different ways of setting the values so Eq. (3.7) is satisfied, and the particular choice will affect the running time of the algorithm. Indeed, note that a

larger value of K_u leads to a coarser discretization of the value space at node u, and thus the RDP algorithm will take less time to evaluate the recurrence for all discretized values. Thus, in practice, we would like to set the scaling factors as large as possible while still satisfying Eq. (3.7). We first present a particular way of setting the values that are rather coarse but lets us prove both the approximation guarantee and the worst-case running-time bound. In Section 5 we discuss practical improvements.

Lemma 3.5.3. *Setting $K_u = \epsilon r_u$, the policy π' returned by RDP is $(1-\epsilon)$-optimal policy.*

Proof. Setting $K_u = \epsilon r_u$, we have

$$\sum_{u \in V} Pr(root \rightsquigarrow u; \pi^*) K_u \leq \epsilon \sum_{u \in V} Pr(root \rightsquigarrow u; \pi^*) r_u = \epsilon z(\pi^*)$$

By Lemma 3.5.2, we prove the lemma. □

3.5.2 Runtime Analysis

Now, we show that the running time of the RDP algorithm is $O(\frac{n^2}{\epsilon^2})$ in the worst case if the scaling factors K_u are assigned as in Lemma 3.5.3. First, it is reasonable to assume that the rewards are constant with respect to n.

Assumption 3.5.1. *There are universal constants m and M such that $m \leq r_u \leq M$ for all $u \in V$.*

As shown later in the barrier removal problem, if the rewards represent the lengths of accessible stream segments, these should not vary with the size of the stream network being modeled.

Let ℓ_u denote the number of discretized values at subtree u. We have the following lemma.

Lemma 3.5.4. *With Assumption 3.5.1 and setting K_u as in Lemma 3.5.3, the number of discretized values at node u is bounded by $O(n_u/\epsilon)$, where n_u is the number of nodes in $\mathcal{T}_u$.*

Proof. The lemma is a direct consequence of the definitions we have made. Let $\text{UB}_v = \sum_{u \in \mathcal{T}_v} r_v$. Then

$$\ell_u = \left\lceil \frac{\text{UB}_u}{K_u} \right\rceil \leq \left\lceil \frac{n_u M}{K_u} \right\rceil \leq \left\lceil \frac{n_u M}{\epsilon m} \right\rceil = O\Big(\frac{n_u}{\epsilon}\Big).$$

□

We are ready to prove the following result.

Theorem 3.5.2. *By setting $K_u = \epsilon r_u$, the running time of RDP is $O(\frac{n^2}{\epsilon^2})$.*

Proof. Let's examine the time spent in computing the recurrence for a single node u. To compute the list of rounded values for subtree u, we first need to compute the lists of rounded values for two children subtrees and then enumerate each pair of rounded values in the lists of two children subtrees and each pair of actions at two edges. Let $T(n_u)$ be the running time for subtree u where n_u is the number of nodes in $\mathcal{T}_u$. Let v, w be two children of u and n_v and n_w be the numbers of nodes in $\mathcal{T}_v$ and $\mathcal{T}_w$ with $n_u = n_v + n_w$. Then, we have the recurrence

$$T(n_u) = O(\frac{n_v \cdot n_w}{\epsilon^2}) + T(n_v) + T(n_w) \tag{3.8}$$

where the number of available actions at each edge is assumed to be bounded by a constant.

It is ready to show that $T(n_u) = O(\frac{n_u^2}{\epsilon^2})$. First, consider a slightly different function $T'(n_u)$ for subtree u with the recurrence as follows.

$$T'(n_u) = O(n_v \cdot n_w) + T'(n_v) + T'(n_w) \tag{3.9}$$

It can be shown that $T'(n_u) = O(n_u^2)$, which helps to show $T(n_u) = T'(n_u)/\epsilon^2 = O(n_u^2/\epsilon^2)$.

To show $T'(n_u) = O(n_u^2)$, the induction method is used. The base case is $T'(1) = O(1)$ for any leaf node. Assume $T'(n_v) = O(n_v^2)$ and $T'(n_w) = O(n_w^2)$. By (3.9), we have

$$\begin{aligned} T'(n_u) &= c_1 \cdot 2 \cdot n_v n_w + c_2 n_v^2 + c_3 n_w^2 \\ &\leq c(2 \cdot n_v n_w + n_v^2 + n_w^2) \leq c(n_v + n_w)^2 = c n_u^2 \end{aligned}$$

where c_1, c_2, c_3 are three constants and $c = \max\{c_1, c_2, c_3\}$. Therefore, $T'(n_u) = O(n_u^2)$ for any node u.

It is easy to show by induction that $T(n_u) = T'(n_u)/\epsilon^2$, so we have $T(n_u) = \frac{n_u^2}{\epsilon^2}$, meaning that the running time for a rooted tree with n nodes is $O(\frac{n^2}{\epsilon^2})$, which proves the theorem. □

3.6 Implementation

Theoretically, the upper bound of the runtime of RDP is quadratic of the size of the underlying tree. This theoretical upper bound is very large when the tree has millions of nodes. In this section, I discuss several techniques to improve the empirical runtime.

3.6.1 Setting the Scaling Factors

To prove the FPTAS, it suffices to set $K_u = \epsilon r_u$ for all u. However, *in practice*, we observe that setting the scaling factors to a constant value α that is larger than ϵr_u still finds near-optimal policies and runs much faster. The reason for the observation is given in paper [98].

3.6.2 Detecting Infeasible Policies

The algorithm can be made faster by exploiting the budget limit b. The idea is to ignore pairs $(\hat{z}_v, \hat{z}_w)$ in the RDP recurrence when the cost to obtain either $\hat{z}_v$ or $\hat{z}_w$ in the subtree already exceeds b, because these will lead to infeasible policies. This technique speeds up

the algorithm especially when the budget limit is small. In our experiments, we observe that even for the subtrees at intermediate depths, a large percentage of values are pruned.

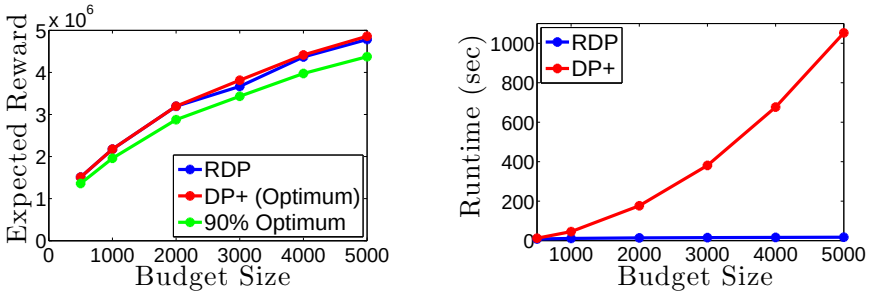
3.7 Experiments

The RDP algorithm is applied to solve the barrier removal problem introduced in Section 2.2. In our experiments, we use data from the CAPS project [58] for the Connecticut River watershed in Massachusetts (shown in red in Fig. 2.2), which has 18550 nodes including 596 dams and 7566 crossings that include different types of small barriers. We assigned passage probabilities to dams and road-stream crossings based on techniques developed in the CAPS project. For dams, the structural height of the dam is a proxy for passability, which maps through a logistic function to a probability value. A subset of road-stream crossings was directly assessed by a field protocol; The remaining crossings were assigned passage scores based on a fitted predictive model. Passage scores were then transformed to probabilities, resulting in a typical range of $[0.7, 1.0]$ for road-crossings and $[0, 0.15]$ for dams.

Using this dataset, we compared our RDP algorithm with the dynamic programming algorithm, called DP+, of [65], which assumes that the costs of actions and the budget b are integral values. DP+ is optimal under this assumption. However, unlike RDP, DP+ is not scalable to large action costs. Therefore, to perform the evaluation we used relatively small integral costs.

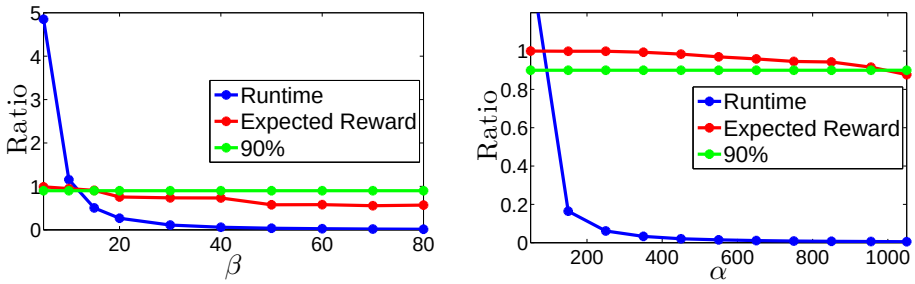
For road-crossings, most of the probabilities are close to 1 to start with, and relatively cheap actions can be taken to clear out the crossing completely. For example, we use $A_{\{(u,v)\}} = \{a_1\}$ with $(p_{uv|a_1} = 1.0, c_{uv|a_1} = 20)$. In contrast, it is relatively difficult and expensive to remove dams completely, so multiple strategies must be considered to improve the passability of dams. For example, we may have $A_{\{(u,v)\}} = \{a_1, a_2, a_3\}$ with $(p_{uv|a_1} = 0.2, c_{uv|a_1} = 20)$, $(p_{uv|a_2} = 0.5, c_{uv|a_2} = 40)$ and $(p_{uv|a_3} = 1.0, c_{uv|a_3} = 100)$.

Now, we present some experimental results.



(a) Optimality w.r.t. budget (b) Runtime w.r.t. budget

Figure 3.2: Solution quality and runtime for different budgets



(a) Time & value for method (1) (b) Time & value for method (2)

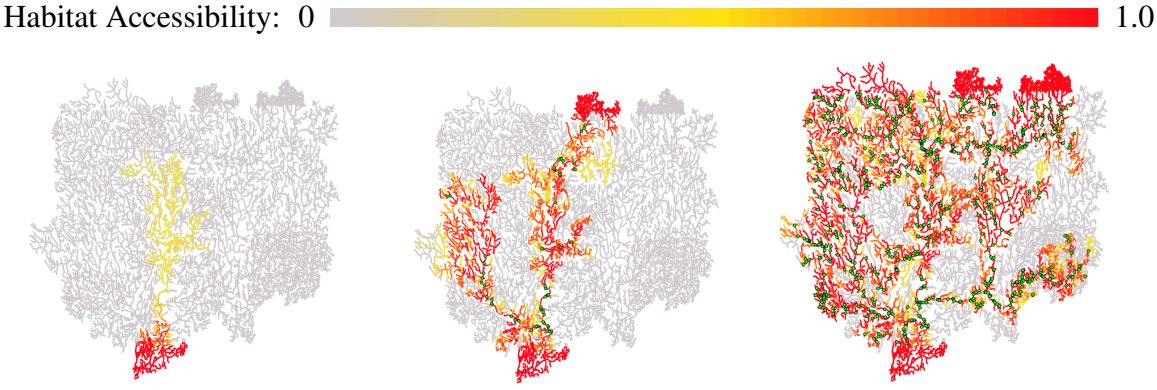
Figure 3.3: Impact of different methods to set K_u

3.7.1 Approximation Quality

Fig. 3.2(a) shows the expected reward of the computed policy for different budgets, as well as OPT (the value of the optimal policy) and 90%OPT. In these experiments, we set all K_u to be a constant $\alpha = 450$. We see that the actually expected reward of the computed policy is very close to the optimal value and guaranteed to be greater than the 90%OPT .

3.7.2 Runtime

Fig. 3.2(b) shows the computation time of our algorithm (with the optimization of detecting infeasible policies) compared with DP+ over a range of budget sizes. We see that RDP runs much faster than the optimal algorithm DP+. Moreover, the runtime of DP+ increases quadratically with the budget size while RDP's runtime remains essentially con-



(a) budget = 0, $z = 5.4 \times 10^5$ (b) budget = 1000, $z = 2.2 \times 10^6$ (c) budget = 10000, $z = 5.8 \times 10^6$

Figure 3.4: Visualization of several barrier removal policies

stant, even for very large budget sizes. For example, when the budget size is 5000, DP+ takes about 20 minutes while RDP takes only 20 seconds.

Fig. 3.2(a) and 3.2(b) together show that by setting $K_u = 450$, RDP runs much faster than DP and produces a near-optimal policy (within 90%).

3.7.3 Different Settings of K_u

We compared the two different ways of setting K_u as discussed in Section 3.6: (1) setting $K_u = \beta K'_u$ where K'_u are the values specified in Lemma 4, and (2) $K_u = \alpha$. We used the constants α and β to relax the optimality guarantee and study the effect on runtime/quality (Fig. 3.3). The runtime ratio is the ratio of RDP's runtime to DP+'s runtime. The value ratio is the ratio of the expected reward of the computed policy obtained by RDP to the optimal expected reward obtained by DP+. We found that for method (1) that before the expected reward becomes worse than 90% of optimal, RDP takes more time than DP+. For method (2), as α increases up to 950, the quality of the policy remains above 90%, but the runtime ratio is less than 0.1. This matches the intuitive explanation provided in Section 5.

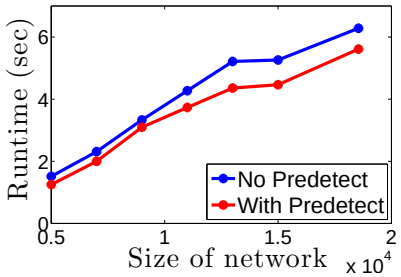


Figure 3.5: RDP's runtime achieving 90% optimality

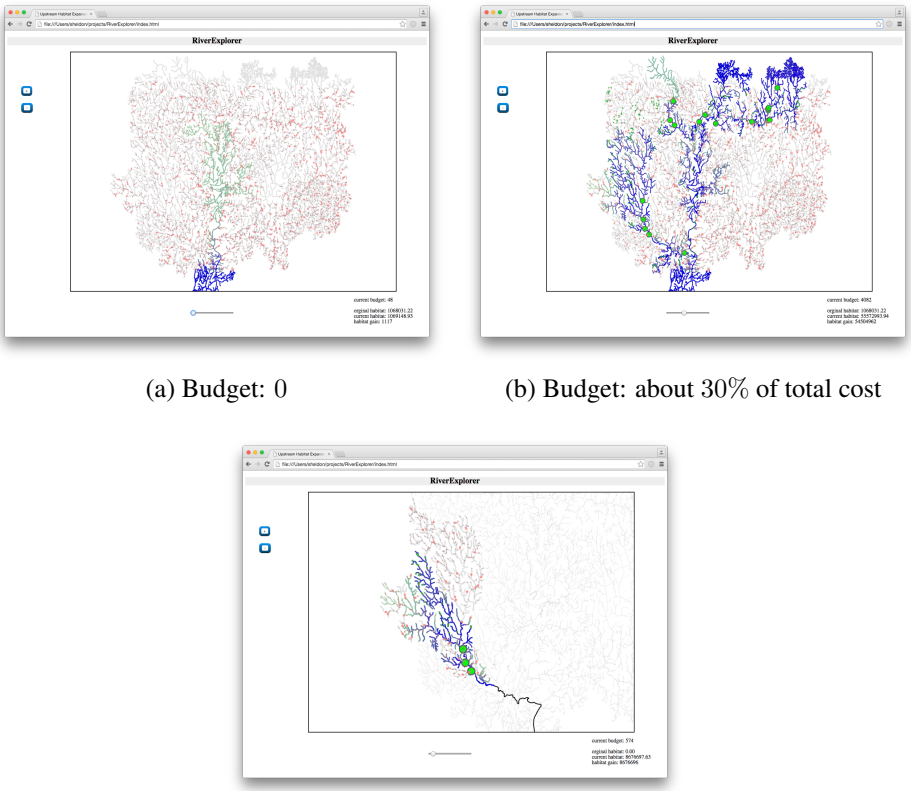
3.7.4 Runtime Curve

Theoretically, we proved that in the worse case the complexity of RDP is $O(\frac{n^2}{\epsilon^2})$. However, in practice, we can get a better computation time by using the techniques described in Section 3.6. By applying the technique of detecting infeasible policies, runtime was reduced by at least 20% over a range of budget sizes. When the budget size is small, many computations can be pruned and runtime is reduced by up to 55%. Moreover, as just shown, the value of K_u can be selected in a better way to further reduce the computation time dramatically. Fig. 3.5 shows the minimum time needed to produce a 90% optimal policy as a function of network size. Subnetworks of different sizes were extracted from the original network for these experiments. The minimum time is obtained by applying method (2) and choosing the largest constant α that produces the desired quality. Surprisingly, both curves—with and without detecting the infeasible policies—are nearly linear except for some small fluctuation in the middle.

3.7.5 Visualizing Policies

To give some sense of how the policies improve the ability of fish to access their habitats, Fig. 3.4 illustrates the accessibility of each stream segment by a distinct color from the color bar at the top. Barriers that are repaired by some action are designated by green circles, regardless of the specific repair action. The budget in (c) is 10 times larger than in (b), leading to a substantial increase in the number of repaired crossings as they are much

cheaper to repair compared to dams, and resulting in significantly better overall accessibility.



(a) Budget: 0

(b) Budget: about 30% of total cost

(c) Investment in small subtree

Figure 3.6: A tool called river explorer built based on the RDP algorithm. The stream with darker color has higher reachability probability. The green dots represent removed/repaired barriers

3.8 Interactive Decision Making

The rounded dynamic programming algorithm can also help a decision maker to determine how much budget to invest. In most of the cases, a decision maker doesn't know clearly how much budget is needed and how effective a certain amount of budget is. Before making the decision, she/he may want to try different budget sizes and pick the best one. The idea of interactive decision making, as described below, can help the decision maker to find the ideal budget size. First, the decision maker sets a budget size. Second, the value

that can be achieved with this budget size is computed by the algorithm and the (near) optimal policy is visualized. Based on the results, the decision maker learn how effective is this budget size. Third, the decision maker changes the budget size and goes back to the second step to see how much improvement (loss) the little change to the budget incurs to the final results. Over some iterations, the decision maker can gradually learn what budget size is ideal. The process may make many iterations, so the ability to solve a stochastic network design problem quickly is critical.

The rounded dynamic programming algorithm is ideal for the interactive decision making. By running RDP once, the dynamic programming tables are computed at each node from which we can easily check the value that can be achieved for different amounts of budget without solving the optimization problem again. Sometimes, we only want to decide the ideal budget size of a small region (e.g., a subtree) of the whole network. Since the tables are available at each subtree, the outcome can be learned by simplely look at the table. Fig. 3.6 shows how RDP is convenient to the interactive decision making. The figure shows several screenshots of a tool called river explorer [20] built based on the RDP algorithm. As demonstrated by Fig. 3.6(a,b), the decision maker can change the budget size (e.g., from 0 to 30% of the total cost) and quickly visualize how much habitat fish can reach and which barriers are removed/repaired. Also, the decision maker can only focus on a subtree and play with different budget sizes shown in Fig. 3.6(c).

3.9 Quota and Prize-collecting Problem for Directed Rooted Tree

As mentioned in Section 2.5, a stochastic network design problem has a correspondent quota problem and a prize-collecting problem.

A prize-collecting problem appears easier to solve than the budget problem for many network design problems [96]. The prize-collecting stochastic network design problem under a directed rooted tree is defined by

$$\max_{\pi} \sum_{v \in V} Pr(root \rightsquigarrow v) \cdot r_v - \beta \cdot cost(\pi) \tag{3.10}$$

in which both the value and the cost of a policy–the penalty for a taken policy–are in the objective function along with a tradeoff parameter β with $\beta \geq 0$. If the value of β is 0, the unconstrained optimization problem can be solved by a simple message passing algorithm from leaf nodes to the root, which takes polynomial time. If the value of β is greater than zero, the hardness of the problem remains to be an open question.

The quota stochastic network design problem under a directed rooted tree is defined by

$$\min_{\pi} \ cost(\pi) \ \ s.t. \ \sum_{v \in V} Pr(root \rightsquigarrow v) \cdot r_v \geq value \tag{3.11}$$

In the problem, it is required that the expected total reward is equal or greater than $value$, and the goal is to minimize the cost to satisfy this requirement. The hardness of the quota problem is another open question.

3.9.1 Dynamic Programming Algorithm

Both the quota and the prize-collecting stochastic network design problem under a directed rooted tree can be solved by the dynamic programming algorithm introduced in section 3.3. The only difference is the way to extract the optimal policy when the table at the root is created. For any one of three problems, the table of the root is scanned once, and the optimal policy is extracted. For the budget problem, the optimal policy produces the maximized expected total reward (or value) and has a cost bounded by the budget $\mathcal{B}$. For the quota problem, the optimal policy uses the least cost and has a value equal or greater than $value$. For the prize-collecting problem, the optimal policy produces the maximum value for the objective (3.10).

The correctness of this algorithm to solve both quota problem and the prize-collecting problem is shown by the following intuitions. Remeber that a DP table equivalently stores a list of tuples (z_u, c_u) for subtree u where z_u is an achievable value and c_u is the least cost

to achieve it by some policy to subtree u. For both problems, we can see that the tuple produced by the optimal policy is in the table. For a prize-collecting problem, if the tuple produced by the optimal policy is not in the table, the tuple will take a higher cost than an existing tuple in the table but achieve the same value, which doesn't maximize the objective function (3.10). For a quota problem, since all achievable values are in the table and are attached by the least-costly policy, the optimal policy can be found from the table.

3.9.2 Improving the Dynamic Programming Algorithm

The table can be made smaller by only keeping Pareto-optimal tuples. Given two tuples (z, c) and (z', c'), we say that (z, c) dominates (z', c') if the first has a larger value and no larger cost or has a smaller cost and no smaller value than the second tuple, that is, $r > r^*, c \leq c^*$ or $r \geq r^*, c < c^*$. A tuple (z^*, c^*) is Pareto-optimal tuple in the sense that no policy can produce a tuple that dominates (z^*, c^*). The following propositions say that it is safe to only keep Pareto-optimal tuples in the table while running the dynamic programming algorithm.

Consider a node u with two children v and w.

Proposition 3.9.1. *To construct all Pareto-optimal tuples at subtree u, only Pareto-optimal tuples at v and w are needed.*

Proof. A Pareto-optimal tuple in the table of u cannot be built by a non-Pareto-optimal tuple in the table of v or w. At subtree v, consider two tuples A and B where A dominates B, so B is not Pareto-optimal. For a pair of actions on the edges from u to v and w and a tuple at w, the tuple for subtree u built from B cannot be Pareto-optimal because it is dominated by the tuple built from A. It is true for subtree w as well. Therefore, using only the Pareto-optimal tuples in the tables of v and w, all Pareto-optimal tuples at u can be constructed. □

Proposition 3.9.2. *For a prize-collecting stochastic network design problem under a directed rooted tree, only the pareto optimal tuples are needed to be kept in the table of each subtree while running the dynamic programming algorithm introduced in section 3.3.*

Proof. By Proposition 3.9.1, the table of the root contains all Pareto-optimal tuples. The tuple produced by the optimal policy to problem (3.10) is Pareto-optimal because a non-Pareto-optimal tuple A doesn't maximize the objective function since a tuple B that dominates A can gives a larger objective value than A. Therefore, it is safe to only keep Pareto-optimal tuples. □

Proposition 3.9.3. *For a quota stochastic network design problem under a directed rooted tree, only the pareto optimal tuples are needed to be kept in the table of each subtree while running the dynamic programming algorithm introduced in section 3.3.*

Proof. There exists a Pareto-optimal tuple (r, c) such that c is the minimum value of the objective function in (3.11) and $r \geq value$. Assume that all tuples that satisfy the above condition are non-Pareto-optimal tuples. Let (r', c') be a non-Pareto-optimal tuple that satisfies the above condition and (r, c) be a Pareto-optimal tuple that dominates (r', c'). If $r > r'$ and $c \leq c'$, (r, c) with $r \geq value$ either gives the minimum objective value ($c = c'$) or gives a smaller objective value ($c < c'$). f $r \geq r'$ and $c < c'$, (r, c) gives smaller objective value and has $r \geq value$. Any one of these cases contradicts to the assumption. Therefore, a Pareto-optimal tuple can achieve the minimum objective value and also satisfies the constraint in (3.11). Since, by Proposition 3.9.1, the table of the root contains all Pareto-optimal tuples, we can find such Pareto-optimal tuple by going through the table once. □

The similar result holds to the budget problem.

Proposition 3.9.4. *For a stochastic network design problem under a directed rooted tree, only the pareto optimal tuples are needed to be kept in the table of each subtree while running the dynamic programming algorithm introduced in section 3.3.*

Proof. There exists a Pareto-optimal tuple (r, c) such that r is the maximum value of the objective function in (3.1) and $c \leq \mathcal{B}$. Assume that all tuples that satisfy the above condition are non-Pareto-optimal tuples. Let (r', c') be a non-Pareto-optimal tuple that satisfies the above condition and (r, c) be a Pareto-optimal tuple that dominates (r', c'). If $r > r'$ and $c \leq c'$, (r, c) gives larger objective value and has $c \geq \mathcal{B}$. If $r \geq r'$ and $c < c'$, (r, c) with $c \geq \mathcal{B}$ either gives the maximum objective value ($r = r'$) or gives a larger objective value ($r > r'$). Any one of these cases contradicts to the assumption. Again, such Pareto-optimal tuple can be found from the table by one pass. □

The dynamic programming algorithms for both the quota and the prize-collecting problem can be made into rounded dynamic programming algorithms with reduced complexities. The details and the theoretical results of these rounded algorithms are out of the score of this thesis.

3.10 Summary

In this chapter, I define the stochastic network design framework under directed rooted trees and show how the barrier removal problem can be formulated by the framework. Then, I introduce a fast approximation algorithm called rounded dynamic programming to solve the problem, which can compute nearly optimal policies theoretically and empirically. It is shown that the algorithm is a fully polynomial-time approximation scheme (FPTAS). When applied to solve the barrier removal problem using the network data of the Connecticut River watershed in Massachusetts, RDP can produce near-optimal solution within a small fraction of the runtime of a benchmark algorithm called DP+. RDP can also scale to problems with larger sizes for which DP+ is not scalable. RDP is an effective planning tool for restoring accessibility of native fish habitat in large river networks.

CHAPTER 4

STOCHASTIC NETWORK DESIGN FOR BIDIRECTED TREES

In Chapter 3, the focus is on the stochastic network design framework for directed rooted trees. This chapter studies the framework for bidirected trees. A *bidirected tree* is obtained by converting each edge in an undirected tree into two directed edges with reversed directions. It models a flow that can originate from any node in the tree with certain probability and spreads through the network. To facilitate the spreading, we optimize the connectivity of the network by taking into account both the probability distribution of the starting locations of flows and the number of nodes that can be reached. One real-world problem that can be modeled by the framework is the *bidirectional barrier removal problem*. For the bidirectional barrier removal problem, a random fish, instead of starting from a single location, may start from any node in a river network and move upstream or downstream to reach other nodes. The goal is to decide which barriers to remove/repair to optimize the connectivity of the network.

A stochastic network design problem under bidirected trees turns out to be harder than under directed rooted trees. But, I can still find a fully polynomial-time approximation scheme (FPTAS) by extending the ideas in Chapter 3. The basic ideas are the same. First, a dynamic programming algorithm is proposed to solve the problem optimally, but the algorithm is a pseudo-polynomial time algorithm and is unscalable to solve problems of large sizes. Then, a rounding strategy is used to modify the algorithm to be a fully polynomial-time approximation scheme (FPTAS). The dynamic programming algorithm and the rounding strategy for bidirected trees are more complex and the complexity of the resulted algorithm is higher than for directed rooted trees.

The structure of this chapter is as follows. First, the stochastic network design framework for bidirected trees is defined. Then, the algorithm is given. At last, some empirical results of applying the algorithm to solve the bidirectional barrier removal problem are shown.

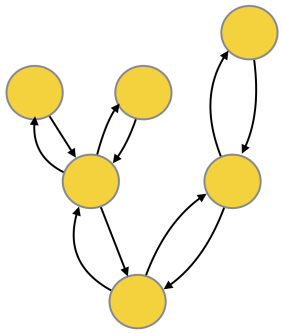


Figure 4.1: A bidirected tree

4.1 Problem Statement

A bidrected tree is a directed graph $\mathcal{T} = (V, E)$ that can be obtained by replacing each edge (u, v) in a undirected tree with two directed edges (u, v) and (v, u) with reverse directions. An example is shown in Fig. 4.1. Each edge $(u, v) \in E$ has a survival probability p_{uv}. Again, we consider mutiple candidate actions on each edge, so a finite set of possible actions $A_{u,v} = A_{v,u}$ is associated with each bidirected edge (u, v). An action $a \in A_{u,v}$ has cost $c_{uv,a}$ and, if taken, simultaneously increases the survival probabilities of two directed edges to $p_{uv|a}$ and $p_{vu|a}$. We assume that $A_{u,v}$ contains a default zero-cost noop action a_0 such that $p_{uv|a_0} = p_{uv}$ and $p_{vu|a_0} = p_{vu}$. A policy π selects an action $\pi(u, v)$—either an actual action or a noop—for each bidirected edge. We write $p_{uv|\pi} := p_{uv|\pi(u,v)}$ for the probability of edge (u, v) under policy π. In addition to the edge probabilities, a non-negative reward $r_{s,t}$ is specified for each pair of vertices $s, t \in V$.

Given a policy π, the s-t *accessibility* $p_{s \rightsquigarrow t|\pi}$ is the product of all edge probabilities on the unique path from s to t, which is the probability that s retains a path to t in the

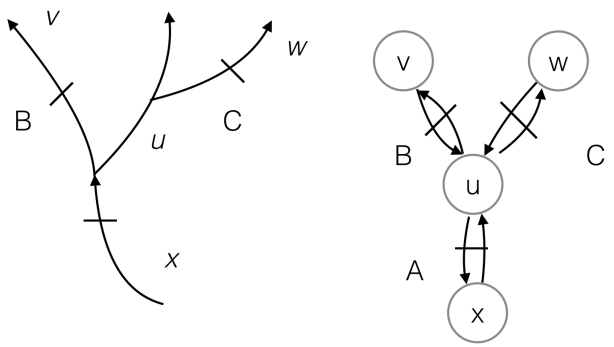


Figure 4.2: Left: sample river network with barriers $\mathrm{A}, \mathrm{B}, \mathrm{C}$ and contiguous regions u, v, w, x. Right: corresponding bidirected tree.

subgraph $\mathcal{T}'$ where each edge is present independently with probability $p_{uv|\pi}$. The total expected reward for policy π is $R(\pi) = \sum_{s,t \in V} r_{s,t}\, p_{s \rightsquigarrow t|\pi}$. Our goal is to find a policy that maximizes $R(\pi)$ subject to a budget b limiting the total cost $c(\pi)$ of the actions being taken. Hence, the resulting policy satisfies $\pi^* \in \arg\max_{\{\pi | c(\pi) \leq b\}} R(\pi)$.

In this work, it is assumed that the rewards factor as $r_{s,t} = h_s h_t$, which is useful for our dynamic programming approach and consistent with several widely used metrics. For example, *network resilience* [11] is defined as the expected number of node pairs that can communicate after random component failures, which is captured in our framework by setting $r_{s,t} = h_s = h_t = 1$. Network resilience is a general model of connectivity that can apply in diverse complex network settings. The ecological measure of *probability of connectivity* (PC) [81], which was the original motivation of our formulation, can also be expressed using factored rewards. PC is widely used in ecology and conservation planning and is implemented in the Conefor software, which is the basis of many planning applications [82]. A precise definition of PC appears later.

4.1.1 Bidirected Barrier Removal Problem

Fig. 4.2 illustrates the bidirectional barrier removal problem in river networks and its mapping to stochastic network design in a bidirected tree. A river network is a tree with

edges that represent stream segments and nodes that represent either stream junctions or barriers that divide segments. Fish begin in each segment and can swim freely between adjacent segments, but can only pass a barrier with a specified *passage probability* or *passability* in each direction; in most cases, downstream passability is higher than upstream passability. To map this problem to stochastic network design, we create a bidirected tree $\mathcal{T} = (V, E)$ where each node $v \in V$ represents a contiguous region of the river network—i.e., a connected set of stream segments among which fish can move freely without passing any barriers—and the value h_v is equal to the total amount of habitat in that region (e.g., the total length of all segments). Each barrier then becomes a bidirected edge that connects two regions, with the passage probabilities in the upstream and downstream directions assigned to the corresponding directed edges. It is easy to see that $\mathcal{T}$ retains a tree structure.

Our objective function $R(\pi)$ is motivated by PC introduced above. It is defined as follows:

$$PC(\pi) = \frac{z(\pi)}{R} = \frac{\sum_{s \in S} \sum_{t \in S} r_{s,t} p_{s \rightsquigarrow t|\pi}}{R} \tag{4.1}$$

where $R = \sum_{s,t} h_s h_t$ is a normalization constant. When h_v is the amount of suitable habitat in region v, $PC(\pi)$ is the probability that a fish placed at a starting point chosen uniformly at random from suitable habitat (so that a point in region s is chosen with probability proportional to h_s) can reach a random target point also chosen uniformly at random by passing each barrier in between.

4.2 Policy Evaluation

Given a bidirected tree $\mathcal{T}$, a divide-and-conquer method can be used to evaluate a policy π. Given an arbitrary node as the *root*, any bidirected tree $\mathcal{T}$ can be viewed as a rooted tree in which each node u has corresponding *children* and *subtrees*. To simplify the algorithm and proofs, the same assumption 3.1.1 as in section 3.1 is made, that is, each node has at most two children.

To evaluate a fixed policy π, we use a divide and conquer method that recursively computes a tuple of three values per subtree of the given tree $\mathcal{T}$. Let v and w be the children of u. The tuple of the subtree rooted at u denoted by $\mathcal{T}_u$ can be calculated using the tuples of subtrees $\mathcal{T}_v$ and $\mathcal{T}_w$. Once the tuple of $\mathcal{T}_{root}$, namely $\mathcal{T}$, is calculated, we can extract the total expected reward from that tuple.

Now, we define the tuple of $\mathcal{T}_u$ to be $\psi_u(\pi) = (\nu_u(\pi), \mu_u(\pi), z_u(\pi))$.

- $\nu_u(\pi) = \sum_{t \in \mathcal{T}_u} p_{u \rightsquigarrow t|\pi} h_t$ is the summation of the s-t accessibilities of all paths from u to $t \in T_u$, each of which is weighted by the habitat h_t of its ending node t.
- $\mu_u(\pi) = \sum_{s \in \mathcal{T}_u} p_{s \rightsquigarrow u|\pi} h_s$ is the summation of the s-t accessbilities of all paths from $s \in T_u$ to u, each of which is weighted by the habitat h_s of its departing node s.
- $z_u(\pi) = \sum_{s \in \mathcal{T}_u} \sum_{t \in \mathcal{T}_u} p_{s \rightsquigarrow t|\pi} r_{s,t}$ ($r_{s,t} = h_s h_t$) represents the total expected reward that a fish obtained by following paths with both starting node and ending node being in $\mathcal{T}_u$.

The tuple $\psi_u(\pi)$ is calculated recursively using $\psi_v(\pi)$ and $\psi_w(\pi)$. To calculate $\nu_u(\pi)$, we note that a path from u to a node in $\mathcal{T}_u \backslash \{u\}$ is the concatenation of either the edge (u, v) with a path from v to $\mathcal{T}_v$ or the edge (u, w) with a path from w to $\mathcal{T}_w$, that is, $\nu_u(\pi)$ can be written as

$$\sum_{t \in \mathcal{T}_v} p_{uv|\pi} p_{v \rightsquigarrow t|\pi} h_t + \sum_{t \in \mathcal{T}_w} p_{uw|\pi} p_{w \rightsquigarrow t|\pi} h_t + h_u = p_{uv|\pi} \nu_v(\pi) + p_{uw|\pi} \nu_w(\pi) + h_u \quad (4.2)$$

Similarly, $\mu_u(\pi) =$

$$\sum_{s \in \mathcal{T}_v} p_{s \rightsquigarrow v|\pi} p_{vu|\pi} h_s + \sum_{s \in \mathcal{T}_w} p_{s \rightsquigarrow w|\pi} p_{wu|\pi} h_s + h_u = p_{vu|\pi} \mu_v(\pi) + p_{wu|\pi} \mu_w(\pi) + h_u \quad (4.3)$$

By dividing the reward from paths that start and end in $\mathcal{T}_u$ based on their start and end nodes, we can express $z_u(\pi)$ as follows:

$$z_u(\pi) = z_v(\pi)+z_w(\pi)+\mu_v(\pi)p_{v\rightsquigarrow w|\pi}\nu_w(\pi)+\mu_w(\pi)p_{w\rightsquigarrow v|\pi}\nu_v(\pi)+h_u\nu_u(\pi)+h_u\mu_u(\pi)-h_u^2 \tag{4.4}$$

The first two terms describe paths that start and end within a single subtree—either $\mathcal{T}_v$ or $\mathcal{T}_w$. The third and fourth terms describe paths that start in $\mathcal{T}_v$ and end in $\mathcal{T}_w$ or vice versa. The last three terms describe paths that start or end at u, with an adjustment to avoid double-counting the trivial path that starts *and* ends at u. That way, all tuples can be evaluated with one pass from the leaf nodes to the root and each node is only visited once. At the root, $z_{root}(\pi)$ is the expected reward of policy π, that is, $z_{root}(\pi) = R(\pi)$.

4.3 Dynamic Programming Algorithm

This section introduces the dynamic programming algorithm to compute the optimal policy. Again, let π_u be the subpolicy for subtree $\mathcal{T}_u$. Instead of maintaining a list of achievable z values, the DP table of each subtree $\mathcal{T}_u$ contains a list of tuples ψ that are reachable by some subpolicies and each tuple is associated with a least-cost subpolicy, that is, $\pi_u^* \in \arg\min_{\{\pi_u|\psi_u(\pi_u)=\psi\}} c(\pi_u)$.

In the same manner, the list of reachable tuples and the associated least-cost subpolicies of u are generated using the tuples of v and w. To do this, for each ψ_v, ψ_w, we first extract the corresponding π_v^* and π_w^*. Then, using these two least-cost subpolicies of the children, for each $a \in A_{uv}$ and $a' \in A_{uw}$, a new subpolicy π_u is constructed for $\mathcal{T}_u$ with cost $c(\pi_u) = c_{uv,a} + c_{uw,a'} + c(\pi_v^*) + c(\pi_w^*)$. Using Eqs. (4.2), (4.3) and (4.4), the tuple $\psi_u(\pi_u)$ of π_u is calculated. If $\psi_u(\pi_u)$ already exists in the list (i.e., $\psi_u(\pi_u)$ was created by some other previously constructed subpolicies), we update the associated subpolicy such that only the minimum cost subpolicy is kept. If not, we add this tuple $\psi_u(\pi_u)$ and subpolicy π_u to the list.

To initialize the recurrence, the list of a leaf subtree contains only a single tuple (h_u, h_u, h_u^2) associated with an empty subpolicy. Once the list of $\mathcal{T}_{root}$ is calculated, we scan the list to pick a pair (ψ^*_{root}, π^*) such that $(\psi^*_{root}, \pi^*) \in \arg\max_{\{(\psi_{root},\pi)|c(\pi)\leq b\}} z_{root}$ where z_{root} is the third element of ψ_{root}. Finally, π^* is the returned optimal policy and z^*_{root} is the optimal expected reward.

4.4 Rounded Dynamic Programming

The DP algorithm is not a polynomial-time algorithm because the number of reachable tuples increases exponentially as we approach the root. In this section, we modify the DP algorithm into an FPTAS algorithm. The idea is to discretize the space of ψ_u into a polynomial number of different tuples. To do this, the three dimensions are discretized using granularity factors K_u^ν, K_u^μ and K_u^z respectively such that the space is divided into a finite number of cubes with volume $K_u^\nu \times K_u^\mu \times K_u^z$.

For any subpolicy π_u of u in the discretized space, there is a rounded tuple $\hat{\psi}_u(\pi_u) = (\hat{\nu}_u(\pi_u), \hat{\mu}_u(\pi_u), \hat{z}_u(\pi_u))$ to underestimate the true tuple $\psi_u(\pi_u)$ of π_u. To evaluate $\hat{\psi}_u(\pi_u)$, we use the same recurrences as (4.2), (4.3) and (4.4), but rounding each intermediate value into a value in the discretized space. The recurrences are as follow:

$$\hat{\nu}_u^{sum}(\pi_u) = p_{uv|\pi_u}\hat{\nu}_v(\pi_u) + p_{uw|\pi_u}\hat{\nu}_w(\pi_u) + h_u \quad \hat{\mu}_u^{sum}(\pi_u) = p_{vu|\pi_u}\hat{\mu}_v(\pi_u) + p_{wu|\pi_u}\hat{\mu}_w(\pi_u) + h_u$$

$$\hat{\nu}_u(\pi_u) = K_u^\nu \left\lfloor \frac{\hat{\nu}_u^{sum}(\pi_u)}{K_u^\nu} \right\rfloor \qquad \hat{\mu}_u(\pi_u) = K_u^\mu \left\lfloor \frac{\hat{\mu}_u^{sum}(\pi_u)}{K_u^\mu} \right\rfloor \tag{4.5}$$

$$\hat{z}_u(\pi_u) = K_u^z \cdot \left\lfloor \frac{\hat{z}_v(\pi_u) + \hat{z}_w(\pi_u) + \hat{\mu}_v(\pi_u) p_{v\rightsquigarrow w|\pi_u} \hat{\nu}_w(\pi_u) + \hat{\mu}_w(\pi_u) p_{w\rightsquigarrow v|\pi_u} \hat{\nu}_v(\pi_u) + h_u \hat{\mu}_u^{sum}(\pi_u) + h_u \hat{\nu}_u^{sum}(\pi_u) - h_u^2}{K_u^z} \right\rfloor \tag{4.6}$$

The modified algorithm, which I called rounded dynamic programming for bidirected trees (RDPB)is the same as the DP algorithm, except that it works in the discretized space.

That is, each node maintains a list of reachable rounded tuples $\hat{\psi}_u$, each one associated with a least costly subpolicy achieving $\hat{\psi}_u$, that is, $\pi_u^* \in \arg\min_{\{\pi_u | \hat{\psi}_u(\pi_u) = \hat{\psi}_u\}} c(\pi_u)$. The reachable tuples for each node are generated by its children's lists of tuples using recurrences (4.5) and (4.6) instead of (4.2), (4.3) and (4.4).

4.4.1 Theoretical Analysis

The main result for the RDPB is as follows.

Theorem 4.4.1. *RDPB is an FPTAS. Specifically, let OPT be the value of the optimal policy. Then, RDPB can compute a policy with value at least $(1-\epsilon)OPT$ in time bounded by $O(\frac{n^8}{\epsilon^6})$.*

Now, we prove the theorem by showing approximation guarantee and analyzing the running time.

4.4.1.1 Approximation Guarantee

Let π^* be the optimal policy and let π' be the policy returned by RDPB. The value loss $z(\pi^*) - z(\pi')$ is bounded if the distance between the true tuple $\psi(\pi)$ and the rounded tuple $\hat{\psi}(\pi)$ is bounded for an arbitrary policy π. In Eqs. (4.5) and (4.6), starting from leaf nodes, each rounding operation introduces an error at most $K_u^{\cdot}$ where $\cdot$ represents ν, μ or z.

For ν, starting from u, each node $t \in \mathcal{T}_u$ introduces error K_t^ν by using the rounding operation. The error is discounted by the accessibility from u to t. For μ, each node $s \in \mathcal{T}_u$ introduces error K_s^μ, discounted in the same way. The total error is equal to the sum of all discounted errors.

Finally, we get the following result by setting

$$K_u^\nu = \frac{\epsilon}{3} h_u, \quad K_u^\mu = \frac{\epsilon}{3} h_u, \quad K_u^z = \frac{\epsilon}{3} h_u^2 \tag{4.7}$$

Lemma 4.4.1. *If condition (4.7) holds, then for all $u \in V$ and an arbitrary policy π:*

$$\nu_u(\pi) - \hat{\nu}_u(\pi) \leq \sum_{t \in \mathcal{T}_u} p_{u \rightsquigarrow t|\pi} K_t^{\nu} = \frac{\epsilon}{3} \sum_{t \in \mathcal{T}_u} p_{u \rightsquigarrow t|\pi} h_t = \frac{\epsilon}{3} \nu_u(\pi) \tag{4.8}$$

$$\mu_u(\pi) - \hat{\mu}_u(\pi) \leq \sum_{s \in \mathcal{T}_u} p_{s \rightsquigarrow u|\pi} K_s^{\mu} = \frac{\epsilon}{3} \sum_{s \in \mathcal{T}_u} p_{s \rightsquigarrow u|\pi} h_s = \frac{\epsilon}{3} \mu_u(\pi) \tag{4.9}$$

The difference of $z(\pi) - \hat{z}(\pi)$ is bounded by the following lemma.

Lemma 4.4.2. *If condition (4.7) holds, $z(\pi) - \hat{z}(\pi) \leq \epsilon z(\pi)$ for an arbitrary policy π.*

Proof. To prove the lemma, we first use induction to prove the following statement:

$$z_u(\pi) - \hat{z}_u(\pi) \leq \epsilon z_u(\pi) \quad \forall u \in V$$

For the base case where u is a leaf node, we have $z_u(\pi) = h_u^2$ and therefore the error $z_u(\pi) - \hat{z}_u(\pi)$ is bounded by $K_u^z = \frac{\epsilon}{3} z_u(\pi)$.

For the induction step, assume that the statement holds for the two children v and w of u. To prove that it also holds for u, we consider the error introduced by **each term** in recurrence (4.6) that uses the rounded values $\hat{\nu}_v(\pi)$ and $\hat{\mu}_w(\pi)$ instead of the true values $\nu_v(\pi)$ and $\mu_w(\pi)$. For the term $\hat{\mu}_v(\pi) p_{v \rightsquigarrow w|\pi} \hat{\nu}_w(\pi)$, the introduced error is

$$\mu_v(\pi) p_{v \rightsquigarrow w|\pi} \nu_w(\pi) - \hat{\mu}_v(\pi) p_{v \rightsquigarrow w|\pi} \hat{\nu}_w(\pi)$$

. Using Lemma 4.4.1, the error is

$$\leq \left(\frac{\epsilon}{3} \mu_v(\pi) \hat{\nu}_w(\pi) + \frac{\epsilon}{3} \hat{\mu}_v(\pi) \nu_w(\pi) + \frac{\epsilon^2}{3^2} \mu_v(\pi) \nu_w(\pi) \right) p_{v \rightsquigarrow w|\pi} \leq \epsilon \mu_v(\pi) p_{v \rightsquigarrow w|\pi} \nu_w(\pi)$$

where the last inequality holds because the rounded value always underestimates the true value.

Similarly, the error for the term $\hat{\mu}_w(\pi)p_{w\rightsquigarrow v|\pi}\hat{\nu}_v(\pi)$ is bounded by $\epsilon\mu_w(\pi)p_{w\rightsquigarrow v|\pi}\nu_v(\pi)$. For the term $h_u\hat{\mu}_u^{sum}$, the error is $h_u(\mu_u(\pi) - \hat{\mu}_u^{sum}(\pi))$. By using Lemma (4.4.1), the error is bounded by $h_u\frac{\epsilon}{3}(\mu_u(\pi) - h_u)$. Similarly, the error for the term $h_u\hat{\nu}_u^{sum}$ is bounded by $h_u\frac{\epsilon}{3}(\nu_u(\pi) - h_u)$. In addition, by the inductive assumption, the errors for $\hat{z}_v(\pi)$ and $\hat{z}_w(\pi)$ are bounded by $\epsilon z_v(\pi)$ and $\epsilon z_w(\pi)$ respectively.

Therefore, the total error for the enumerator of Equation (4.6) is bounded by

$$\epsilon(z_v(\pi) + z_w(\pi) + \mu_v(\pi)p_{v\rightsquigarrow w|\pi}\nu_w(\pi) + \mu_w(\pi)p_{w\rightsquigarrow v|\pi}\nu_v(\pi) + h_u\nu_{\cdot,u}(\pi) + h_u\mu_u(\pi) - 2\cdot h_u^2)$$

According to the definition of $\hat{z}_u(\pi)$ in Equation (4.6), the enumerator is divided by K_u^z, rounded and then multiplied by K_u^z. These operations introduce an additional error K_u^z, which is bounded by ϵh_u^2 based on condition (4.7) in the paper. By adding the error to the total error above, the error $z_u(\pi) - \hat{z}_u(\pi)$ is bounded by $\epsilon z_u(\pi)$ where the expression of $z_u(\pi)$ is shown in Equation (4.4) in the paper. Hence, the statement holds for u.

Finally, as defined in the paper, $z(\pi) = z_{root}(\pi)$ and $\hat{z}(\pi) = \hat{z}_{root}(\pi)$. Therefore, the lemma holds. □

It is ready to show the bound of $z(\pi^*) - z(\pi')$–the main result for the approximation guarantee.

Theorem 4.4.2. *Let π^* and π' be the optimal policy and the policy return by RDPB respectively. Then, if condition (4.7) holds, we have $z(\pi^*) - z(\pi') \leq \epsilon z(\pi^*)$.*

Proof. By Lemma 4.4.2, we have $z(\pi^*) - \hat{z}(\pi^*) \leq \epsilon z(\pi^*)$. Furthermore, $z(\pi') \geq \hat{z}(\pi') \geq \hat{z}(\pi^*)$ where the second inequality holds because π' is the optimal policy with respect to the rounded policy value. Therefore, we have $z(\pi^*) - z(\pi') \leq z(\pi^*) - \hat{z}(\pi^*)$ which proves the theorem. □

4.4.1.2 Runtime Analysis

We derive the runtime result of Theorem 4.4.1, that is, if condition (4.7) holds, the runtime of RDPB is bounded by $O(\frac{n^8}{\epsilon^6})$. First, it is reasonable to make the following assumption as for the directed rooted tree setting:

Assumption 4.4.1. *The value h_u is constant with respect to n and ϵ for each $u \in V$.*

Let $m_{u,\hat{\nu}}$, $m_{u,\hat{\mu}}$ and $m_{u,\hat{z}}$ denote the number of different values for $\hat{\nu}_u$, $\hat{\mu}_u$ and $\hat{z}_u$ respectively in the rounded value space of u.

Lemma 4.4.3. *If condition (4.7) holds, then*

$$m_{u,\hat{\nu}} = O\Big(\frac{n_u}{\epsilon}\Big), \qquad m_{u,\hat{\mu}} = O\Big(\frac{n_u}{\epsilon}\Big), \qquad m_{u,\hat{z}} = O\Big(\frac{n_u^2}{\epsilon}\Big) \tag{4.10}$$

for all $u \in V$ where n_u is the number of vertices in subtree $\mathcal{T}_u$.

Proof. The number $m_{u,\hat{\nu}}$ is bounded by $\frac{\sum_{t\in\mathcal{T}_u} h_t}{K_u^{\nu}}$ where $\sum_{t\in\mathcal{T}_u} h_t$ is a naive and loose upper bound of ν_u obtained assuming all passabilities of streams in $\mathcal{T}_u$ are 1.0. By Assumption (4.4.1), $m_{u,\hat{\nu}} = O(\frac{n_u}{\epsilon})$. The upper bound of $m_{u,\hat{\mu}}$ can be similarly derived. Assuming all passabilities are 1.0, the upper bound of z_u is $\sum_{s\in\mathcal{T}_u}\sum_{t\in\mathcal{T}_u} h_s h_t$. Therefore, $m_{u,\hat{z}} \leq \frac{\sum_{s\in\mathcal{T}_u}\sum_{t\in\mathcal{T}_u} h_s h_t}{K_u^{z}} = O(\frac{n_u^2}{\epsilon})$ □

Recall that RDPB works by recursively calculating the list of reachable rounded tuples and associated least costly subpolicy. Using Lemma 4.4.3, the following main result can be obtained:

Theorem 4.4.3. *If condition (4.7) holds, the runtime of RDPB is bounded by $O(\frac{n^8}{\epsilon^6})$.*

Proof. Let $T(n)$ be the maximum runtime of RDPB for any subtree with n vertices. In RDPB, for node u with children v and w, we compute the list and associated subpolicies by iterating over all combinations of $\hat{\psi}_v$ and $\hat{\psi}_w$. For each combination, we iterate over

all available action combinations $a_{uv} \in A_{uv}$ and $a_{uw} \in A_{uw}$, which takes constant time because the number of available candidate actions are constant w.r.t. n and ϵ. Therefore, we can bound $T(n)$ using the following recurrence:

$$\begin{aligned} T(n_u) &= O(m_{v,\hat{\nu}} m_{v,\hat{\mu}} m_{v,\hat{z}} m_{w,\hat{\nu}} m_{w,\hat{\mu}} m_{w,\hat{z}}) + T(n_v) + T(n_w) \leq c\frac{n_v^4 n_w^4}{\epsilon^6} + T(n_v) + T(n_w) \\ &\leq \max_{0 \leq k \leq (n_u - 1)} c\frac{k^4 (n_u - k - 1)^4}{\epsilon^6} + T(k) + T(n_u - k - 1) \end{aligned}$$

where $n_u = 1 + n_v + n_w$ as $\mathcal{T}_u$ consists of u, $\mathcal{T}_v$ and $\mathcal{T}_w$. The second inequality is due to Lemma 4.4.3. The third inequality is obtained by a change of variable.

It can be shown that $T(n) \leq c\frac{n^8}{\epsilon^6}$ by induction. For the base case $n = 0$, we have $T(n) = 0$ and for the base case $n = 1$, the subtree only contains one node, so $T(n) = c$. Now assume that $T(k) \leq c\frac{k^8}{\epsilon^6}$ for all $k < n$. Then, we can show that

$$T(n) \leq \max_{0 \leq k \leq (n-1)} \frac{c}{\epsilon^6} \left(k^4 (n - k - 1)^4 + k^8 + (n - k - 1)^8 \right) \leq c\frac{n^8}{\epsilon^6} \tag{4.11}$$

Thus, the theorem holds. The first inequality is obtained by plugging $T(k)$ and $T(n_u - k - 1)$. What is left is to prove the second inequality.

To derive the second inequality, we show for any integer $k \in [0, n-1]$ that

$$\begin{aligned} & k^4 (n - k - 1)^4 + k^8 + (n - k - 1)^8 \\ & \leq 2k^4 (n - k - 1)^4 + k^8 + (n - k - 1)^8 \\ & = \left(k^4 + (n - k - 1)^4 \right)^2 \\ & \leq \left(k^4 + 2k^2 (n - k - 1)^2 + (n - k - 1)^4 \right)^2 = \left(\left(k^2 + (n - k - 1)^2 \right)^2 \right)^2 \\ & \leq \left(\left(k^2 + 2k(n - k - 1) + (n - k - 1) \right)^2 \right)^4 = \left((k + n - k - 1)^2 \right)^4 \\ & = (n - 1)^8 \leq n^8 \end{aligned}$$

Therefore, we have

$$\max_{0 \leq k \leq (n-1)} k^4(n-k-1)^4 + k^8 + (n-k-1)^8 \leq n^8$$

which proves the inequality. Therefore, the assumption $T(n) \leq c\frac{n^8}{\epsilon^6}$ holds, which proves the theorem. □

4.5 Implementation

In the proof of the main theorem 4.4.1, the values of Ks are set by (4.7) The empirical runtime is much faster than the upper bound of runtime in theorem 4.4.3 due to the following observations. First, Ks can be set much larger than in (4.7) because the upper bound of runtime we found may not be the tightest one. Second, a lot of tuples in discretized space are not reachable while the proof of the upper bound runtime is based on the assumption that all tuples in the space are accessed.

Therefore, the performance of RDPB can be improved by making the following two modifications. First, before calculating the list of reachable tuples of u, we estimate the upper bound and lower bound of the reachable values of $\hat{\nu}_u$, $\hat{\mu}_u$ and $\hat{z}_u$ using the list of tuples of its children. Then, we dynamically assign the values to $K_u^{\cdot}$ by fixing the total number of different discrete values of $\hat{\nu}_u$, $\hat{\mu}_u$ and $\hat{z}_u$ in the space, that is, this number determines the granularity of discretization. For example, if the upper bound and the lower bound of $\hat{\nu}_u$ are 1000 and 500 respectively and we want 10 different values, the value of K_u^{ν} is set to be $\frac{1000-500}{10} = 50$. With a finer granularity of discretization, a slower algorithm is obtained, but it has a better solution quality. In our experiments, by setting the numbers to be 50, 50, 150 for $\hat{\nu}_u$, $\hat{\mu}_u$ and $\hat{z}_u$, the algorithm is very fast and we can get very good solution quality. Second, sparse matrix representation is used to store and access the reachable tuples, which makes the runtime much faster than the upper bound that assumes all tuples in the space are used once.

4.6 Experiments

In experiments, the RDPB algorithm is applied to solve the bidirectional barrier removal problem introduced in section 4.1. The data is from the CAPS project [58] for river networks in Massachusetts (Fig. 2.2) mentioned in section 2.2.2. Barrier passabilities are calculated from barrier features using the model defined by the CAPS project. We created actions to model practical repair activities. For road-crossings, most passabilities start close to 1 and are cheap to repair relative to dams. To model this, we set $A_{u,v} = \{a_1\}$, $p_{uv|a_1} = p_{vu|a_1} = 1.0$ and $c_{uv|a_1} = 5$. In contrast, it is difficult and expensive to remove dams, so multiple strategies must be considered to improve their passability. We created actions $A_u = \{a_1, a_2, a_3\}$ with action a_1 having $p_{uv|a_1} = p_{vu|a_1} = 0.2$ and $c_{uv|a_1} = 20$; action a_2 having $p_{uv|a_2} = p_{vu|a_2} = 0.5$ and $c_{uv|a_2} = 40$; and action a_3 having $p_{uv|a_3} = p_{vu|a_3} = 1.0$ and $c_{uv|a_3} = 100$. Note that the passage probabilities from upstream and downstream are the same. Later, the case that two probabilities are different is considered.

Now, we present the experimental results.

4.6.1 Results on Small Networks

We compared SAA+MILP, RDPB and a greedy algorithm (denoted by Greedy) on small river networks. The algorithm called SAA+MILP does not guarantee to produce optimal solutions, which is discussed in Chapter 5. The basic idea of it is to use a sample average approximation (SAA) procedure to approximate the stochastic network design problem by a mixed linear integer program (MILP) and find the optimal solution of the program as an approximate solution. SAA+MILP used 20 samples for the sample average approximation and IBM CPLEX on 12 CPU cores to solve the integer program. RDPB1 used finer discretization than RDPB2, therefore requiring a longer runtime. The results in Table 4.1 show that RDPB1 gives the best increase in expected reward (relative to a zero-cost policy) in most cases and RDPB2 produces similarly good solutions, but takes less time. Although Greedy is extremely fast, it produces poor solutions on some networks.

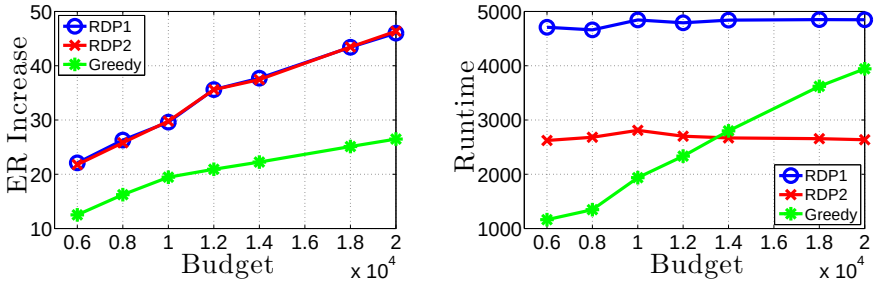
SAA+MILP gives better results than Greedy, but fails to scale up. For example, on a network with 781 segments and 604 barriers, SAA+MILP needs more than 16G of memory to construct the MILP.

number of		ER Increase				Runtime			
Segments	barriers	SAA+MILP	Greedy	RDPB1	RDPB2	SAA+MILP	Greedy	RDPB1	RDPB2
106	36	3.7	**4.1**	4.1	4.0	3.3	0.0	0.7	0.4
101	71	4.0	3.6	**4.3**	4.3	19.5	0.0	2.5	1.2
163	91	11.3	11.2	**12.3**	12.1	42.3	0.0	13.6	6.8
263	289	20.7	11.1	**25.3**	24.8	1148.7	0.7	263.3	98.7
499	206	48.6	**55.6**	53.8	53.2	116.0	0.7	11.9	6.4
456	464	124.1	96.8	**146.9**	144.3	8393.5	0.7	359.9	142.0
639	609	51.8	25.8	**53.7**	51.6	12720.1	1.3	721.2	242.4

Table 4.1: Comparison of SAA+MILP, RDPB and Greedy. Time is in second. Each unit of expected reward is 10^7 (square meters). "ER increase" means the increase in expected reward after taking the computed policy.

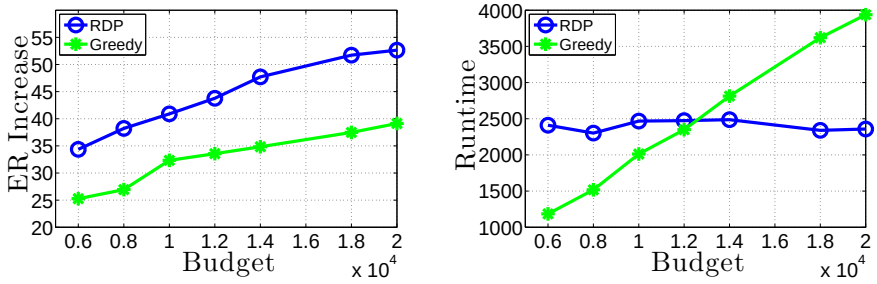
4.6.2 Results on Large Networks

We compared RDPB and Greedy on a large network—the Connecticut River watershed, which has 10451 segments, 587 dams and 7545 crossings. We tested both algorithms on three different settings of action passabilities.



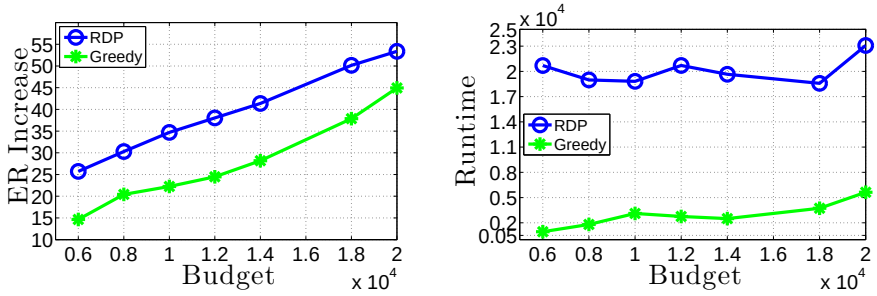
(a) Expected reward increase (b) Runtime in seconds

Figure 4.3: Aymmetric passabilities.



(a) Expected reward increase (b) Runtime in seconds

Figure 4.4: Asymmetric passabilities & downstream passabilities $= 1$



(a) Expected reward increase (b) Runtime in seconds

Figure 4.5: Asymmetric passabilities & varying downstream passabilities.

4.6.3 Actions with Symmetric Passabilities

In this experiment, we used the action setting introduced earlier. The expected reward increase (Fig. 4.3a) and runtime (Fig. 4.3b) are plotted for different budgets. For the expected reward, each unit represents $10^{14}m^2$. Runtime is in seconds. As before, RDPB1 uses finer discretization of tuple space than RDPB2. As Fig. 4.3 shows, the RDPB algorithms give much better solution quality than the greedy algorithm. With a budget of 20000, the ER increase of RDPB1 is almost twice the increase for Greedy. Incidentally, RDPB1 doesn't improve the solution quality by much, but it takes much longer time to finish. Notice that both RDPB1 and RDPB2 use constant runtime because the number of discrete

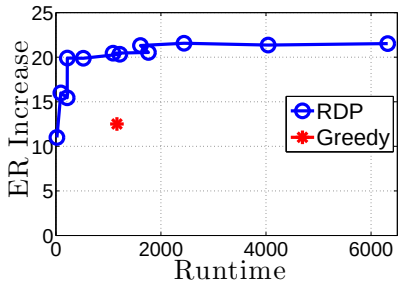


Figure 4.6: Time/quality tradeoffs

values in both settings are bounded. In contrast, the runtime of Greedy increases with the budget size and eventually exceeds RDPB2's runtime.

4.6.4 Actions with Asymmetric Passabilities

The RDPB algorithms work with asymmetric passabilities as well. For road-crossings, we set the actions to be the same as before as the initial passabilities are already near 1. For dams, we first considered the case in which the downstream passabilities are all 1—which happens for some fish—and all upstream passabilities are the same as before. The results are shown in Figures 4.4a and 4.4b. In this case, RDPB still performs better than Greedy and tends to use less time as the budget increases.

We also considered a hard case in which the downstream passabilities of a dam are: $(p_{vu|a_1} = 0.8)$, $(p_{vu|a_2} = 0.9)$ and $(p_{vu|a_3} = 1.0)$. These variations of passabilities produce more tuples in the discretized space. Our RDPB algorithm still works well and produces better solutions than Greedy over a range of budgets as shown in Fig. 4.5a. As expected in such hard cases, RDPB needs much more time than Greedy. However, obtaining high quality solutions to such complex conservation planning problems in a matter of hours makes the approach very valuable.

4.6.5 Tradeoff Between Time and Solution Quality

Finally, we tested the time/quality tradeoff offered by RDPB. The tradeoff is controlled by varying the level of discretization. These experiments are done using the network of Connecticut River watershed with symmetric passabilities. Fig. 4.6 shows how runtime and expected reward grow as we refine the level of discretization. As can be seen, RDPB converges quickly on high-quality results and exhibits the desired diminishing returns property of anytime algorithms—the quality gain is large initially and it diminishes as we continue to refine the discretization.

4.7 Limitation

In this chapter and chapter 3, the focus is on the stochastic network design framework for both directed rooted trees and bidirected trees, and on how two versions of the barrier removal problem are formulated using the framework. Two algorithms RDP and RDPB are developed, one for each setting. With the basic setting of stochastic network design defined in section 2.1, each edge is associated with only one available action, that is, we can either take the action or do nothing. One advantage of RDP and RDPB is that they can work with multiple action alternatives. That is, at each edge, one of the multiple actions can be taken to change the survival probability. Both algorithms also have limitations. For RDP, one action can only raise the probability of one edge. For RDPB, one action will change the probabilities of exact two edges. They are not general enough to deal with the setting mentioned in section 2.4.2. That is, one action can affect an arbitrary number of edges. It is an interesting future work how to develop efficient approximation algorithms to deal with this general setting.

4.8 Summary

This chapter discusses the stochastic network design framework for bidirected trees, which models the flow of influence that may start from any node of a tree and spreads into

the tree along both directions of edges. It is shown how a bidirectional barrier removal problem can be formulated as a stochastic network design problem. To solve the problem, a fast approximation algorithm called rounded dynamic programming for bidirected trees (RDPB) is developed, which can compute nearly optimal policies both theoretically and empirically. Theoretically, the algorithm is a fully polynomial-time approximation scheme that takes time $O(\frac{n^8}{\epsilon^6})$ to produce a $(1-\epsilon)$ optimal solution. Empirically, applying the algorithm to solve the bidirectional barrier removal problem, the results show that the algorithm can run much faster than this theoretical runtime bound.

CHAPTER 5

STOCHASTIC NETWORK DESIGN FOR GENERAL DIRECTED GRAPHS

The focus of this chapter is on the stochastic network design framework for general directed graphs–a more general setting than Chapter 3 and 4. A real-world problem called *spatial conservation planning* is introduced and is formulated using the framework. For general directed graphs, problems become harder, and building an FPTAS seems impossible. Instead, an approximate algorithm using different optimization techniques can be built. Although it is not theoretically proved that the solution produced by the algorithm is guaranteed to be near optimal within a fixed percentage, empirical results show that the algorithm can produce nearly optimal solutions. The basic idea of the algorithm is to use the sample average approximation (SAA) method to convert the stochastic network design problem into a deterministic network design problem, solve the deterministic network design problem, and use the computed solution as an approximate solution of the original problem. The deterministic optimization problem can be formulated as a mixed integer program (MIP), but solving it directly using a standard MIP solver appears to be very time-consuming. To increase the scalability, I develop an approximate algorithm combining the Lagrangian relaxation technique and a primal-dual algorithm. Applying to a conservation planning problem in which the goal is to facilitate the dispersal of red-cockaded woodpeckers (RCWs) within a landscape, the algorithm runs much faster than a MIP solver and a greedy baseline and can produce high-quality solutions.

The structure of this chapter is as follows. First, the definition of the stochastic network design framework is briefly reviewed. Then, the SAA method is introduced, and some of its theoretical properties are given. After that, it is shown that how the SAA method helps

to write a stochastic network design problem into a MIP. At last, the fast algorithm to solve the MIP is introduced, and experimental results are given.

5.1 Problem Statement

The input of a stochastic network design problem is a stochastic network defined by a directed graph $G = (V, E)$, where V is a set of nodes and E is a set of directed edges. Each edge is either present (survival) or absent (failed). The initial survival probability of an edge (u, v) is p^o_{uv}. At each edge, an action denoted by a_{uv} with cost c_{uv} can be taken to raise the survival probability to p^m_{uv} ($p^m_{uv} > p^o_{uv}$). The probability of failure equals to one minus the survival probability. Each pair of nodes (s, t) is associated with a reward r_{st}. In the stochastic network, if all edges on the path from s to t are present, the reward r_{st} can be collected. That is, each reward can be collected with a probability, which can be raised by taking actions. A policy π selects an action $\pi(u, v)$—either the available action or no-action—for each edge. The goal of the stochastic network design problem is to find a policy π to maximize the expected total reward that can be collected. Mathematically, the problem can be written as

$$\max_{\pi} R(\pi) = \sum_{s,t \in V} r_{s,t}\, p_{s \rightsquigarrow t|\pi} \qquad \text{subject to} \quad cost(\pi) \leq \mathcal{B} \tag{5.1}$$

where $p_{s \rightsquigarrow t|\pi}$ is the probability that there exists at least one path from s to t on which all edges are present, and $\mathcal{B}$ is a budget limit. This is a hard stochastic network design problem as shown by the following theorem.

Theorem 5.1.1. *For a stochastic network defined on a directed graph, calculation of the probability that a pair of nodes are connected is #P-hard.*

Proof. The theorem can be derived from the results in paper [93]. □

The theorem says that for a given policy π, calculation of $R(\pi)$ is at least #P-hard while #P-hard is at least as hard as NP-hard. Therefore, to build a scalable algorithm, we focus on approximate algorithms.

5.2 Sample Average Approximation

The stochastic network design problem is a one-stage stochastic optimization problem. That is, we take all actions that we need before the outcomes of any stochastic events are revealed. After these actions are executed, we don't take any actions in the future. In contrast, actions are taken over multiple stages in a multiple-stage stochastic optimization problem. The outcomes of random events are gradually revealed at different stages.

One-stage stochastic optimization problems have been studied by people in the field of operations research [22, 86], and the Monte Carlo sampling is one of the widely used methods [35, 23, 46]. A one-step stochastic optimization problem can be written as

$$\min_{\pi} z(\pi) := \mathbb{E}_{\xi}[Z(\pi, \xi)] \qquad s.t. \quad cost(\pi) \leq \mathcal{B} \tag{5.2}$$

π (a policy) represents all decision variables. ξ is set of random variables that model all random events. The probability distribution of ξ is **independent** from the value of π. The property enables us to use sample average to approximate the expectation of ξ. As shown later, a stochastic network design problem can be written as (5.2) and satisfies this property if π and ξ are defined in a specific way. $Z(\cdot)$ is a function of both π and ξ. $cost(\pi)$ is the cost of a policy π. $\mathcal{B}$ is a real number, such as budget limit, bounds the cost of the policy.

An optimization problem in form (5.2) is hard if the domains of π and ξ contain an exponentially large number of different values with respect to the problem size, which is usually true for many problems. For some problems, the domain of ξ may even contain an infinite number of different values. So, evaluation of $Z(\pi, \xi)$ becomes intractable. To

reduce the complexity, the idea of a sampling method is to draw a sequence of samples of ξ and use the sample average to approximate the expectation, which is shown as follows.

$$z(\pi) \approx \hat{z}(\pi) := \frac{1}{N} \sum_{i=1:N} Z(\pi, \xi_i) \tag{5.3}$$

where $\{\xi_1, ..., \xi_N\}$ are N independent samples of ξ. Also, we have the following theorem.

Theorem 5.2.1. *For a given x, $\hat{z}(\pi)$ converges to $z(\pi)$ as N goes to infinity.*

Proof. Since each $Z(\pi, \xi_i)$ is an unbiased estimator of $Z(\pi, \xi)$, by the central limit theorem, we know the average $\frac{1}{N} \sum_{i=1:N} Z(\pi, \xi_i)$ converges to the mean of $Z(\pi, \xi)$ as N goes to infinity. □

As the probability distribution of ξ is independent of π, we can obtain an approximate of $z(\pi)$ for different π using the same set of N samples. Therefore, we can solve the following optimization problem instead and use the solution of it as an approximate solution of the original problem (5.2).

$$\min_{\pi} \hat{z}_N(\pi) := \frac{1}{N} \sum_{i=1:N} Z(\pi, \xi_i) \quad s.t. \quad c(\pi) \leq \mathcal{B} \tag{5.4}$$

We also concern how close the (nearly) optimal solution of problem (5.4) is to the (nearly) optimal solution of problem (5.2). Let $\hat{\pi}_N$ be the optimal solution of (5.4) for N samples and π^* be the optimal solution of problem (5.2). We have the following results.

Theorem 5.2.2. *$\hat{x}_N$ converges to x^* as N goes to infinity.*

Proof. See paper [46]. □

5.2.1 Convergence Rates

Theorem 5.2.2 tells us that the optimal solution of problem (5.4) will converge to the optimal solution of problem (5.2) if we can increase N to infinite but not tell us how to

choose N in order to get a sufficiently good solution. In this section, we provide more information how to choose N, which can be summarized by the following Corollary.

Corollary 5.2.1. *If the size of the solution space is exponential of the problem size, to obtain a sufficiently good solution, the number of samples needs to be linear of the problem size.*

Now, we prove Corollary 5.2.1 and explain what it means by "a sufficiently good solution". For an $\epsilon > 0$, let S^ϵ be the set of ϵ-optimal solutions of problem (5.4), that is, $S^\epsilon = \{\pi \mid z(\pi) \leq z(\pi^*) + \epsilon\}$. For an $\delta \in [0, \epsilon)$, let $\hat{S}_N^\delta$ be the set of δ-optimal solutions of problem (5.2) for N samples, that is, $\hat{S}_N^\delta = \{\pi \mid \hat{z}_N(\pi) \leq \hat{z}(\hat{\pi}_N) + \delta\}$. Then, we have the following theorem.

Theorem 5.2.3. *For a value $\alpha \in (0, 1)$, the sample size N which is needed for the probability $Pr(\hat{S}_N^\delta \subseteq S^\epsilon)$ to be at least $1 - \alpha$ should satsify*

$$N \geq \frac{constant}{(\epsilon - \delta)^2} log\left(\frac{number\ of\ policies}{\alpha}\right)$$

Proof. The proof can be found in paper [46], which uses the large deviation theory. □

5.3 Sample Average Approximation for Stochastic Network Design

In this section, the sample average approximation (SAA) method is applied to solve stochastic network design problems for general directed graphs. The basic idea is as follows. First, a deterministic optimization problem in the form of (5.4) is constructed from a stochastic network design problem (5.1) by a sampling technique that is explained later. Second, the deterministic optimization problem is written as a *mixed integer program* that is then solved by a standard MIP solver. The rest of this section explains these two parts.

5.3.1 Sampling Procedure

The Monte Carlo sampling method is used to construct a deterministic optimization problem for problem (5.1). Recall that each edge has a survival probability which equals to p^o_{uv} if the action on the edge (u, v) is not taken and is p^m_{uv} if the action is taken. For a given policy π, the survival probability of an edge (u, v) is denoted by $p(u, v; \pi)$ or $p_{uv|\pi}$ for short. We can directly sample whether the edge is present or absent by flipping a biased coin with probability $p(u, v; \pi)$ if we know π. To be able to use SAA method, we want to draw samples independent from π. Remember, ξ represents all random events that are distributed independently from any π. We want

$$\mathbb{E}_\xi[f(\xi, \pi)] = R(\pi) \tag{5.5}$$

where $R(\pi)$ is the resilience of the network with the policy π defined by (2.4). Now, the important question is what ξ represents specifically in a stochastic network design problem.

To define ξ, let's first define a stochastic graph $\boldsymbol{G'} = (V, E', \boldsymbol{\xi})$ while $G = (V, E)$ is the input graph of the stochastic network design problem. There are two parallel edges e^o (the *original* edge) and e^m (the *modifed* edge) in E' for each edge $e \in E$ and two random variables $\boldsymbol{\xi}_{e^o}$ and $\boldsymbol{\xi}_{e^m}$ for each edge respectively. If $\boldsymbol{\xi}_{e^o} = 1$, the edge e^o is present. Otherwise, e^o is absent. Similarly, $\boldsymbol{\xi}_{e^m} = 1$, the edge e^m is present. Otherwise, e^m is absent. We further define a random graph $\boldsymbol{G'}(\pi) = (V, E'(\pi), \boldsymbol{\xi})$ parameterized by a policy π. This graph always includes the original edge and will include the modified edge if and only if the action on the edge e is taken or $a_e \in \pi$. Assuming that $(u, v) \in E$ and u, v are not connected by other edges, let's consider the random event that v is connected to u. If $a_{uv} \notin \pi$, the event occurs if and only if $\xi_{(u,v)^o}$ equals to 1. We want this probability to be p^o_{uv}. If $a_{uv} \in \pi$, the event occurs if and only if at least one of $\xi_{(u,v)^o}$ and $\xi_{(u,v)^m}$ equals to 1. We want this probability to be p^m_{uv}. 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 random variables $\boldsymbol{\xi}_{e^o}$ and $\boldsymbol{\xi}_{e^i}$:

$$\boldsymbol{\xi}_{e^o} = \begin{cases} 0 & \text{if } U_e > p_e^o \\ 1 & \text{if } U_e \leq p_e^o \end{cases} \qquad \boldsymbol{\xi}_{e^m} = \begin{cases} 0 & \text{if } U_e > p_e^m \\ 1 & \text{if } U_e \leq p_e^m \end{cases} \tag{5.6}$$

Now, we let $\xi = \{\boldsymbol{\xi}_{e^o}, \boldsymbol{\xi}_{e^m}\}$ of which the probability distribution is independent from any policy, so we can sample the values of $\boldsymbol{\xi}_{e^o}$ and $\boldsymbol{\xi}_{e^m}$ independently of any policy. First, a value of U_e is sampled. Then, we can determine the value of $\boldsymbol{\xi}_{e^o}$ and $\boldsymbol{\xi}_{e^m}$ easily. Now, we claim the following result.

Theorem 5.3.1. *For an arbitrary policy $\pi \subseteq E$ and two vertices $s, t \in V$, the probability that there is a directed path from s to t in $\boldsymbol{G}'(\pi)$ equals to the probability t is connected to s in G, the original stochastic network.*

Proof. Let's consider an arbitrary policy π and an edge $(u, v) \in E$. If we assume there are no other edges, the probability that u is connected to v is

$$Pr(u \rightsquigarrow v) = \begin{cases} Pr(\boldsymbol{\xi}_{(u,v)^o} = 1) & \text{if } a_{uv} \notin \pi \\ Pr(\boldsymbol{\xi}_{(u,v)^o} = 1 \text{ or } \boldsymbol{\xi}_{(u,v)^m} = 1) & \text{if } a_{uv} \in \pi \end{cases} \tag{5.7}$$

We show that if $a_{uv} \notin \pi$, $Pr(u \rightsquigarrow v) = p_{uv}^o$ and if $a_{uv} \in \pi$, $Pr(u \rightsquigarrow v) = p_{uv}^m$.

- Case 1: $a_{uv} \notin \pi$. Only edge $(u, v)^o$ exists in $\boldsymbol{G}'(\pi)$. The probability of $\boldsymbol{\xi}_{e^o}$ is p_{uv}^o.
- Case 2: $a_{uv} \in \pi$, both $(u, v)^o$ and e^i are present in $\boldsymbol{G}'(\pi)$. Then,

$$\begin{aligned} Pr(\boldsymbol{\xi}_{(u,v)^o} = 1 \text{ or } \boldsymbol{\xi}_{(u,v)^m} = 1) &= 1 - Pr(\boldsymbol{\xi}_{(u,v)^o} = 0 \text{ and } \boldsymbol{\xi}_{(u,v)^m} = 0) \\ &= 1 - Pr(U_e > p_{uv}^m) = Pr(U_e \leq p_{uv}^m) = p_{uv}^m \end{aligned}$$

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 only consider edges $(v_i, v_{i+1})^o$ and $(v_i, v_{i+1})^i$ if $a_{v_i v_{i+1}} \in \pi$ for all $i = 1 : n-1$. We ignore

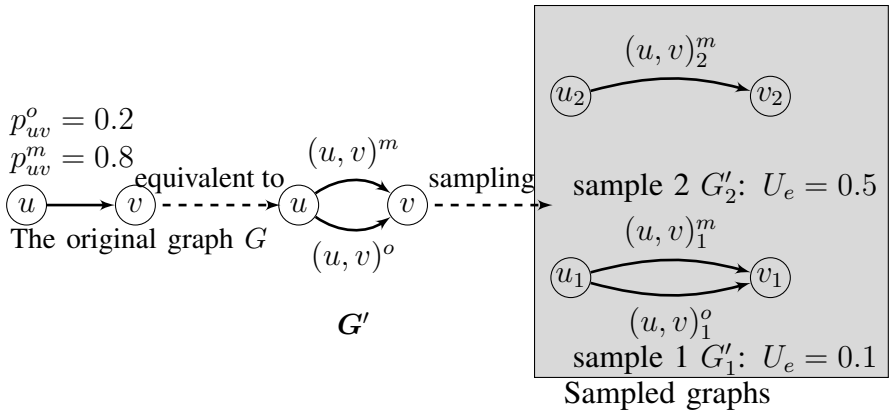


Figure 5.1: An example of sampling procedure. The orginal graph is G and the random graph is $\boldsymbol{G'}$. We have $V^d = \{u_1, u_2, v_1, v_2\}$, $E^d = \{(u,v)^o_1, (u,v)^m_1, (u,v)^m_2\}$, $E^d_{uv} = \{(u,v)^m_1, (u,v)^m_2\}$. $E^d_0 = \{(u,v)^o_1\}$.

all other edges in both graphs for now. Then, for an arbitrary policy π, the probability that v_n is connected to v_1 in the original stochastic network equals to the probability that v_n is connected to v_1 in $\boldsymbol{G'}(\pi)$ based on the previous claims. Thus, basically, these two graph gives equal probability for any path. So, the claim in the theorem holds. □

Let's define $Z(\pi, \boldsymbol{\xi})$ to be the total reward collected in the graph $G'(\pi)$ with a fixed value of $\boldsymbol{\xi}$, in which a reward r_{st} is collected if and only if there is a directed path from s to t. Then, we have

Corollary 5.3.1. *If*

$$Z(\pi, \boldsymbol{\xi}) = \sum_{s,t \in V} r_{st}\, \mathbb{I}\left[t \textit{ is connected to } s \textit{ in } \boldsymbol{G'}(\pi)\right],$$

then $\mathbb{E}_{\boldsymbol{\xi}}[Z(\pi, \boldsymbol{\xi})] \;=\; R(\pi)$ *for any policy* π.

Proof. The proof can be obtained from Theorem 5.3.1 directly. □

Sampling procedure can be summarized as follows. We generate N samples of $\boldsymbol{G'}$, $\{G'_1, ..., G'_N\}$ by sampling N values of U_e for each edge e using (5.6). In each G'_k, the presence or absence of each edge is known. See Fig. 5.1 for an example of the sampling

procedure. In the example, the original graph has two nodes u and v and an edge (u, v) with probabilities $p^o_{uv} = 0.2$ and $p^m_{uv} = 0.8$. The correspondent random graph $\boldsymbol{G'}$ has two parallel edges. We draw two samples. For the first sample G'_1, the randomly drawn value of U_{uv} is 0.1, so by (5.6), we end up having both e^o_{uv} and e^m_{uv} present. Similarly, for G'_2, the drawn value of U_{uv} is 0.5, so only e^m_{uv} is present.

5.3.2 Construction of Deterministic Optimization Problems

To construct the deterministic optimization problem, we first combine N sample graphs $\{G'_1, ..., G'_N\}$ into a single graph $G^d = (V^d, E^d)$ (e.g., d stands for "deterministic"). Let the node in G'_i indexed by v_i and edges denoted by e^o_i and e^m_i where v represents a node and e represents an edge in the orginal graph G respectively. The node set V^d contains nodes from all sample graphs. The edge set E^d contains the original edges and the modified edges in all sample graphs.

$$V^d = \{v_i |\ i = 1 : N, v_i \in G'_i\}$$
$$E^d = \{e^o_i |\ i = 1 : N, e^o_i \in G'_i\} \bigcup \{e^m_i |\ i = 1 : N, e^m_i \in G'_i\}$$

For each edge (u, v) in the orginal graph G, all modified edges in E^d corresponding to (u, v) are grouped into an edge set E^d_{uv}, that is, $E^d_{uv} = \{e^m_i |\ i = 1 : N, e^m_i \in G'_i, e = (u, v)\}$. An action a_{uv}, if taken, will purchase all edges in E^d_{uv} with cost c_{uv} . All original edges in E^d are grouped into an additional edge set E_0, that is, $E_0 = \{e^o_i |\ i = 1 : N, e^o_i \in G'_i\}$. An action a_0 with cost $c_0 = 0$ will purchase all edges in set E_0. Fig. 5.1 gives an example of these sets. Then, we form a deterministic optimization problem in which we purchase edge sets to maximize the average reward collected.

$$\max_{\pi} \frac{1}{N} \sum_{k=1}^{N} \sum_{s,t \in V} r_{st}\, \mathbb{I}[t \text{ is connected to } s \text{ in } G'_k(\pi)] \quad s.t. \quad \sum_{a_e \in \pi} c_e \leq \mathcal{B} \tag{5.8}$$

where $\mathbb{I}(\cdot)$ is an indicator function that returns 1 if the event inside is true and returns 0 otherwise. By Corollary 5.3.1 and Theorem 5.3.1, for a π, the objective of (5.8) converges to the objective of (2.5) as N goes to infinity. The problem (5.8) can be written as a MIP as I will show in next section, for which a standard MIP solver can be used to find the optimal or the nearly optimal solution.

5.4 Budget Set Weighted Directed Steiner Graph Problem

The deterministic optimization problem (5.8) can be formulated as a new network design problem called *Budget Set Weighted Directed Steiner Graph* (BSW-DSG) that can be written as a *mixed integer program* (MIP).

Broadly, in a *Budget Set Weighted Directed Steiner Graph* (BSW-DSG) problem, we want to decide how to purchase edges, which are grouped into sets, to connect a specified set of nodes in the network. The formal definition is as follows. The input of a BSW-DSG problem consists of a directed graph $G^d = \{V^d, E^d\}$ and a set of o-d pairs $\Theta = \{(o_1, d_1), ..., (o_T, d_T)\}$ ($o_i, d_i \in V^s$), each pair associated with a reward $r_{o_i d_i}$. Specifically, o_i represents the node of the origination of a flow, and d_i represents the node of the destination of the flow. We are also given a collection of edge sets $\mathbb{E} = \{E_1, ..., E_S\}$ where $E_i \subseteq E^d$ and each E_i is associated with a cost c_i. A subset $\mathcal{A} \subseteq \mathbb{E}$ corresponds to a subgraph $G^d(\mathcal{A}) = \{V^d, E^d(\mathcal{A})\}$ with $E^d(\mathcal{A}) = \cup_{E_i \in \mathcal{A}} E_i$. For a given $G^d(\mathcal{A})$, if d_i is connected to o_i for any pair (o_i, d_i), a reward $r_{o_i d_i}$ is collected. The goal is to purchase edge sets from $\mathbb{E}$, subject to a budget limit $\mathcal{B}$, to maximize the total rewards that are collected in $G^d(\mathcal{A})$. That is,

$$\max_{\mathcal{A} \subseteq \mathbb{E}} \sum_{(o,d) \in \Theta} r_{od} \cdot \mathbb{I}[d \text{ is connected to } o \text{ in } G^d(\mathcal{A})] \quad s.t. \quad \sum_{E_i \in \mathbb{E}} c_i \leq \mathcal{B} \tag{5.9}$$

where $\mathbb{I}$ is an indicator function.

To make later presentation clear, we prefer to write a BSW-DSG problem as a minimization problem. We define a value M that is greater than the reward of any pair, that is, $M \geq r_{od}$ for any $(o, d) \in \Theta$. Let $M_{od} = M - r_{od} \geq 0$, representing the penalty charged for not connecting the pair (o, d). Then, the problem is equivalent to

$$\min_{\mathcal{A} \subseteq \mathbb{E}} \sum_{(o,d) \in \Theta} M_{od} \cdot \mathbb{I}[d \text{ is } \textbf{not} \text{ connected to } o \text{ in } G^d(\mathcal{A})] \quad s.t. \quad \sum_{E_i \in \mathbb{E}} c_i \leq \mathcal{B} \tag{5.10}$$

I use (5.10) as the standard formulation of a BSW-DSG problem.

It is straightforward to write the deterministic optimization problem (5.8) as a BSW-DSG problem. In the BSW-DSG problem, we use the directed graph $G^d = \{V^d, E^d\}$ defined in section 5.3.2 as the input graph. The collection of edge sets consist of E_0 and all E^d_{uv}s defined in section 5.3.2. The cost of the edge set E_0 is 0 and the cost of the edge set E^d_{uv} is c_{uv}. To define Θ, for any pair of nodes (o, d) ($o \in V$, $d \in V$), if the reward r_{od} is not 0, we add a pair (o, d) into Θ with reward r_{od} and penalty $M_{od} = M - r_{od}$. So, the problem (5.8) is written as a BSW-DSG problem. The next sections provide techniques to solve a BSW-DSG problem in general.

5.4.1 Hardness of BSW-DSG

A BSW-DSG problem is a discrete optimization problem and is hard to solve.

Theorem 5.4.1. *A BSW-DSG problem, in general, 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-DSG problem. The Knapsack problem

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

(where p_i is profit and c_i is cost) is equivalent to

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

where $M_i = M - p_i > 0$ and M is bigger than all p_is.

To build the BSW-DSG problem for this Knapsack problem, we define (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 with $E_i = \{(o, d_i)\}$ with 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 BSW-DSG problem is equivalent to the Knapsack problem.

To show that the objective function is neither submodular nor supermodular, we use a simple graph as shown in Fig. 5.2 with $V = \{o, u, v\}$ and $E = \{e_1, e_2, e_3, e_4\}$. Let the unique o-d pairs be (o, d) with the penalty $M_{o,d} > 0$ and the collection of edge sets being $\mathbb{E} = \{\{e_1\}, \{e_2\}, \{e_3\}, \{e_4\}\}$. Let $f(\cdot)$ be the objective function. Given a set of edges, if o is connected to d, the value of the objective function is 0. Otherwise, the objective function equals to $M_{o,d}$. For example,

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

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.

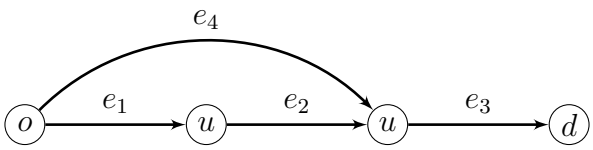


Figure 5.2: An example

$$\min \sum_{(o,d)\in\Theta} M_{o,d} z^{od} \tag{5.1}$$

$$s.t. \quad \sum_{e\in\delta^+(S)} x_e^{od} + z^{od} \geq 1 \quad \forall (o,d) \in \Theta, S \in \mathcal{S}_o^{od} \tag{5.2}$$

$$\sum_{i:e\in E_i} y_i \geq x_e^{od} \quad \forall (o,d) \in \Theta, \forall e \in E^s \tag{5.3}$$

$$\sum_{E_i\in\mathbb{E}} c_i y_i \leq \mathcal{B} \tag{5.4}$$

$$x_e^{od} \in [0,1] \quad \forall (o,d) \in \Theta \ \forall e \in E^s \tag{5.5}$$

$$y_i \in \{0,1\} \ \forall E_i \in \mathbb{E}, \qquad z^{od} \in \{0,1\} \ \forall (o,d) \in \Theta \tag{5.6}$$

Figure 5.3: MIP formulation of the budget BSW-DSG problem

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. □

5.5 Formulation of a BSW-DSG problem as a MIP

A BSW-DSG problem can be formulated as a mixed integer program (MIP) shown in Fig. 5.3. A binary variable x_e^{od} is defined for each edge $e \in E$ and o-d pair indicating whether e is on (x_e^{od} =1) a path from o to d or not (x_e^{od} =0). A binary decision variable y_i is defined for each edge set $E_i \in \mathbb{E}$ 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 (z^{od} =1) or not/connected (z^{od} =0). Let set $\mathcal{S}_o^{od}$ consist

of all subsets of V^d that contain o but not d, $\mathcal{S}_o^{od} = \{S | S \subseteq V^d \wedge o \in S \wedge d \notin S\}$. Let set $\delta^+(S)$ 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 (5.2), 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 (5.3), an edge is on the path from o to d only if it is purchased. Constraint (5.5) is the budget constraint. It is easy to show that relaxing the binary variable x to be a continuous variable in $[0, 1]$ will not change the value of the optimal solutions.

The MIP has an exponentially large number of constraints, but the ellipsoid method can be used to find the optimal solution. A standard solver, such as IBM CPLEX optimizer [38] or Gurobi [68], can be used to solve it directly.

5.6 A Fast Algorithm to Solve BSW-DSG Problems

Although a BSW-DSG problem can be formulated as a MIP, directly solving it by a MIP solver can only scale to networks of small sizes and a small number of samples as shown in experimental results later. In this section, I will introduce a much faster algorithm to find high-quality solutions. By studying a BSW-DSG problem shown in Fig. 5.3, I observed that it is the budget constraint that causes the problem very difficult to be solved by a MIP solver. To avoid dealing with this constraint directly, I instead solve a Lagrangian relaxation problem constructed by a method called *Lagrangian relaxation*. The nearly optimal solution of the Lagrangian relaxation problem is used as a solution to the original problem.

The Lagrangian relaxation technique has been used to solve many constrained discrete optimization problems [2, 39, 53]. The basic idea is to relax a complex constraint and bring it to the objective together with a Lagrangian multiplier β. The new optimization problem is called *relaxation problem* parameterized by β. For the BSW-DSG problem, the relaxation problem is a prize-collecting version of the BSW-DSG called *Prize-Collecting Set Weighted Directed Steiner Graph* (PCSW-DSG), of which the objective function consists of both the

objective function of the BSW-DSG problem and the cost of the policy weighted by a parameter β that serves to be a tradeoff factor. Our definitions of the BSW-DSG problem and the PCSW-DSG problem are the generalizations to the budget and the prize-collecting Steiner tree variants [41]. The MIP formulation of the PCSW-DSG problem is shown in Fig. 5.4 (5.1-5.5)

Due to the special structure of the relaxation problem as explained in the next section, a bisection procedure can be used to find a β, with which by solving the relaxation problem, we can get a nearly optimal solution to the original BSW-DSG problem.

In summary, the algorithm of solving a BSW-DSG problem consists of three steps. First, we build a relaxation problem that is parametrized by a β. Then, we use a bisection procedure to find a value of β. At last, we solve a PCSW-DSG problem with that $beta$ and output the found solution.

5.6.1 Bisection Procedure

Let's first analyze the function $L(\beta)$ in Fig. 5.4. Let $(x_\beta, y_\beta, z_\beta)$ denote the optimal solution of the PCSW-DSG problem for a given β. Let, $Z(\beta) = \sum_{(o,d)\in\Theta} M_{o,d} z_\beta^{od}$, $C(\beta) = \sum_{E_i\in\mathbb{E}} c_i y_{i\beta} - \mathcal{B}$, and $L(\beta) = Z(\beta) + \beta C(\beta)$. We have the following properties.

Proposition 5.6.1. *As β increases, $Z(\beta)$ is nondecreasing and $C(\beta)$ is nonincreasing.*

Proof. Intuitively, larger β puts larger penalty on the cost and thus results in a less costly y but larger $Z(\beta)$. To prove Proposition 5.6.1 formally, I first prove the following claim.

Claim: If $\beta_1 < \beta_2$, then $Z(\beta_1) \leq Z(\beta_2)$ and $C(\beta_1) \geq C(\beta_2)$.

Let's use the following two notations.

$$Z(z) = \sum_{(o,d)\in\Theta} M_{o,d} z^{od} \text{ and } C(y) = \sum_{E_i\in\mathbb{E}} c_i y_{i\beta} - \mathcal{B} \tag{5.11}$$

where both z and y are two vectors.

Prize-Collecting Problem / Primal Problem:

$$L(\beta) = \min \sum_{(o,d)\in\Theta} M_{o,d} z^{od} + \beta \left(\sum_{E_i \in \mathbb{E}} c_i y_i - \mathcal{B} \right) \tag{5.1}$$

$$s.t. \quad \sum_{e\in\delta^+(S)} x_e^{od} + z^{od} \geq 1 \quad \forall (o,d) \in \Theta, S \in \mathcal{S}_o^{od} \tag{5.2}$$

$$\sum_{i:e\in E_i} y_i \geq x_e^{od} \quad \forall (o,d) \in \Theta, \forall e \in E^s \tag{5.3}$$

$$x_e^{od} \in [0,1] \quad \forall (o,d) \in \Theta \ \forall e \in E^s \tag{5.4}$$

$$y_i \in \{0,1\} \ \forall E_i \in \mathbb{E}, \qquad z^{od} \in \{0,1\} \ \forall (o,d) \in \Theta \tag{5.5}$$

where (5.2), (5.3), (5.4), (5.5) are basically constraints (5.2), (5.3), (5.5), (5.6) in Fig. 5.3 respectively

Dual Problem:

$$\max \sum_{(o,d)\in\Theta, S\in\mathcal{S}_o^{od}} \mu_S^{od} \tag{5.6}$$

$$s.t. \sum_{S:e\in\delta^+(S), S\in\mathcal{S}_s^{od}} \mu_S^{od} - \lambda_e^{od} \leq 0 \quad \forall (o,d) \in \Theta, \forall e \in E \tag{5.7}$$

$$\sum_{S:S\in\mathcal{S}_s^{od}} \mu_S^{od} \leq M_{o,d} \quad \forall (o,d) \in \Theta \tag{5.8}$$

$$\sum_{(o,d)\in\Theta} \sum_{e\in E_i} \lambda_e^{od} \leq \beta c_i \quad \forall E_i \in \mathbb{E} \tag{5.9}$$

$$\mu_S^{od} \in [0,\infty) \quad \forall (o,d) \in \Theta, S \in \mathcal{S}_o^{od} \tag{5.10}$$

$$\lambda_e^{od} \in [0,\infty) \quad \forall (o,d) \in \Theta, e \in E \tag{5.11}$$

Figure 5.4: Primal and dual of the PCSW-DSG problem

Let $(x_{\beta_1}, y_{\beta_1}, z_{\beta_1})$ and $(x_{\beta_2}, y_{\beta_2}, z_{\beta_2})$ be two minimizers for parameters β_1 and β_2 respectively.

Assume that $Z(z_{\beta_1}) > Z(z_{\beta_2})$. It implies $C(y_{\beta_1}) < C(y_{\beta_2})$. Otherwise, $Z(z_{\beta_1}) + \beta_1 C(y_{\beta_1}) > Z(z_{\beta_2}) + \beta_1 C(y_{\beta_2})$ such that $(x_{\beta_2}, y_{\beta_2}, z_{\beta_2})$ is a better solution for β_1 which contradicts $(x_{\beta_1}, y_{\beta_1}, z_{\beta_1})$ being a minimizer for β_1.

Since $(x_{\beta_2}, y_{\beta_2}, z_{\beta_2})$ is a minimizer for β_2, we have

$$Z(z_{\beta_2}) + \beta_2 C(y_{\beta_2}) \leq Z(z_{\beta_1}) + \beta_2 C(y_{\beta_1}) \tag{5.12}$$

$$\Rightarrow Z(z_{\beta_1}) - Z(z_{\beta_2}) \geq \beta_2 \left(C(y_{\beta_2}) - C(y_{\beta_1}) \right) > \beta_1 \left(C(y_{\beta_2}) - C(y_{\beta_1}) \right) \tag{5.13}$$

$$\Rightarrow Z(z_{\beta_1}) + \beta_1 C(y_{\beta_1}) > Z(z_{\beta_2}) + \beta_1 C(y_{\beta_2}) \tag{5.14}$$

where (5.13) is true because we proved earlier by contradiction that $C(y_{\beta_2}) - C(y_{\beta_1}) > 0$ and $\beta_1 < \beta_2$.

(5.14) indicates $(x_{\beta_2}, y_{\beta_2}, z_{\beta_2})$ is a better solution for β_1 that contradicts that $(x_{\beta_1}, y_{\beta_1}, z_{\beta_1})$ is the minimizer for β_1. Thus, my first assumption that $Z(z_{\beta_1}) > Z(z_{\beta_2})$ is wrong, so we have $Z(z_{\beta_1}) \leq z(z_{\beta_2})$.

$Z(z_{\beta_1}) \leq Z(z_{\beta_2})$ also implies $C(y_{\beta_1}) \geq C(y_{\beta_2})$. Otherwise, if $C(y_{\beta_1}) < C(y_{\beta_2})$, we get $Z(z_{\beta_1}) + \beta_2 C(y_{\beta_1}) < Z(z_{\beta_2}) + \beta_2 C(y_{\beta_2})$ that says $(x_{\beta_1}, y_{\beta_1}, z_{\beta_1})$ is a better solution for β_2 which is a contradication.

In summary, we have $Z(\beta_1) = Z(z_{\beta_1}) \leq Z(z_{\beta_2}) = Z(\beta_2)$ and $C(\beta_1) = C(y_{\beta_1}) \geq C(y_{\beta_2}) = C(\beta_2)$. □

The bisection procedure starts with a suitably large interval bounding β and then narrows it iteratively. At each iteration, a new β is picked as the middle point of the current range and then the primal-dual algorithm is used to calculate $C(\beta)$ and produce a near-optimal strategy for β. By Proposition 5.6.1, if $C(\beta) < \mathcal{B}$, all β' greater than β will give $Z(\beta') \geq Z(\beta)$ and therefore can be abandoned. If $C(\beta) > \mathcal{B}$, all β' smaller than β will give $C(\beta') \geq C(\beta)$ and therefore can be abandoned. The procedure terminates when the range is less than some threshold, and a β in the range along with the computed strategy is returned.

Proposition 5.6.1 is true when the PCSW-DSG problem is solved optimally for each β, but the bisection procedure is still valid with our approximate algorithm. Let $(\hat{x}_\beta, \hat{y}_\beta, \hat{z}_\beta)$ denote the near-optimal solution computed by the primal-dual algorithm for some β. And let $\hat{Z}(\beta) = \sum_{(r,k')\in E^{\mathcal{T}}} \hat{z}_{rk'\beta}$ and $\hat{C}(\beta) = \sum_{s=1}^{M} c_s \hat{y}_{s\beta}$. Then, we have

Proposition 5.6.2. *Suppose terminal vertices in line 6 of Algorithm 1 are chosen in a predefined order, as β increases, $\hat{Z}(\beta)$ is nondecreasing and $\hat{C}(\beta)$ is nonincreasing.*

Proof. Intuitively, as β becomes larger, the equality in (5.9) in the dual problem becomes harder to satisfy. Since the terminal vertices are picked in the same order over different β,

Algorithm 1 Primal-Dual Algorithm for PCSW-DSG

function PRIMALDUAL($G^d, \mathcal{E}, \beta$)
  Set $y_i \leftarrow 0 \ \forall E_i \in \mathbb{E}$ and $z^{od} \leftarrow 0 \ \forall (o,d) \in \Theta$
  $F \leftarrow \phi$
  For each pair (o,d), the set C^{od} contains all vertices reachable from o in (V^d, F)
  Initially, $C^{od} = \{o\}$ for each (o,d)
  Mark all o-d pairs as *active*
  while there exists some *active* pairs **do**
    Increase $\mu^{od}_{C^{od}}$ and λ^{od}_e for all *active* pairs simultaneously **until**
    if (5.8) in Fig. 5.4 becomes tight for (o,d) **then**,
      Mark (o,d) *abandoned* and set $z^{od} = 1$
    else ▷ (5.9) in Fig. 5.4 becomes tight for E_i
      set $y_i = 1$ and $F \leftarrow F \cup e$
      for each *active* (o,d) **do**
        expand C^{od} by e
      end for
      if C^{od} contains d for some (o,d) **then**
        Mark (o,d) *connected*
      end if
    end if
  end while
  ▷ Remove unused edge sets
  for each E_i with $y_i = 1$ **do**
    if remove edges in E_i from F, all *connected* pairs are still connected **then**
      Remove E_i and $y_i \leftarrow 0$
    end if
  end for
  return y
end function

the set of edge sets purchased for a larger β is the subset of edge sets for a smaller β which gives us Proposition 5.6.2. □

Finally, if the primal-dual algorithm performs near-optimally for the PCSW-DSG problem, the final solution computed by the bisection procedure will be near optimal for the BSW-DSG problem. Although this procedure is not accompanied by a formal approximation guarantee, we observe that it produces near-optimal results on the problem instances used in our experiments.

5.6.2 Solving the Prize-Collecting Problem

I provide a primal-dual algorithm shown in Algorithm 1 to solve the prize-collecting problem. The algorithm borrows ideas from the primal-dual algorithms of the generalized Steiner tree problem [96]. 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 [94].

Definition 5.6.1. ***Primal Complementary Slackness Condition***

$$x_e^{od} = 1 \Longrightarrow \sum_{S:e\in\delta^+(S),S\in\mathcal{S}_s^{od}} \mu_S^{od} - \lambda_e^{od} = 0$$

$$z^{od} = 1 \Longrightarrow \sum_{S:S\in\mathcal{S}_s^{od}} \mu_S^{od} = M_{o,d}$$

$$y_i = 1 \Longrightarrow \sum_{(o,d)\in\Theta} \sum_{e\in E_i} \lambda_e^{od} = \beta c_i$$

Namely, $x_e^{od} = 1$*,* $z^{od} = 1$ *and* $y_i = 1$ *imply equality in constaints (5.7), (5.8) and (5.9) in Fig. 5.4 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^d, F) where $F = \phi$ and a feasible dual solution where all μ and λ are 0. The graph (V^d, 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 (5.2) in Fig. 5.4. Note that (5.3) is always satisfied as we proceed. A pair is *active* meaning that constraint (5.2) of this pair is violated for some S. To satisfy (5.2) 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 (5.2) 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 (5.7) 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 (5.8) becomes tight for some (o, d), by the slackness condition, we set $z^{od} = 1$ and never consider this pair again because all constraints at (5.2) that involve this pair are satisfied (line 10). If constraint (5.9) 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 22-26, the unused edge sets are removed.

5.6.3 Complexity

Proposition 5.6.3. *If we assume that* $|\mathbb{E}| \leq |E|$ *and* $|\Theta| \leq |E|$*, the runtime of Algorithm 1 is bounded by* $O(|\Theta||E|)$.

Proof. The algorithm takes at most $(|E|+|\Theta|)$ iterations, because at each iteration, either an edge is added into F or a pair becomes inactive (abandoned or connected). When all edges are in F, the loop terminates too. In line 8, we calculate the minimum amount that the variable μ of all active C^{od} can increase. To do this,

To do this, at each iteration, we don't need to worry about constraint (5.7). We only need to check constraint (5.8) for all pairs and (5.9) for all edge sets in Fig. 5.4. It takes $O(|\Theta|+|\mathbb{E}|)$ In total, the line 8 takes $O((|E|+|\Theta|) \cdot (|\Theta|+|\mathbb{E}|))$.

Line 14 will only take $O(|E||\Theta|)$ in total because each edge will be expanded at most once for each pair. During the prunning, checking connectivity only takes $O(|\mathbb{E}|)$ time. Except these lines, other lines take insignificant runtime. Thus, for a connected network where $|V| \leq |E|$, the total runtime will be $O((|E|+|\Theta|) \cdot (|\Theta|+|\mathbb{E}|))$. □

5.6.4 Improvements

Several techniques can improve the quality of a solution produced by Algorithm 1.

5.6.4.1 Local Search

We found it useful to do a local search, with 10-20 iterations, in a small range around the β value returned by the bisection procedure to further improve the solution using random orders of picking terminals. We note, however, that the improvement is limited, indicating that the order of picking terminals doesn't affect the quality that much.

5.6.4.2 Greedy Padding

Since we find a near-optimal β rather than an optimal one, the computed solution may not use all the available budget. Therefore, to improve the solution, a greedy procedure can be used to continue purchasing edge sets until the budget is exhausted. In our experiments, only several additional edge sets can be purchased, so the procedure is very fast, but the improvement is not significant, usually 1-2%.

5.7 Case Study: Conservation Planning Problem

Spatial conservation planning problems have received significant attention in the AI community, resulting in a range of strategies for conserving land parcels in order to support the recovery of an endangered species or preserve biodiversity. Sheldon *et al.* [87] study a restricted version of this problem—*Red-cockaded Woodpecker* (RCW)—in which the planner selects a set of land parcels, subject to a budget constraint, to maximize the spread of a population over a geographical network of land patches within a specific time horizon. The underlying spreading process is modeled as a network diffusion process or cascade [42] using a *metapopulation model* developed by ecologists. Due to the stochasticity of process, the RCW problem can be written as a stochastic optimization problem. I apply my algorithm to solve the RCW problem. Remember that the basic idea of the algorithm is to first construct the deterministic optimization using the sampling technique and then solve the deterministic problem using the bisection procedure and the primal-dual algorithm. Sheldon *et al.* [87] use a different sampling technique to convert the RCW prob-

lem into a MIP and use a MIP solver to solve the MIP. Their method can only deal with acyclic directed graphs while the RCW problems are defined on acyclic directed graphs. I observe that my sampling procedure degenerates into their sampling procedure in the RCW problem.

In the rest of this section, I formally define the RCW problem, formulate it as a stochastic network design problem, and show the experimental results of applying my algorithm to the problem.

5.7.1 Problem Settings

Consider a conservation area consisting of habitat patches that are the atomic units in the population dynamics model and can be either occupied or unoccupied by the species. These patches are grouped into non-overlapping *parcels* $\mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_L$, which are the smallest units available for purchase. A parcel $\mathcal{P}_i$ can be purchased with cost c_i used to restore and conserve its habitat patches so they are suitable for the species to occupy. A patch can be occupied only if the parcel that it belongs to is purchased. A conservation strategy is an L-dimensional vector y where $y_l = 1$ if that the l^{th} parcel is purchased and $y_l = 0$ otherwise. Let $\mathcal{P}$ be the set of parcels and Pa be the set of patches.

Sheldon *et al.* [87] use a *metapopulation model* from ecology [34] to describe the stochastic occupancy dynamics of the species. In this model, a patch is either occupied or unoccupied at each time step $h \in \{0, 1, ..., H-1\}$. At time 0, the species occupy a set of patches called *sources*. At any time h, an occupied patch v triggers two stochastic events: (1) the population at u colonizes an unoccupied patch v with probability P^c_{uv} (typically decays with distance between the patches) that makes v occupied at time $h+1$; and (2) the population at v becomes extinct with probability P^e_v, making v unoccupied at time $h+1$.

The goal of conservation planning is to select a set of parcels to purchase, without exceeding a budget $\mathcal{B}$, such that the expected number of patches being occupied at time

$H-1$ is maximized. More concretely, given a strategy y, define 0-1 random variable $X(y)_{v,h}$ for each patch v and each time h to indicate whether the patch v is occupied at time h ($X(y)_{v,h}=1$) or not ($X(y)_{v,h}=0$). The best strategy is obtained by solving the following optimization problem:

$$\arg\max_{y} \sum_{v} \mathbb{E}[X(y)_{v,H-1}] \quad s.t. \sum_{l:y_l=1} c_l \leq \mathcal{B} \tag{5.15}$$

5.7.2 Formulated as a Stochastic Network Design Problem

The diffusion process of the species can be written as a layered graph as shown in Fig. 5.5. The graph contains a node v_h for patch v and time h. For a pair of patches (u,v) ($u \neq v$) and a $h \in \{0,\ldots,H-2\}$, the graph contains a directed edge (u_h, v_{h+1}), meaning that v may be occupied at $h+1$ when u is occupied at h. For a patch v and $h \in \{0,\ldots,H-2\}$, the graph contains a directed edge (v_h, v_{h+1}), meaning that the population occupying v at h may survive to time $h+1$ with some probability. The graph also contains a unique node s as the dummy source and it has a directed edge to any v_0 if v is initially occupied, for example, u and v are the initially occupied nodes in the Fig. 5.5.

Now, we set the original survival probability and the modified survival probability. Note that the management actions in this problem are to purchase land units. Therefore, for any patch v, we set $p^o_{u_h,v_{h+1}}=0$ for all $h \in \{0,\ldots,H-2\}$ and $p^o_{s,v_0}=0$ if the edge (s,v_0) exists, meaning that before the patch v is purchased, all edges going into this patch are blocked. In contrast, we set $p^m_{u_h,v_{h+1}}=p^c_{uv}$ if $u \neq v$ and $p^m_{u_h,v_{h+1}}=1-p^e_v$ if $u=v$ for all $h \in \{0,\ldots,H-2\}$ and $p^m_{s,v_0}=1$ if the edge (s,v_0) exists, meaning that after the patch v is purchased, all edges going into this patch are passable. To configure the actions, for each parcel $\mathcal{P}_i$, an action a_i is defined to raise the probability of any edge (u_h, v_{h+1}) with $v \in \mathcal{P}_i$ from $p^o_{u_h,v_{h+1}}$ to $p^m_{u_h,v_{h+1}}$. For example, in Fig. 5.5, $\mathcal{P}_1$ contains patches u and v, so the action a_1 can all edges related to u and v to 1. Specifically, an action affacts edges in the set

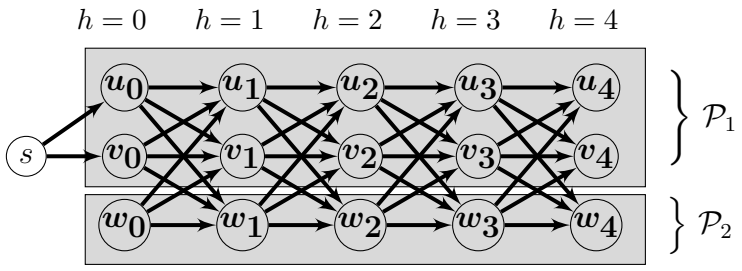


Figure 5.5: An example of the input directed graph of the stochastic network design problem that models the RCW problem, in which there are three patches u, v, w, two parcels $\mathcal{P}_1, \mathcal{P}_2$ and 5 time steps.

$$\{u_h, v_{h+1} | u, v \in \mathcal{P}_i, h \in \{0, \ldots, H-2\}\} \bigcup \{s, v_0 | v \in \mathcal{P}_i, \text{ v is occupied at time } 0\}$$

At last, we configure the reward function. Obviously, this is a single flow problem. Therefore, we set $r_{s,v_{H-1}} = 1$ for any patch v and the reward of other pairs to be 0, meaning that we only care how many patches can be reached by birds at the last time step.

With this configuration, we constructed an instance of the stochastic network design problem which is equivalent to the RCW problem, and we see the graph is a directed acyclic graph.

5.7.3 Sampling Procedure and the MIP Formulation

The stochastic optimization problem (5.15) is very hard to solve directly, so the SAA method discussed in Section 5.3 is used. We sample N independent sample graphs using the procedure discussed in Section 5.3.1, each of which represents a possible outcome of the stochastic process. Since the original survival probabilities are all 0, all original edges e^o are absent. For the modified edges e^m, we sample a value of U_e and determine their the presences.

More concretely, one sample graph $G'_k(\pi)$ is a layered graph, as shown in Fig. 5.6, that contains a node v_h for any patch v and any time step h. The edges in each graph $G'_k(\pi)$ are determined by the sampling procedure, that is, $G'_k(\pi)$ contains all modified edges that are sampled to be present. With a conservation strategy or a policy π, we can determine

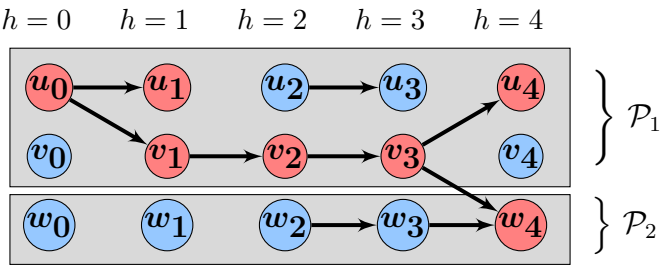


Figure 5.6: Example of the layered graph of a scenario with three patches u, v, w and two parcels $\mathcal{P}_1, \mathcal{P}_2$, showing parcels (grey boxes), occupied (red) and unoccupied (blue) patches.

whether v is occupied at h by checking whether there exists a *valid path* from some u_0 to v_h such that (1) u is occupied at time 0 and (2) all patches on the path are purchased. Then, the discrete optimization problem can be written as

$$\max_{\pi \subseteq \mathcal{A}} \frac{1}{N} \sum_{k=1}^{N} \sum_{v_{H-1} \in \text{ set of patches}} \mathbb{I}[v \text{ is connected to } s \text{ in } G'_k(\pi)] \quad s.t. \quad \sum_{e \in \pi} c_e \leq \mathcal{B} \tag{5.16}$$

which is the same as (5.8) but customized to the RCW problem by removing pairs of the rewards being 0.

This problem can be formulated as a compact MIP by [87] so that the standard MIP solver runs more efficiently. Let the variable $x^k_{v,H-1}$ to denote whether the node v_{H-1} is connected to s in k^{th} sample graph. Let the binary variable y_i to denote whether the parcel $\mathcal{P}_i$ is purchased or not. Constraint (5.2) basically says that the total cost of purchased parcels is no greater than the budget limit. Constraint (5.3) says that a node $x^k_{v,h}$ is connected to s only if the correspondent patch v is in some purchased parcels. Constraint (5.4) says that a node $x^k_{v,h}$ is connected to s only if a node that has an directed edge to the node is connected to s. Constraint (2.5) requires the value of each x variable to be in range $[0, 1]$ instead of being in $\{0, 1\}$. It can be shown that in the optimal solution, the value of any $x^k_{v,h}$ is either 0 or 1. The MIP formulation in Fig. 5.7 is more compact than the general MIP formulation of the BCSW-DSG problem in Fig. 5.3. So, in the RCW problem, if we use the standard MIP solver, we use it to solve MIP in Fig. 5.7.

$$\max_{x,y} \frac{1}{N} \sum_{k=1}^{N} \sum_{v \in Pa} x_{v,H-1}^{k} \tag{5.1}$$

$$s.t. \quad \sum_{\mathcal{P}_i \in \mathcal{P}} c_i y_i \leq \mathcal{B} \tag{5.2}$$

$$x_{v,h}^{k} \leq \sum_{i: v \in \mathcal{P}_i} y_i \quad \forall k = 1:N, \forall v \in Pa, \forall h = 0:H-1 \tag{5.3}$$

$$x_{v,h+1}^{k} \leq \sum_{(u_h, v_{h+1}) \in G'_k} x_{u,h}^{k} \quad \forall k = 1:N, \forall v \in Pa, \forall h = 0:H-1 \tag{5.4}$$

$$0 \leq x_{v,h}^{k} \leq 1 \quad \forall k = 1:N, \forall v \in Pa, \forall h = 0:H-1 \tag{5.5}$$

$$y_i \in \{0,1\} \quad \forall \mathcal{P}_i \in \mathcal{P} \tag{5.6}$$

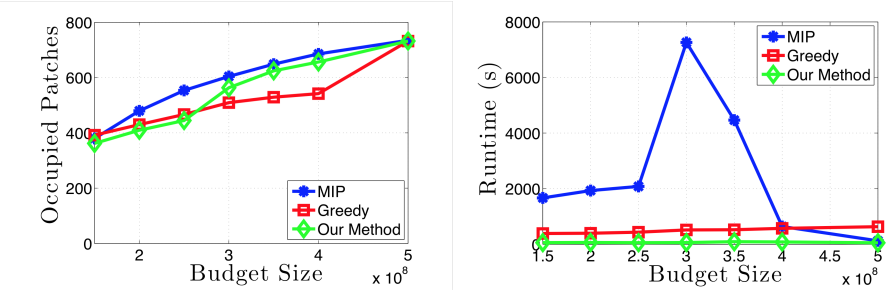
Figure 5.7: A compact MIP formulation for the RCW problem

5.7.4 Experimental Results

We used the data set for the RCW problem introduced by [87]. The study area includes 2537 patches and 443 parcels. The data set specifies the colonization probabilities between all pairs of patches, the grouping of patches into parcels, the prices of parcels and the initial occupancy status of patches. We used a planning horizon of 100 years and compared our algorithm with a MIP solver and with a greedy algorithm. In each iteration, the greedy algorithm chooses the edge set with the highest ratio of increase in objective value to cost, which gives better solutions than another greedy variant that chooses the edge set with the highest increase in objective value. We used the Gurobi Optimizer as the MIP solver [31]. All the experiments were run on a 2.2GHz Intel Core i7 CPU with 16GB of RAM.

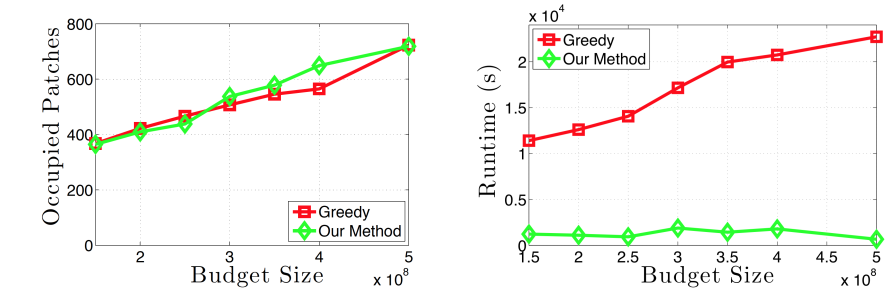
5.7.4.1 Small Sample Size

We compare three algorithms for 10 samples and the results are shown in Fig. 5.8. For our algorithm, we applied Algorithm 1 twice, once using the forward method and once using the backward method, and then applied the local search and greedy padding techniques to the better of those two solutions. In Fig. 5.8(b), our algorithm runs significantly faster than MIP and 10–20 times faster than the greedy algorithm. In Fig. 5.8(a), both our algo-



(a) Optimality w.r.t. budget (b) Runtime w.r.t. budget

Figure 5.8: Optimality and runtime versus budget with 10 samples



(a) Optimality w.r.t. budget (b) Runtime w.r.t. budget

Figure 5.9: Optimality and runtime versus budget with 300 samples

rithm and the greedy algorithm produce near-optimal solutions. Our algorithm outperforms the greedy algorithm on larger budget sizes and performs slightly worse on smaller budget sizes.

5.7.4.2 Large Sample Size

We also tested our algorithm and the greedy baseline using 300 samples. The results are show in Fig. 5.9. We did not test Gurobi because it fails to solve MIPs with more than 20 samples by either using too much time or running out of memory. Our algorithm performs almost identically to the greedy algorithm when the budget is smaller than 3×10^8 and outperforms the greedy algorithm when the budget is greater than 3×10^8. Fig. 5.9(b) shows that our algorithm is 10–30 times faster than the greedy algorithm and its running time is almost constant with respect to budget, while the greedy algorithm takes longer for larger

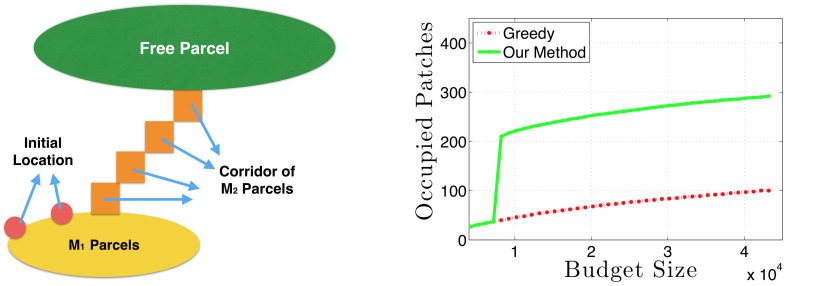
budgets. This implies that our algorithm can scale to larger network sizes or more samples, that latter of which leads to improved solution quality within the SAA methodology.

5.7.4.3 Synthetic Data

In the RCW problem, even though the discrete optimization problem is NP-hard, the greedy algorithm is still able to produce a solution within 80% of the optimal value, which suggests that the problem does not badly violate submodularity. In this section, we design a group of problems that are more challenging. These synthetic problems are motivated by the corridor design problem, which is another important conservation planning problem [29] in which the goal is to purchase a subset of parcels to build a (long) corridor connecting distant habitat areas in a fragmented landscape. When formulated within our context, the objective function violates submodularity because purchasing individual parcels has no or little effect on the objective function until the entire corridor is purchased, at which point the endpoints become connected, and there is a large jump in the objective value.

With this motivation, we construct a small problem instance as follows (see Fig. 5.10(a)). We first create M_1 parcels, each of which are directly connected to the source population and so carry an immediate reward of p_1 if purchased. We also create a free parcel which has reward of p_2 if it is reachable from the source. The free parcel is connected by a corridor of M_3 parcels, which have reward of p_3 if accessible. All parcels in the corridor must be purchased to realize the reward of the free parcel. If we set $p_2 \gg p_1 > p_3$, the optimal strategy should first purchase all parcels in the corridor whenever the budget allows, and first purchase all of the M_1 parcels if the budget is not enough to purchase the corridor.

The experimental results of our algorithm and the greedy algorithm on the synthetic data are shown in Fig. 5.10(b). Our algorithm gives the optimal solution as its curve overlaps with the curve of the MIP solver. When the budget is not enough (less than 7500) to purchase all parcels in the corridor, the greedy algorithm performs optimally. But it per-



(a) Corridor design problem (b) Optimality w.r.t. budget

Figure 5.10: Experiments on synthetic data

forms poorly when the budget is enough (greater than 7500) because it never purchases any parcels in the corridor before all M_1 parcels are purchased. From these results, we observed that when the objective function is not submodular as in our simple problem setting, the greedy algorithm may produce arbitrarily bad solutions while our algorithm can still produce near-optimal solutions.

5.8 Summary

In this chapter, I study the stochastic network framework under general directed graphs. One problem within the framework, in general, is at least #P-hard, so we try to design an approximate algorithm and find nearly optimal solutions of the problem. The basic idea is to construct a deterministic optimization problem using an SAA method and use the solution to the deterministic optimization problem as the approximate solution. The deterministic optimization problem is also NP-hard, and its size increases along with the number of samples and the size of the network. A fast algorithm combining a bisection procedure and a primal-dual algorithm is developed to produce high-quality solutions. At last, the algorithm is applied to solve a conservation planning problem called the RCW problem. The algorithm runs about 120 times faster than a MIP solver and up to 30 times faster than a greedy algorithm. The solutions produced by the algorithm also are nearly optimal and slightly better than the solutions produced by the greedy algorithm.

CHAPTER 6

STOCHASTIC NETWORK DESIGN FOR DISTANCE MINIMIZATION

In Chapter 5, the goal of a network design problem is to improve the connectivity of a set of specified node pairs in a network, for which it is assumed that the lengths of all edges equal to one unit. The uncertainty of an edge is whether it is present or absent. In this chapter, we study a more general setting in which both the connectivity and the distance between these node pairs, so we allow an edge to have an arbitrary length in the graph. Since the length of an edge may not be known beforehand, we use a probability distribution to describe it. An action can be taken to change the probability distribution and make the length of an edge shorter.

One motivating problem of this setting is the emergency medical service (EMS) response time minimization problem. In the problem, we are given a road network and a set of origin-destination (o-d) node pairs. An origin node represents an ambulance dispatch location, and a destination node presents a patient location. During natural disasters such as flood or earthquakes, a road may be damaged which makes the road impossible for traffic to get across or incurs traffic some delays to traffic. The response time of EMS is increased by either case. To keep the response time within an acceptable level, people selectively reinforce some roads before a natural disaster to reduce the travel time on the road when the disaster happens. Here is the optimization problem that a decision maker needs to solve. Given a limited amount of budget, which road segments should we select to reinforce such that the average EMS response time between a set of node pairs is minimized.

To solve the problem, we define a more general network design framework called *stochastic network design with arbitrary edge lengths* (SND-AEL) and formulate the EMS

response time minimization problem as an SND-AEL problem. The SND-AEL framework extends the stochastic network design framework in the following ways. Each edge has a length randomly distributed in a range $[0, \infty]$, where ∞ equivalently means the edge doesn't exist. An action can be taken to change the probability distribution of the length, for example, to make the length *stochastically shorter*, which is explained later. For each disconnected o-d pair (e.g., distance is ∞), a penalty M_{uv} is charged. The SND-AEL problem is to decide which actions to take subject to a budget limit to minimize both the distance between the connected o-d pairs and the total penalties charged.

A SND-AEL problem, in general, is harder to solve than a stochastic network design problem under a general directed graph. To develop a scalable algorithm, the same basic idea in Chapter 5 is used. First, a sample average approximation (SAA) method is conceived to construct a deterministic optimization problem. Then, a fast algorithm is developed to solve the deterministic optimization problem, and the obtained solution is used as an approximate solution to the original problem. The SAA method and the fast algorithm are different from the ones used in Chapter 5.

The structure of this chapter is as follows. First, the SND-AEL framework is defined and more details about the EMS response time minimization problem are provided. Then, the sampling procedure and the fast algorithm are introduced. At last, the experimental results of applying the algorithm to the EMS response time minimization problem are presented.

Both the sampling procedure and the fast algorithm extend the ones discussed in Chapter 5. If you already read Chapter 5, you may find some texts in this chapter are repetitive, which are still included to make this chapter self-contained. The difference between the algorithm for SND-AEL and the algorithm introduced in Chapter 5 are mentioned in this chapter..

6.1 Stochastic Network Design with Arbitrary Edge Lengths

This section defines the SND-AEL framework and explains how the framework is a more general framework than the stochastic network design framework.

6.1.1 Stochastic Network with Arbitrary Edge Lengths

A stochastic network with arbitrary edge lengths consists of a directed graph $G = (V, E)$ where each edge $e \in E$ has a *length* randomly distributed within $[0, \infty]$ (∞ indicates that the edge is absent or fails). This is a generalization of the stochastic network introduced in Chapter 3 in which each edge is either present or absent. The survival probability equals to the probability that the edge length is less than ∞.

6.1.2 Management Actions

At each edge (u, v), an action a_{uv} can be taken to change an edge length distribution. Let π denote a policy, a set of actions being taken. Then, the length of edge e is represented by a random variable $L_e(\pi)$ whose distribution depends on π. We are given two cumulative distribution functions (CDF): $F_e^o : [0, \infty] \to [0, 1]$ (o means "original") and $F_e^m : [0, \infty] \to [0, 1]$ (m means "modifed") so that

$$\text{CDF of } L_e(\pi) = \begin{cases} F_e^o & \text{if } a_e \notin \pi \\ F_e^m & \text{if } a_e \in \pi \end{cases} \tag{6.1}$$

The action is taken to make the length stochastic shorter, that is, two CDFs satisfy the constraint $F_e^o(l) \geq F_e^m(l)$ for any $l \in [0, \infty]$. Let $L(\pi)$ denote the random vector consisting of $L_e(\pi)$ for all $e \in E$.

6.1.3 Decision Making Problem

For a given policy π, G along with $L_e(\pi)$ defines a probability distribution over a set of networks with edge lengths. Each network in the set has node set V, edge set E and edge lengths drawn from range $[0, \infty]$. Let $SPL(u, v, L(\pi); G)$ represent the shortest path

length or distance from a node u to a node v. If the distance is ∞, $SPL(u, v, L(\pi); G)$ is set to be M_{od} which is greater than any finite path length M_{od} represents the penalty charged for not connecting u and v. Then, the decision making problem can be written as

$$\min_{\pi \subseteq \mathcal{A}} \sum_{u,v \in V} \mathbb{E}_{L(\pi)} \left[r_{uv} \cdot SPL(u, v, L(\pi); G) \right] \qquad s.t. \quad \sum_{a_e \in \pi} c_e \leq \mathcal{B} \tag{6.2}$$

where $\mathcal{A}$ contains all available actions and r_{uv} represents the importance of the pair (u, v). We don't call r_{uv} the reward of (u, v) because the problem is a minimization problem. r_{uv} is better to be called as the *importance* of (u, v). Note that if a pair (u, v) has larger importance, more efforts need to be invested to minimize the distance from u to v.

6.1.4 Extensions

In the above definition, we assume that the length of each edge is independently distributed, but our solution method is applicable to more complex cases where the lengths of multiple edges are correlated or one investment can affect multiple edges simultaneously.

6.1.5 Comparison with Stochastic Network Design

A stochastic network design problem with arbitrary edge lengths and a standard stochastic network design problem share lots of similarities and have some differences.

- A stochastic network design problem focuses on the connectivity of the network while an SND-AEL problem focuses on both connectivity and distance.

- Two frameworks use the concept of the stochastic network to model the uncertainty of underlying phenomenon in slightly different way. A stochastic network design problem defines each edge to be either present or absent with certain probability. An SND-AEL problem defines a probability distribution for the length of each edge. The latter definition is more general than the former one because the length of an edge being ∞ is equivalent to the edge being absent and the length of the edge less than ∞ is equivalent to the edge being present.

- Both frameworks define management actions to modify the outcomes of a stochastic network. In a stochastic network design problem, an action can be taken to raise the survival probability and therefore improve the connectivity of the network. In an SND-AEL problem, an action can be taken to shorten the length of an edge and improve both the connectivity (e.g., shortening the length from ∞ to some value $< \infty$) and the distance.

- Both frameworks use r_{uv} to represent the importance of connecting a pair of nodes. In an SND-AEL problem, with a larger the value of r_{uv}, we want the probability that u is connected to v to be higher and the distance from u to v to be shorter.

- In a stochastic network design problem, the goal is to maximize the expected total rewards. A reward is collected for each pair of connected nodes. In an SND-AEL problem, the goal is to minimize the expected weighted distance between a set of node pairs in which a penalty is charged for each disconnected node pair. If the penalty is very large (e.g., infinity), the SND-AEL becomes equivalent to a stochastic network design problem.

6.1.6 Connection to the Continuous-Time Influence Maximization:

We note that the SND-AEL framework is also related to the continuous-time influence maximization problems [77, 19]. The time of a node being infected can be considered as the shortest path length from a source to this node 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 [44].

6.1.7 Complexity

Theorem 6.1.1. *An SND-AEL problem is APX-hard and is neither submodular nor supermodular.*

Proof. A problem called *predisaster transportation network preparation* introduced by the paper [83] is a special case of the SND-AEL framework. It is proved that a predisaster transportation network preparation problem, in general, is APX-hard and is neither sub-modular nor super-modular [83]. Thus, an SND-AEL problem, in general, has the same hardness results. □

6.2 Algorithm to Solve SND-AEL Problems

The basic idea to solve an SND-AEL Problem is the same as the idea to solve a stochastic network design problem for general directed graphs introduced in Chapter 5. The two components–a sampling procedure and a fast primal-dual algorithm–are discussed as follows.

6.3 Sampling Procedure

We use the SAA method to recast the stochastic optimization problem (6.2) as a deterministic analogue using N sampled scenarios that are generated by sampling from the underlying distribution. The basics of the SAA method is introduced in Section 5.1. However, we cannot 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 $\boldsymbol{\xi} = \{\boldsymbol{\xi}_e | \forall e \in E'\}$ and $\boldsymbol{\xi}_e$ is a random variable representing the length of edge $e \in E'$. The distribution of $\boldsymbol{\xi}_e$ 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^0}) and will include the invested edge (length ξ_{e^i}) if and only if $e \in \pi$. If $e \notin \pi$, the distance from u to v is ξ_{e^0}; we want this to have CDF H_e.

If $e \in \pi$, the distance from u to v is $\min\{\xi_{e^0}, \xi_{e^1}\}$ because both edges are in the graph; 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:

$$\boldsymbol{\xi}_{e^o} = \min_{l:F_e^o(l)=U_e} l, \qquad \boldsymbol{\xi}_{e^i} = \min_{l:F_e^m(l)=U_e} l \tag{6.3}$$

With this definition, we can sample the values of $\boldsymbol{\xi}_{e^o}$ and $\boldsymbol{\xi}_{e^i}$ independently of any policy. 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 F_e^o and F_e^m respectively. If the function F_e^o or F_e^m is strictly increasing, we have $\boldsymbol{\xi}_{e^o} = (F^o)^{-1}(U_e)$ or $\boldsymbol{\xi}_{e^i} = (F^m)^{-1}(U_e)$. Also, if F_e^o or F_e^m is a discrete CDF, the probability of any sampled value is nonzero. Now, we claim the following result.

Theorem 6.3.1. *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 the original SND-AEL problem.*

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

$$Pr(L_e(\pi) \le l) = \begin{cases} Pr(\boldsymbol{\xi}_{e^o} \le l) & \text{if } e \notin \pi \\ Pr(\min\{\boldsymbol{\xi}_{e^o}, \boldsymbol{\xi}_{e^i}\} \le l) & \text{if } e \in \pi \end{cases} \tag{6.4}$$

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 $l'' = \min_{l':H_e(l')=U_e} l'$. If $l'' \leq l$, 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

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

  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

only consider edges $(v_i, v_{i+1})^o$ and $(v_i, v_{i+1})^i$ if $(v_i, v_{i+1}) \in \pi$ for all $i = 1 : n-1$. 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 graph gives the same probability for any shortest path length and any path. So, the claim in the theorem holds. □

To summarize the SAA procedure, first, we generate N samples of $\boldsymbol{G}'$, $\{G'_1, ..., G'_N\}$, by sampling the value of U_e's and applying (6.3). In G'_k, edges have fixed lengths. Then, we form the following deterministic optimization problem:

$$\min_{\pi \subseteq E} \frac{1}{N} \sum_{(o,d)\in\Theta} \sum_{k=1}^{N} SPL(o, d, G'_k(\pi)) \; s.t. \sum_{e\in\pi} c_e \leq \mathcal{B} \tag{6.5}$$

where $SPL(o, d, G'(\pi))$ is the shortest path length in $G'(\pi)$ from o to d. By theorem 6.3.1, for any policy, the objective of (6.5) converges to the objective of (6.2) as N goes to infinity. In the next two sections, we formally define the problem (6.5) and give a fast algorithm to solve it.

6.4 Budget Set Weighted Shortest Path Steiner Graph (BSW-SPSG) Problem

In this section, wedefine a new network design problem called budget set weighted shortest path Steiner graph (BSW-SPSG) problem, which is similar to the BSW-DSG problem introduced earlier in Section 5.4. Then, we show how the deterministic optimization problem (6.5) can be formulated as a BSW-SPSG problem.

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 $\Theta = \{(o_1, d_1), ..., (o_T, d_T)\}$ $(o_i, d_i \in V^s)$ where each pair is associated with a penalty M_{st}. We are also given a collection of edge

sets $\mathbb{E} = \{E_1, ..., E_S\}$ where $E_i \subseteq E^s$ and each E_i is associated with a cost c_i. A subset $\mathcal{A} \subseteq \mathbb{E}$ 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 $(o, d) \in \Theta$ is minimized. That is,

$$\min_{\mathcal{A} \subseteq \mathbb{E}} \sum_{(o,d) \in \Theta} SPL(o, d, G^s(\mathcal{A})) \quad s.t. \quad \sum_{E_i \in \mathbb{E}} c_i \leq \mathcal{B} \tag{6.6}$$

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 the optimization problem (6.5) 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 $\mathbb{E}$. The vertex set of G^s is the union of those from the samples $\{G'_i\}$. The edges of G^s include original edges $\{e^o_k\}$ and invested edges $\{e^i_k\}$ 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 for *all* edges in G are grouped into an additional edge set E_0 of zero cost. Finally, let set Θ contains o-d pairs in all sampled graphs. An example is shown in Fig. 6.1.

The BSW-SPSG problem can be formulated as an MIP shown in Fig. 6.2. You may notice that the MIP formulation for the BSW-SPSG problem is the same as the MIP formulation for the BSW-DSG problem in Fig. 5.3 except that in the objective we have addition term for distance. To be self-contained, each component of this MIP is still explained.

A binary variable x^{od}_e 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 (=0). A binary decision variable y_i is defined for each edge set $E_i \in \mathbb{E}$ 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 $\mathcal{S}^{od}_o$ consist of all subsets of V that contain o but not d, $\mathcal{S}^{od}_o = \{S | S \subseteq V \wedge o \in S \wedge d \notin S\}$. Let set $\delta^+(S)$ 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

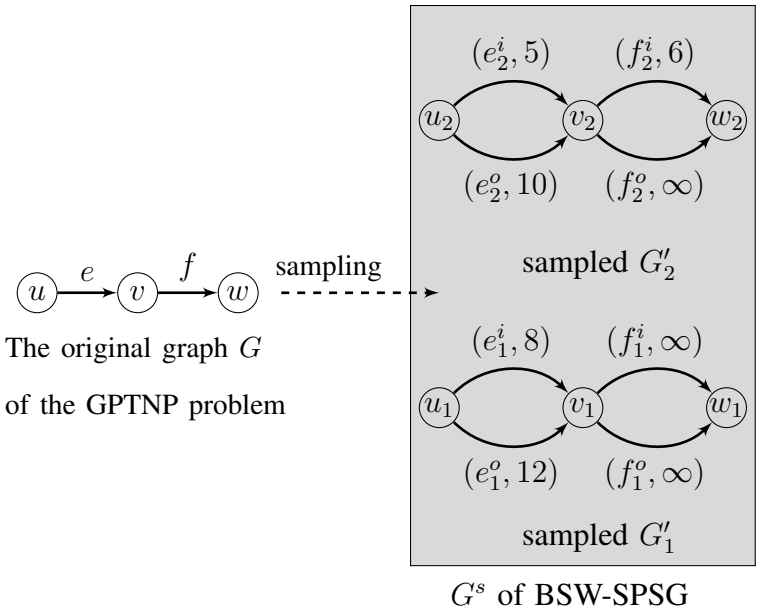


Figure 6.1: An example of creating the BSW-SPSG problem. The tuple $(e_2^i, 5)$ means the edge e_2^i has sampled length 5. The graph G^s contains all vertices and edges in gray. We have edge sets $E_e = \{e_1^i, e_2^i\}$, $E_f = \{f_2^i\}$ and $E_0 = \{e_1^o, e_2^o\}$ with $c_0 = 0$ in the BSW-SPSG problem. If the o-d pair in the GPTNP is $\{u, w\}$, in BSW-SPSG, $\Theta = \{(u_1, w_1), (u_2, w_2)\}$

constraint (6.2), 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 (6.3), an edge is on the shortest path from o to d only if it is purchased. Constraint (6.4) 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.

Theorem 6.4.1. *The BSW-SPSG problem is NP-hard and its objective function is neither sub- nor super-modular.*

Proof. The theorem can be proved using the same techniques being used in the proof of Theorem 5.4.1 and are omitted. □

6.5 Solving the BSW-SPSG Problem

To solve a 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 [53,

$$\min \sum_{(o,d)\in\Theta} \Big(\sum_{e\in E^s} d_e x_e^{od} \Big) + M_{o,d} z^{od} \tag{6.1}$$

$$s.t. \quad \sum_{e\in\delta^+(S)} x_e^{od} + z^{od} \geq 1 \quad \forall (o,d) \in \Theta, S \in \mathcal{S}_o^{od} \tag{6.2}$$

$$\sum_{i:e\in E_i} y_i \geq x_e^{od} \quad \forall (o,d) \in \Theta, \forall e \in E^s \tag{6.3}$$

$$\sum_{E_i\in\mathbb{E}} c_i y_i \leq \mathcal{B} \tag{6.4}$$

$$x_e^{od} \in [0,1] \quad \forall (o,d) \in \Theta \ \forall e \in E^s \tag{6.5}$$

$$y_i \in \{0,1\} \ \forall E_i \in \mathbb{E}, \qquad z^{od} \in \{0,1\} \ \forall (o,d) \in \Theta \tag{6.6}$$

Figure 6.2: MIP formulation of the budget BSW-SPSG problem

39]. The MIP of PCSW-SPSG is shown in Fig. 6.3. Then, the **bisection procedure** is used to find a β such that by solving the prize-collecting problem with β, a high quality solution of the budget problem can be obtained. The bisection procedure is basically the same as the one introduced in Section 5.6.1. Briefly, the bisection procedure starts with a large interval of β and halves 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.

6.5.0.1 Solving the Prize-Collecting Problem

To scale up to large networks, a fast primal-dual algorithm is developed, shown in Algorithm 2, to solve the PCSW-SPSG problem approximately. Fig. 6.3 gives the dual formulation of the LP relaxation of the prize-collecting problem. To compare the dual problem of PCSW-SPSG with the dual problem of BSW-DSG in Fig. 5.4, the only difference is the constraint (6.4). In constraint (6.4), the difference of two variables is less than d_e while in constraint (5.7), the difference of two variables is less than 0.

The algorithm borrows ideas from the primal-dual algorithms of the single pair shortest path problem and the generalized Steiner tree problem [96]. The basic idea is to repeatedly

Prize-Collecting Problem / Primal Problem:

$$L(\beta) = \min\left(\sum_{(o,d)\in\Theta}\left(\sum_{e\in E} d_e x_e^{od}\right) + M_{o,d} z^{od}\right) + \beta\left(\sum_{E_i\in\mathbb{E}} c_i y_i - \mathcal{B}\right) \tag{6.1}$$

$$s.t. \quad x, y \text{ satisfy constraints (6.2), (6.3), (6.5), (6.6) in Fig. 6.2} \tag{6.2}$$

Dual Problem:

$$\max \sum_{(o,d)\in\Theta, S\in\mathcal{S}_o^{od}} \mu_S^{od} \tag{6.3}$$

$$s.t. \sum_{S: e\in\delta^+(S), S\in\mathcal{S}_s^{od}} \mu_S^{od} - \lambda_e^{od} \le d_e \quad \forall (o,d)\in\Theta, \forall e\in E \tag{6.4}$$

$$\sum_{S: S\in\mathcal{S}_s^{od}} \mu_S^{od} \le M_{o,d} \quad \forall (o,d)\in\Theta \tag{6.5}$$

$$\sum_{(o,d)\in\Theta}\sum_{e\in E_i} \lambda_e^{od} \le \beta c_i \quad \forall E_i \in \mathbb{E} \tag{6.6}$$

Figure 6.3: Primal and dual of the prize-collecting problem

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 [94].

Definition 6.5.1. ***Primal Complementary Slackness Condition*** $x_e^{od} = 1$, $z^{od} = 1$ *and* $y_i = 1$ *imply equality in constaints* (6.4), (6.5) *and* (6.6) *in Fig. 6.3 respectively.*

Lines 2-20 of Algorithm 2 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 (6.2) in Fig. 6.2. Note that (6.3) in Fig. 6.2 is always satisfied as we proceed. A pair is *active* meaning that constraint (6.2) of this pair is violated for some S. To satisfy (6.2) 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 (6.2) is violated.

Algorithm 2 Primal-Dual Algorithm

1: **function** PRIMALDUAL($G^s, \mathcal{E}, \beta$)
2: Set $y_i \leftarrow 0 \ \forall E_i \in \mathbb{E}$ and $z^{od} \leftarrow 0 \ \forall (o,d) \in \Theta$
3: $F \leftarrow \phi$
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 $\mu^{od}_{C^{od}}$ of all *active* pairs simultaneously. If (6.4) in Fig. 6.3 becomes tight for edge e, increase λ^{od}_e to maintain feasibility **until**
9: **if** (6.5) in Fig. 6.3 becomes tight for (o,d) **then**,
10: Mark (o,d) *abandoned* and set $z^{od} = 1$
11: **else** ▷ (6.6) in Fig. 6.3 becomes tight for E_i
12: set $y_i = 1$ and $F \leftarrow F \cup e$
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** exists a shortest path using an edge in E_i **then**
24: $y_i \leftarrow 1$
25: **else**
26: $y_i \leftarrow 0$
27: **end if**
28: **end for**
29: **return** y
30: **end function**

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 (6.4) in Fig. 6.3 becomes tight for some e, by the slackness condition, we want to add e in to F (set $x^{od}_e = 1$), but e may not be purchased yet. So, we continue increasing both $\mu^{od}_{C^{od}}$ and λ^{st}_e with the same speed until following cases happen. If (6.5) in Fig. 6.3 becomes tight for some (o,d), by the slackness condition, we set $z^{od} = 1$ and never consider this pair again because all constraint (6.2)s in Fig. 6.2 that involve this pair are satisfied (line 10). If constraint (6.6) in Fig. 6.3 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 6.5.1. *The runtime of the algorithm is bounded by* $O((|E^s|+|\mathbb{E}|+|\Theta|)^2)$.

Proof. The algorithm takes at most $(|E|+|\Theta|)$ iterations, because at each iteration, either an edge is added into F or a pair becomes inactive (abandoned or connected). 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 (6.4) in Fig. 6.2 for all edges, constraint (6.5) in Fig. 6.2 for all pairs and constraint (6.6) in Fig. 6.2 for all edge sets. It takes time $O(|E|+|\Theta|+|\mathbb{E}|)$. In total, the line 8 takes $O((|E|+|\mathbb{E}|+|\Theta|)^2)$. Line 14 will only take $O(|E||\Theta|)$ in total because each edge will be expanded at most once for each pair. The shortest path calculation in line 21 takes time $O(|\Theta|\cdot(|E|+|V|\log|V|)$ in total for all pairs if we use Dijkstra's algorithm for each pair. Except these lines, other lines take insignificant runtime. Thus, for a connected network where $|V|\leq |E|$, the total runtime will be $O((|E|+|\mathbb{E}|+|\Theta|)^2)$.

□

6.6 Case Study: Predisaster Preparation Problem

The Predisaster preparation problem is introduced by the paper [72]. They consider a road network where edges represent road segments and nodes represent the intersection of roads. During natural disasters such as earthquakes or flood, each road segment has a probability, called the *failure rate*, to be damaged. If that happens, the road either becomes impassable, or the damage incurs a big delay to traffic. To reduce the travel time between a set of origin and destination locations, such as escaping from the origin locations to the destination locations during natural disasters, it is much cheaper and more realistic to reinforce road segments before the disasters. Once we have a model to predict the failure

rates of road segments during the disaster, the problem of deciding which roads to reinforce can be formulated as a stochastic optimization problem.

A predisaster preparation problem can be modeled as a stochastic network design problem with arbitrary edge length. The road network is modeled as a directed graph G=(V, E), where the length of an edge represents the *travel time* on the road when disasters happen and the length being ∞ means that the road becomes impassable. An action a_e with cost c_e can repair edge e before disasters to reduce its travel time stochastically. We are also given a set of o-d pairs, which represent important origin-destination locations in a specific application. Given a road network and travel times specified for edges, the time to get from a location A to a location B is assumed to equal to the shortest path length from A to B. The goal of the problem is to select a subset of actions to execute before disasters, subject to the budget limit $\mathcal{B}$, to minimize expected total travel time between a set of o-d pairs during disasters.

In the next sections, two predisaster preparation problems–Istanbul earthquake preparation problem and emergency medical service (EMS) response time minimization problem–are used as test cases to evaluate our algorithm.

6.6.1 Experimental Settings

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 the experiments, we treat the 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 test-

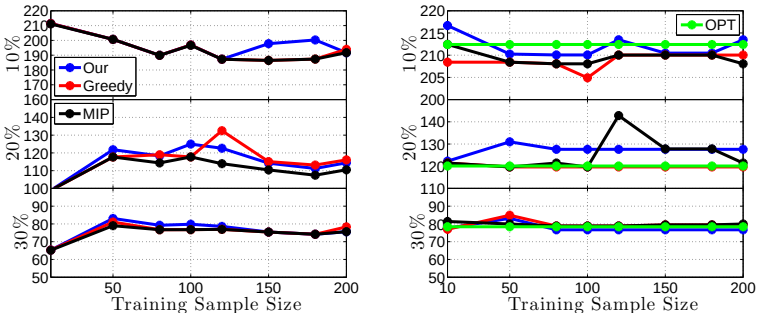
ing step. In testing, we draw another group of samples as testing samples and calculate, for a policy, the average travel time as a testing value. The testing value of a policy is a fair estimation of its expected travel time. As N increases, the convergence of the testing value is a good indicator that SAA converges.

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

6.6.2 Istanbul Earthquake Preparation

The first predisaster preparation problem is called the Istanbul earthquake preparation problem [72]. In the problem, we are expected to move casualties after an earthquake from highly affected areas, as determined in the Japan International Cooperation Agency Report (2002), to areas with large medical support capacities. The task is to minimize the sum of the expected shortest-path distances for five 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 the investment. We used the basic settings described by [72], with $M_{st} = 120$. On this small problem, we only report solution quality. In Fig. 6.4, 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 [83].

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, which indicates a slight difference in problem setting relative to previous work. The training value of each algorithm is always smaller than the testing value due to overfitting. Also, varying the sample size from 10 to 200 barely impacts the testing value, which implies that 10 training samples are enough for convergence.



(a) Training (b) Testing

Figure 6.4: "Istanbul Prep" benchmark: Y-axis shows path lengths.

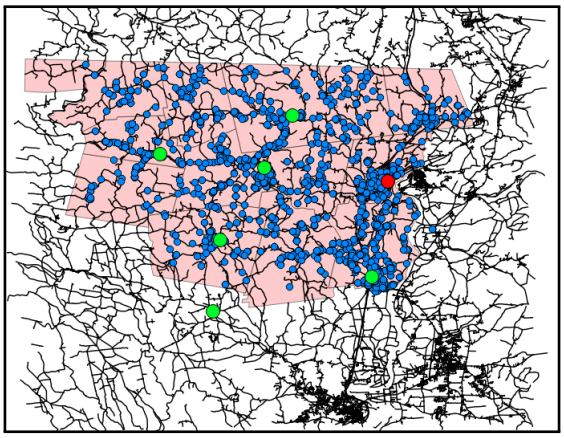
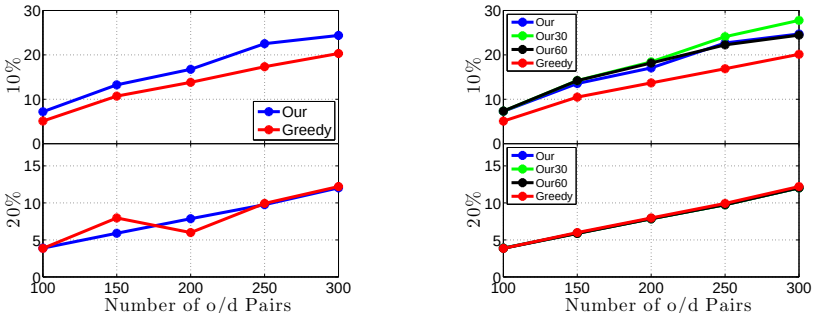


Figure 6.5: The road network showing patient locations (blue dots), hospital (red dot), and ambulance dispatch locations (green dots).

6.6.3 Emergency Medical Service Response Time Minimization Problem

The second real-world problem is the EMS response time minimization problem, which is the main focus of the experiments. Roadway stream crossing structures, such as culverts, become vulnerable to flood as climate changes. The failure of crossing structures causes road segments to be flooded, 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 only care about the 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



(a) Training (b) Testing

Figure 6.6: Performance on the sub-network of flood preparation problem. Y-axis represents the total travel time in seconds ($\times 10^4$).

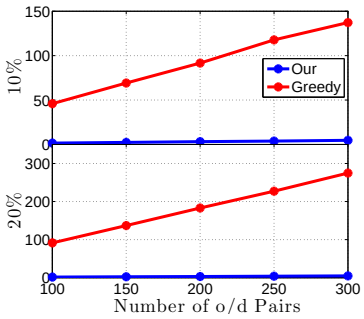


Figure 6.7: Training time (mins) on the sub-network.

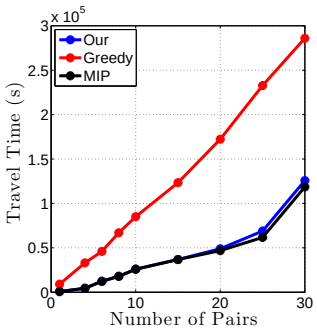


Figure 6.8: Complete network with survival probabilities 0.

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. 6.5, 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 correspondent 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.

6.6.3.1 Results on Sub-Networks

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. 6.6. 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"'s. For 20% budget, both algorithms perform the same in most cases. In terms of runtime, shown in Fig. 6.7, our algorithm is significantly faster. In testing, we use 100 testing samples and evaluate the policies produced by a 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.

6.6.3.2 Results on Complete Networks

We also tested on the complete road network using 10 training samples and 100 testing samples. In this case, the greedy algorithm is not scalable as one iteration takes more than 10 hours. Instead, we compared with two other methods. The results are shown in Table 6.1. 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,

Algorithm	Budget	Train ($\times 10^6$)	Time (h)	Test ($\times 10^6$)
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 6.1: On the complete network of Deerfield river watershed.

our algorithm is the only known method that can solve this problem with relatively good performance.

6.6.3.3 Experiments with More Challenging Settings

As shown above, the greedy algorithm performs quite well in training in terms of solution quality. One reason may be that in the road network, a large portion of o-d pairs are not far from each other. Additionally, the survival probabilities of crossings are relatively high such that, in the created BSW-SPSG problem, only one or two crossings fail on the path from o to d. In this case, fixing one crossing could reduce the total travel time immediately, which offers good guidance for the greedy algorithm. Intuitively, the greedy strategy will fail when a path is disconnected by multiple crossings simultaneously and no one-step reduction is available. We were curious about the performance of our algorithm is such more challenging settings. So we designed a BSW-SPSG problem using the complete network by making all crossings fail simultaneously. This problem setting is related to the robust optimization. Since a few pairs and a small budget are used, running the optimal MIP solver is feasible. The results in Fig. 6.8 show that the greedy algorithm performs poorly in this case, while our algorithm produces near optimal solutions.

6.7 Summary

In this chapter, we study a more general framework called stochastic network design with arbitrary edge length (SND-AEL) in which the goal is to both optimize the connectivity between a set of o-d pairs and also minimize the distance between them. The SND-AEL framework extends the stochastic network design framework. To develop an efficient al-

gorithm to solve an SND-AEL problem, the basic idea is to construct a deterministic optimization problem by sampling the lengths of edges, and use the solution to the deterministic optimization problem as the approximate solution. The deterministic optimization problem is formulated as a new network design problem called Budget Set Weighted Shortest Path Steiner Graph (BSW-SPSG), which is solved approximately by a bisection procedure and a primal-dual algorithm. At last, we test the algorithm on two real-world predisaster preparation problems. The experimental results show that my algorithm is much faster than existing algorithms and is able to produce high-quality solutions.

CHAPTER 7

SUMMARY

The thesis studies how to select management actions to modify the structure of a network to steer phenomena that occur in the network toward certain desired objectives. Several real-world problems are which river barriers to remove to facilitate fish to access their historical habitats in a river network, where to set conservation areas to help the dispersal of red-cockaded woodpeckers (RCW), and which roads to reinforce to minimize the disruption of emergency medical services by natural disasters. A computational tool is needed to help decision makers select a cost-efficient strategy among a large number of candidate modification strategies by taking into account a budget constraint. These problems are treated as stochastic optimization problems. Existing algorithms either perform poorly or fail to scale to problems of large sizes. The goal of the thesis is to develop scalable algorithms that can solve a broad range of such network optimization problems. To achieve this goal, a general problem framework called a stochastic network design framework is defined to model a broad range of problems including the examples mentioned earlier. Several scalable algorithms and optimization techniques are developed to solve problems within this framework.

7.1 Stochastic Network Design Framework

The major component of the framework is a stochastic network. A stochastic network is defined using a directed graph, in which each edge is either present or absent with a survival probability. Actions with certain costs can be taken to raise its survival probability and therefore improve the connectivity of the network. In the barrier removal problem,

the survival probability of an edge models the probability that fish can pass a barrier represented by the edge, and an action is taken to improve the passage of the barrier. In the conservation planning problem, the survival probability models the probability that RCWs can successfully travel along an edge and survive in the habitat area represented by the ending node of the edge, and an action is taken to improve the living condition of the habitat area. The optimization problem is to determine which actions to take subject to a budget limit.

7.2 Fast Algorithms

Scalable algorithms are developed under three different settings that gradually become more general. The underlying network is a tree (Chapter 3 and 4), the underlying network is a general directed graph (Chapter 5), and edges in the network have arbitrary lengths that make the decision-making problem optimize both connectivity and distance of the network (Chapter 6). In Chapter 3, the underlying network is assumed to be a directed rooted tree, which models a diffusion process that starts from a single root node and gradually spreads into a tree-structured network. An example is the spreading process of fish in a river network. A fully polynomial-time approximation scheme (FPTAS) called rounded dynamic programming (RDP) algorithm is developed for this setting, of which the basic idea is to combine a dynamic programming algorithm and a rounding strategy. Applying to a real-world barrier removal problem, RDP is much faster than a mixed integer solver and can produce nearly optimal solutions. In Chapter 4, the underlying network is assumed to be a bidirected tree, in which a diffusion process starts from instead of a single root node but multiple nodes. This setting makes the optimization problem harde, for which a FPTAS called rounded dynamic programming for bidirected trees (RDPB) with worse runtime complexity is developed. Applying to a real-world bidirectional barrier removal problem, RDPB is faster than the greedy algorithm and can produces a solution with better qualities within a reasonable amount of time. In Chapter 5, techniques are developed for general

directed graphs that makes the optimization problem at least #P-hard. The basic idea is to construct a deterministic optimization problem using an SAA method, and then to develop a fast primal-dual algorithm to compute a nearly optimal solution for the deterministic optimization problem, which, as shown by experiments, is also a nearly optimal solution of the original optimization problem. Applying to the conservation planning for red-cockaded woodpeckers (RCW), the algorithm is around 120 times faster than a mixed integer solver with slightly worse solutions and around 30 times faster than a greedy algorithm with better solutions. In Chapter 6, a fast algorithm is developed for networks with arbitrary edge lengths. The length of an edge is randomly distributed in a range $[0, \infty]$, which can be reduced by an action stochastically. The goal is to determine which actions to take to minimize the distance between a set specified of o-d pairs. The algorithm for this setting is based on the same structure as the algorithm in Chapter 5 but has a different sampling and a more complex primal-dual components. The algorithm is applied to a predisaster preparation problem for emergency medical services (EMS) in which the goal is to decide which roads to reinforce to reduce the response time of EMS during floods. The real-world road network has $50,000$ edges but an existing state-of-the-art method can only scale to 1000 edges. The algorithm can compute high-quality solutions within 6 hours on this large network and runs 400 times faster than a greedy algorithm.

7.3 Future Directions

The framework and algorithms are very general and can be applied to a broad range of network optimization problems. Three real-world problems are formulated by the framework, and high-quality are produced by the algorithms within a reasonable amount of time. However, there still exists aspects that the framework does not model and assumptions made by the framework that some real-world problems don't satisfy. This section discusses several properties that some real-world problems possess but are not modeled by the stochastic network design frameworks. It is an open question on how to enrich the frame-

work to include these properties and how to develop scalable algorithms to solve problems within the new framework.

7.3.1 Robust Optimization

In the stochastic network design framework, it is assumed that probability distributions have already been created by domain experts or by machine learning techniques to describe the uncertainty of the underlying phenomena. For example, algorithms can be applied only when all survival probabilities are known. This assumption is valid for many problems including the applications given in the thesis. For the barrier removal problem, the probability that fish can pass a barrier is estimated from data by ecologists. For the RCW problem, the probability that RCWs will successfully spread from a location A to a location B is estimated from real-world observations by machine learning techniques [97]. For the EMS response time minimization problem, the failure rates of culverts during floods are estimated by environmental scientists.

In many situations, the existing probability distribution may not be accurate enough to help find good decisions or plans. For example, not enough data are available to construct a sufficiently confident probability distribution, or available data involves lots of noise.

An alternative optimization technique to deal with these situations is called [4, 5]. In the robust optimization, since the uncertainty of a random variable cannot be accurately described, the worst value of that random variable is used when making decisions and planning. A recent work [52] provides an algorithm based on the idea of robust optimization to solve the RCW problem, for which only a confident range of the probability of an edge–instead of a point estimate of the probability–is available. For example, we only know the probability that the RCWs will spread from a location A to a location B is in a range such as $[0.2, 0.6]$, which could be a 90% confidence interval of the spreading probability. Instead of maximizing the expected total rewards as done by the stochastic network design framework, the authors of the paper choose a policy to minimize the max-regret, where

max-regret is the largest quantity by which the decision maker could regret taking the policy while allowing probabilities to vary within the ranges [6]. It is an interesting future research direction on how to modify my stochastic network design framework to deal with the model uncertainty and how to develop robust optimization techniques to solve these problems.

7.3.2 Multiple Processes

In the stochastic network design framework, only a single phenomenon or process that occurs in a network is considered and some other external effects are ignored. For the barrier removal problem, it is assumed that there only exists a single diffusion process of fish (e.g., salmon) in a river network while other types of fish, each of which forms a diffusion process, that may eat salmon or compete for habitats against salmon are ignored. If these other types of fish do exist, the decisions made by ignoring them are far from optimal. For the EMS response time minimization problem, it is assumed that the travel time for an ambulance to get from a location A to a different location B is the shortest cumulate travel time while the travel time of a road is calculated by the length of the road divided by its speed limit. This assumption is confidently realistic while the traffic is not busy or in rural area–the latter holds for the network we use. It becomes an unrealistic assumption during rush hours or in big cities, such as New York city, where the traffic is always very busy. In these situations, a better way is to describe the traffic and the competition between different vehicles using the model of congestion game [64] and calculate travel time at a Nash equilibrium point. It is an interesting future research direction on how to change the stochastic network design framework to model multiple processes that occur in the network and the relationships between these processes.

7.3.3 Adaptive Decision Making

A stochastic network design problem is basically a one-step decision-making problem, that is, the decision is only made once, and all management actions that will be executed are

determined at that point. As shown earlier, the barrier removal problem is formulated as a one-step decision-making problem in the sense that a decision maker decides which barriers to remove in a river network and removes those barriers before fish enter the river network. No further decisions are made afterwards. For this problem, actions are executed before the fish start to spread. A different way is to allow actions being taken at multiple stages. At each stage, we collect more information about the underlying phenomenon that helps to reduce uncertainty, and we decide which actions to take based on this information. This type of decision-making framework is called adaptive decision-making. In comparison, for a one-step decision-making problem, all available funds are spent at once while for an adaptive decision-making problem funds are spent gradually over multiple steps. One-step decision-making simplifies the decision-making process and makes the resulted optimization problem tractable. Also, taking actions early reduces the cost. For example, repairing a malfunctioning culvert is more expensive than reinforcing it before it fails. The advantage of adaptive decision-making is that it allows people to make better decisions as more and more information is collected to reduce the uncertainty–although the execution of these decisions may be very expensive at that point. But, an adaptive decision-making problem is usually very hard to solve, and some assumptions, such as adaptive submodular introduced by paper [27], are needed to make the problem tractable. Two well-studied adaptive decision-making framework are Markov decision process (MDP) [76] and partially observable Markov decision process (POMDP) [88], which are used in slightly different contexts such as autonomous driving [73]. It is an interesting future research direction on how to define an adaptive version of the stochastic network design framework and how to develop fast algorithms to solve them.

BIBLIOGRAPHY

[1] Arborescence. https://en.wikipedia.org/wiki/Arborescence_(graph_theory).

[2] Ahuja, Ravindra K., Magnanti, Thomas L., and Orlin, James B. *Network flows*. Alfred P. Sloan School of Management, MIT, 1988.

[3] Ball, Michael O. Computational complexity of network reliability analysis: an overview. *IEEE Transactions on Reliability 35*, 3 (1986), 230–239.

[4] Ben-Tal, Aharon, El Ghaoui, Laurent, and Nemirovski, Arkadi. *Robust optimization*. Princeton University Press, 2009.

[5] Bertsimas, Dimitris, Brown, David B, and Caramanis, Constantine. Theory and applications of robust optimization. *SIAM review 53*, 3 (2011), 464–501.

[6] Boutilier, Craig, Patrascu, Relu, Poupart, Pascal, and Schuurmans, Dale. Constraint-based optimization with the minimax decision criterion. In *Principles and Practice of Constraint Programming* (2003), Springer, pp. 168–182.

[7] Cerdeira, J Orestes, Gaston, Kevin J, and Pinto, Leonor S. Connectivity in priority area selection for conservation. *Environmental Modeling & Assessment 10*, 3 (2005), 183–192.

[8] Chen, Wei, Wang, Chi, and Wang, Yajun. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2010), ACM, pp. 1029–1038.

[9] Chen, Wei, Wang, Yajun, and Yang, Siyu. Efficient influence maximization in social networks. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2009), ACM, pp. 199–208.

[10] Chen, Wei, Yuan, Yifei, and Zhang, Li. Scalable influence maximization in social networks under the linear threshold model. In *Proceedings of 10th International Conference on Data Mining* (2010), IEEE, pp. 88–97.

[11] Colbourn, Charles J. Network resilience. *SIAM Journal on Algebraic Discrete Methods 8*, 3 (1987), 404–409.

[12] Colbourn, Charles J. Analysis and synthesis problems for network resilience. *Mathematical and Computer Modelling 17*, 11 (1993), 43–48.

[13] Conrad, Jon, Gomes, Carla P, van Hoeve, Willem-Jan, Sabharwal, Ashish, and Suter, Jordan. Connections in networks: Hardness of feasibility versus optimality. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. Springer, 2007, pp. 16–28.

[14] Conrad, Jon M, Gomes, Carla P, van Hoeve, Willem-Jan, Sabharwal, Ashish, and Suter, Jordan F. Wildlife corridors as a connected subgraph problem. *Journal of Environmental Economics and Management 63*, 1 (2012), 1–18.

[15] Dengiz, Berna, Altiparmak, Fulya, and Smith, Alice E. Efficient optimization of all-terminal reliable networks, using an evolutionary approach. *IEEE Transactions on Reliability 46*, 1 (1997), 18–26.

[16] Dilkina, Bistra, and Gomes, Carla P. Solving connected subgraph problems in wildlife conservation. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. Springer, 2010, pp. 102–116.

[17] Dilkina, Bistra, Lai, Katherine J, and Gomes, Carla P. Upgrading shortest paths in networks. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. Springer, 2011, pp. 76–91.

[18] Domingos, Pedro, and Richardson, Matt. Mining the network value of customers. In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data mining* (2001), ACM, pp. 57–66.

[19] Du, Nan, Song, Le, Gomez-Rodriguez, Manuel, and Zha, Hongyuan. Scalable influence estimation in continuous-time diffusion networks. In *Proceedings of the Neural Information Processing Systems* (2013), pp. 3147–3155.

[20] Faughnan, Kim. *UI credit: , Mount Holyoke College 15*.

[21] Feige, Uriel, Mirrokni, Vahab S, and Vondrak, Jan. Maximizing non-monotone submodular functions. *SIAM Journal on Computing 40*, 4 (2011), 1133–1153.

[22] Fouskakis, Dimitris, and Draper, David. Stochastic optimization: a review. *International Statistical Review 70*, 3 (2002), 315–349.

[23] Fu, Michael C. Optimization for simulation: Theory vs. practice. *INFORMS Journal on Computing 14*, 3 (2002), 192–215.

[24] Fujishige, Satoru. *Submodular functions and optimization*, vol. 58. Elsevier, 2005.

[25] Gao, Chao, Liu, Jiming, and Zhong, Ning. Network immunization and virus propagation in email networks: experimental evaluation and analysis. *Knowledge and information systems 27*, 2 (2011), 253–279.

[26] Glover, Fred, and Laguna, Manuel. *Tabu Search*. Springer, 2013.

[27] Golovin, Daniel, and Krause, Andreas. Adaptive submodularity: A new approach to active learning and stochastic optimization. In *Proceedings of the Conference on Learning Theory* (2010), pp. 333–345.

[28] Golovin, Daniel, and Krause, Andreas. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *Journal of Artificial Intelligence Research* (2011), 427–486.

[29] Gomes, Carla P. Computational sustainability: Computational methods for a sustainable environment, economy, and society. *The Bridge 39*, 4 (2009), 5–13.

[30] Gupta, Anupam, and Könemann, Jochen. Approximation algorithms for network design: A survey. *Surveys in Operations Research and Management Science 16*, 1 (2011), 3–20.

[31] Gurobi. Optimization software. `http://www.gurobi.com/`, 2015.

[32] Hansen, Eric A, and Zhou, Rong. Anytime heuristic search. *Journal of Artificial Intelligence Research 28* (2007), 267–297.

[33] Hanski, I., Ed. *Metapopulation Ecology*. Oxford University Press, 1999.

[34] Hanski, Ilkka, and Ovaskainen, Otso. The metapopulation capacity of a fragmented landscape. *Nature 404*, 6779 (2000), 755–758.

[35] Homem-de Mello, Tito, and Bayraksan, Güzin. Monte carlo sampling-based methods for stochastic optimization. *Manuscript, under review for Surveys in Operations Research and Management Science. Preprint available at Optimization Online (http://www. optimization-online. org)* (2013).

[36] Hwang, Chii-Ruey. Simulated annealing: theory and applications. *Acta Applicandae Mathematicae 12*, 1 (1988), 108–111.

[37] Hwang, Frank K, Richards, Dana S, and Winter, Pawel. *The Steiner tree problem*. Elsevier, 1992.

[38] ILOG, IBM. Cplex optimizer.

[39] Jain, Kamal, and Vazirani, Vijay V. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *Journal of the ACM 48*, 2 (2001), 274–296.

[40] Johnson, David S, Lenstra, Jan Karel, and Kan, AHG. The complexity of the network design problem. *Networks 8*, 4 (1978), 279–285.

[41] Johnson, David S, Minkoff, Maria, and Phillips, Steven. The prize collecting steiner tree problem: theory and practice. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA 17)* (2000), vol. 1, Citeseer, p. 4.

[42] Kempe, David, Kleinberg, Jon, and Tardos, Éva. Maximizing the spread of influence through a social network. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2003), pp. 137–146.

[43] Kempe, David, Kleinberg, Jon, and Tardos, Éva. Influential nodes in a diffusion model for social networks. In *Proceedings of the International Colloquium on Automata, Languages, and Programming* (2005), Springer, pp. 1127–1138.

[44] Khalil, Elias Boutros, Dilkina, Bistra, and Song, Le. Scalable diffusion-aware optimization of network topology. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2014), ACM, pp. 1226–1235.

[45] Kleywegt, Anton J., Shapiro, Alexander, and Homem-de Mello, Tito. The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization 12* (February 2002), 479–502.

[46] Kleywegt, Anton J, Shapiro, Alexander, and Homem-de Mello, Tito. The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization 12*, 2 (2002), 479–502.

[47] Konak, Abdullah, and Smith, Alice E. Network reliability optimization. In *Handbook of Optimization in Telecommunications*. Springer, 2006, pp. 735–760.

[48] Korf, Richard E. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial intelligence 27*, 1 (1985), 97–109.

[49] Korf, Richard E. Real-time heuristic search. *Artificial Intelligence 42*, 2-3 (1990), 189–211.

[50] Krause, Andreas, and Golovin, Daniel. Submodular function maximization. *Tractability: Practical Approaches to Hard Problems 3* (2012), 19.

[51] Kruskal, Joseph B. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society 7*, 1 (1956), 48–50.

[52] Kumar, Akshat, Singh, Arambam James, Varakantham, Pradeep, and Sheldon, Daniel. Robust decision making for stochastic network design. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI)* (2016), pp. 3857–3863.

[53] Kumar, Akshat, Wu, Xiaojian, and Zilberstein, Shlomo. Lagrangian relaxation techniques for scalable spatial conservation planning. In *Proceedings of the 26th Conference on Artificial Intelligence* (2012), pp. 309–315.

[54] Lai, Katherine J, Gomes, Carla P, Schwartz, Michael K, McKelvey, Kevin S, Calkin, David E, and Montgomery, Claire A. The steiner multigraph problem: wildlife corridor design for multiple species.

[55] LeBras, Ronan, Dilkina, Bistra N, Xue, Yexiang, Gomes, Carla P, McKelvey, Kevin S, Schwartz, Michael K, Montgomery, Claire A, et al. Robust network design for multi-species conservation. In *Proceedings of the 27th Conference on Artificial Intelligence* (2013), pp. 1305–1312.

[56] Lenstra, Jan K. *Local search in combinatorial optimization*. Princeton University Press, 1997.

[57] Leskovec, Jure, Adamic, Lada A, and Huberman, Bernardo A. The dynamics of viral marketing. *ACM Transactions on the Web (TWEB) 1*, 1 (2007), 5.

[58] McGarigal, Kevin, Compton, Bradley W., Jackson, Scott D., Plunkett, Ethan, Rolih, Kasey, Portante, Theresa, and Ene, Eduard. Conservation assessment and prioritization system (CAPS). Tech. Rep. November, Department of Environmental Conservation, Univ. of Massachusetts Amherst, 2011.

[59] Merriam, GRAY. Connectivity: a fundamental ecological characteristic of landscape pattern. In *Proceedings of the 1st seminar of the International Association of Landscape Ecology on Methodology in Landscape Ecological Research and Planning* (1984).

[60] Miller-Hooks, Elise, Zhang, Xiaodong, and Faturechi, Reza. Measuring and maximizing resilience of freight transportation networks. *Computers & Operations Research 39*, 7 (2012), 1633–1643.

[61] Mladenović, Nenad, and Hansen, Pierre. Variable neighborhood search. *Computers & Operations Research 24*, 11 (1997), 1097–1100.

[62] Moilanen, Atte, Wilson, Kerrie A, and Possingham, Hugh Philip. *Spatial conservation prioritization: quantitative methods and computational tools*. Oxford University Press Oxford, UK, 2009.

[63] Neeson, Thomas M, Ferris, Michael C, Diebel, Matthew W, Doran, Patrick J, OHanley, Jesse R, and McIntyre, Peter B. Enhancing ecosystem restoration efficiency through spatial and temporal coordination. *Proceedings of the National Academy of Sciences 112*, 19 (2015), 6236–6241.

[64] Nisan, Noam, Roughgarden, Tim, Tardos, Eva, and Vazirani, Vijay V. *Algorithmic game theory*, vol. 1. Cambridge University Press Cambridge, 2007.

[65] O'Hanley, Jesse Rush, and Tomberlin, David. Optimizing the removal of small fish passage barriers. *Environmental Modeling and Assessment 10*, 2 (2005), 85–98.

[66] Önal, Hayri, and Briers, Robert A. Designing a conservation reserve network with minimal fragmentation: a linear integer programming approach. *Environmental Modeling & Assessment 10*, 3 (2005), 193–202.

[67] Önal, Hayri, and Wang, Yicheng. A graph theory approach for designing conservation reserve networks with minimal fragmentation. *Networks 51*, 2 (2008), 142–152.

[68] Optimization, Gurobi, et al. Gurobi optimizer reference manual. *URL: http://www. gurobi. com 2* (2012), 1–3.

[69] Pascual-Hortal, Lucía, and Saura, Santiago. Comparison and development of new graph-based landscape connectivity indices: towards the priorization of habitat patches and corridors for conservation. *Landscape Ecology 21*, 7 (2006), 959–967.

[70] Pastor-Satorras, Romualdo, Castellano, Claudio, Van Mieghem, Piet, and Vespignani, Alessandro. Epidemic processes in complex networks. *arXiv preprint arXiv:1408.2701* (2014).

[71] Pattnaik, SB, Mohan, S, and Tom, VM. Urban bus transit route network design using genetic algorithm. *Journal of Transportation Engineering 124*, 4 (1998), 368–375.

[72] Peeta, Srinivas, Salman, F Sibel, Gunnec, Dilek, and Viswanath, Kannan. Pre-disaster investment decisions for strengthening a highway network. *Computers & Operations Research 37*, 10 (2010), 1708–1719.

[73] Pineda, Luis Enrique, Lu, Yi, Zilberstein, Shlomo, and Goldman, Claudia V. Fault-tolerant planning under uncertainty. In *Proceedings of the 23th International Joint Conference on Artificial Intelligence* (2013), pp. 2350–2356.

[74] Prim, Robert Clay. Shortest connection networks and some generalizations. *Bell System Technical Journal 36*, 6 (1957), 1389–1401.

[75] Prömel, Hans Jürgen, and Steger, Angelika. *The Steiner tree problem: a tour through graphs, algorithms, and complexity*. Springer Science & Business Media, 2012.

[76] Puterman, Martin L. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[77] Rodriguez, Manuel Gomez, and Schölkopf, Bernhard. Influence maximization in continuous time diffusion networks. *arXiv preprint arXiv:1205.1682* (2012).

[78] Rong-Hong, Jan. Design of reliable networks. *Computers & Operations Research 20*, 1 (1993), 25–34.

[79] Russell, Stuart, Norvig, Peter, and Intelligence, Artificial. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs 25* (1995), 27.

[80] Satyanarayana, Appajosyula, and Wood, R Kevin. A linear-time algorithm for computing k-terminal reliability in series-parallel networks. *SIAM Journal on Computing 14*, 4 (1985), 818–832.

[81] Saura, Santiago, and Pascual-Hortal, Lucía. A new habitat availability index to integrate connectivity in landscape conservation planning: comparison with existing indices and application to a case study. *Landscape and Urban Planning 83*, 2 (2007), 91–103.

[82] Saura, Santiago, and Torne, Josep. Conefor sensinode 2.2: A software package for quantifying the importance of habitat patches for landscape connectivity. *Environmental Modelling & Software 24*, 1 (2009), 135–139.

[83] Schichl, Hermann, and Sellmann, Meinolf. Predisaster preparation of transportation networks. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence* (2015), pp. 709–715.

[84] Schneider, Christian M, Mihaljev, Tamara, Havlin, Shlomo, and Herrmann, Hans J. Suppressing epidemics with a limited amount of immunization units. *Physical Review E 84*, 6 (2011), 061911.

[85] Schumaker, Nathan H. Using landscape indices to predict habitat connectivity. *Ecology* (1996), 1210–1225.

[86] Shapiro, Alexander, Dentcheva, Darinka, et al. *Lectures on stochastic programming: modeling and theory*, vol. 9. SIAM, 2009.

[87] Sheldon, Daniel, Dilkina, Bistra, Elmachtoub, Adam, Finseth, Ryan, Sabharwal, Ashish, Conrad, Jon, Gomes, Carla, Shmoys, David, Allen, William, Amundsen, Ole, and Vaughan, William. Maximizing the spread of cascades using network design. In *Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence* (2010), pp. 517–526.

[88] Smallwood, Richard D, and Sondik, Edward J. The optimal control of partially observable markov processes over a finite horizon. *Operations Research 21*, 5 (1973), 1071–1088.

[89] Srivaree-ratana, Chat, Konak, Abdullah, and Smith, Alice E. Estimation of all-terminal network reliability using an artificial neural network. *Computers & Operations Research 29*, 7 (2002), 849–868.

[90] Sterbenz, James PG, Hutchison, David, Çetinkaya, Egemen K, Jabbar, Abdul, Rohrer, Justin P, Schöller, Marcus, and Smith, Paul. Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines. *Computer Networks 54*, 8 (2010), 1245–1265.

[91] Sviridenko, Maxim. A note on maximizing a submodular set function subject to a knapsack constraint. *Operations Research Letters 32*, 1 (2004), 41–43.

[92] Tambosi, Leandro R, Martensen, Alexandre C, Ribeiro, Milton C, and Metzger, Jean P. A framework to optimize biodiversity restoration efforts based on habitat amount and landscape connectivity. *Restoration Ecology 22*, 2 (2014), 169–177.

[93] Valiant, Leslie G. The complexity of computing the permanent. *Theoretical Computer Science 8*, 2 (1979), 189–201.

[94] Vazirani, V. *Approximation Algorithms*. Springer, 2003.

[95] Williams, Justin C, ReVelle, Charles S, and Levin, Simon A. Spatial attributes and reserve design models: a review. *Environmental Modeling & Assessment 10*, 3 (2005), 163–181.

[96] Williamson, David P., and Shmoys, David B. *The design of approximation algorithms*. Cambridge University Press, 2011.

[97] Wu, Xiaojian, Sheldon, Daniel, and Zilberstein, Shlomo. Parameter learning for latent network diffusion. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligene* (2013).

[98] Wu, Xiaojian, Sheldon, Daniel, and Zilberstein, Shlomo. Rounded dynamic programming for tree-structured stochastic network design. In *Proceedings of the 28th Conference on Artificial Intelligence* (2014).

[99] Wu, Xiaojian, Sheldon, Daniel, and Zilberstein, Shlomo. Stochastic network design in bidirected trees. In *Proceedings of the Neural Information Processing Systems* (2014).