Approximation Algorithms for Graph Protection Problems

by

Alexander Lange

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Combinatorics and Optimization

Waterloo, Ontario, Canada, 2015

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

We study a budgeted cut problem known as Graph Protection, where the goal is to remove edges of a given graph in order to protect valuable nodes from stochastic, infectious threats. This problem was recently proposed by Shmoys and Spencer to model challenges associated with wildfire prevention when resources are limited and only probabilistic data of ignition location is known. The input consists of a graph with node values and edge costs, a budget, and probabilistic data signifying the likelihood a set of nodes will ignite; the goal is to remove a set of edges with total cost at most the budget so as to maximize the expected protected value unreachable from these stochastic ignition scenarios.

Our focus is on the design of approximation algorithms for Graph Protection when an ignition scenario can contain an arbitrary number of nodes, a setting mostly left open in the literature. Our main positive result is the design of constant-factor approximation algorithms for Graph Protection when the input graph is a tree and each node has an independent chance of being an ignition point. We also show that in the general case when ignition probabilities are defined by a distribution over subsets of nodes, the problem becomes at least as hard as the Densest k-Subgraph problem and thus, such an approximation algorithm is unlikely to exist.

Acknowledgements

First and foremost, I thank Chaitanya Swamy for his generosity and support, which have been undoubtedly invaluable throughout my time at the University of Waterloo. This work, in particular, would not be possible without his advisement and mentorship, including the many meetings in which all of these results were obtained and his thorough and patient proofreading of this document. I will always cherish my time working on approximation algorithms, and this fondness is a direct result of his passion, understanding, and teaching.

I thank Laura and Costis for agreeing to review this thesis, and for being two significant sources of knowledge and expertise during my time here. I also recognize the invaluable guidance of the C&O administration, without whom, I certainly would not be where I am now; I give special thanks to Melissa for helping me through all of it.

Lastly, I give my love and thanks to my family and friends, especially my parents and my girlfriend Tori, who have shown me a constant stream of love, appreciation, support, and unwavering (and usually undeserved) patience.

Contents

Chapter 1

Introduction

1.1 Overview

The spread of infection through a network is a fundamental graph process spanning multiple areas of combinatorial optimization and computer science. In this thesis, we focus on a model that abstracts settings where the network represents a landscape, and the goal is to protect this landscape against the spread of wildfires. In particular, we are interested in preventative care, where actions based on probabilistic information are taken prior to an ignition in the hopes of mitigating a potentially catastrophic spread. These actions usually take the form of preventative fuel treatment, such as prescribing small, controlled fires or mechanically thinning or removing fuels of the landscape. Naturally, these treatments are constrained by limited resources due to, for example, location, time, or cost.

Recently, Shmoys and Spencer [40] proposed the *Graph Protection Problem* to model the spreading of wildfires and the problems related to its prevention. In Graph Protection, we are given a graph where the nodes and edges represent the spatial properties of the landscape. Each node serves as some area within the landscape, and is associated with a value as well as probabilistic data that signifies the likelihood the area will ignite. Once an area ignites, fire spreads through the edges of the graph. Its prevention is modeled as the removal of such edges, each of which has an associated cost of removal. The goal is to maximize the expected value protected by a set of removed edges when knowing the probabilistic data of source ignition, such that the total cost of removal does not exceed a given budget. A formal definition of the problem appears in Section 1.2.

Graph Protection falls under the umbrella of *stochastic optimization*, a class of mathematical optimization that attempts to model problems containing uncertainty in the input data. The problem is also computationally intractable, and thus it is unlikely that there exists efficient, exact algorithms for it. In such cases, an approach often taken is to devise *approximation algorithms* for the problem, where a feasible solution is found that has value at least a provable factor away from the optimal value. Shmoys and Spencer present

approximation algorithms for different problem variants, most of which involve the special cases where the input graph is a tree and all ignition scenarios consist of a single node.

This work is concerned with the more general Graph Protection problem where there can exist an arbitrary number of ignition scenarios, each (possibly) containing multiple ignition nodes. There are a variety of ways by which such ignition scenarios can take shape, all of which were left open by [40]; the only setting considered in [40] that allows for ignition scenarios to contain multiple nodes is the rather specialized case where there are a constant number of ignition scenarios each containing a constant number of nodes. Our main contribution (see Section 1.3) is the development of constant-factor approximation algorithms for this problem when the input graph is a tree and each node has an independent chance of ignition. (Note that this succinctly captures a setting where there are an *exponential* number of scenarios, each of which may contain an arbitrary number of ignition nodes.) We also show that in the general case when ignition probabilities are defined by a distribution over subsets of nodes, the problem becomes at least as hard as the Densest k-Subgraph problem and thus, such an approximation algorithm is unlikely to exist.

1.2 Problem Definition and Scope

In the Graph Protection (GP) problem, we are a given a landscape graph $G = (V, E)$, where V is a set of $|V| = n$ nodes representing areas of the landscape, and E is a set of edges (i.e. unordered pairs of nodes) that symbolize adjacency of these areas. Each node $u \in V$ has a non-negative value w_u, and each edge $e \in E$ has a non-negative cost c_e. We are also given a budget B, representing the maximum amount of resources available for prevention. As is common practice, we will generalize the weight function w and the cost function c to that of sets. That is, for $S \subseteq V$, let $w(S) = \sum_{u \in S} w_u$ and for $Y \subseteq E$, let $c(Y) = \sum_{e \in Y} c_e$.

In addition to the above information, GP consists of a set of ignition scenarios $\mathcal{I}$, each of which is a subset of nodes $I \subseteq V$ that has probability p_I of igniting. In other words, with probability p_I, I becomes the starting point of a wildfire in the landscape. Once an ignition scenario is activated, all nodes reachable from the scenario nodes will burn. Our goal is to remove edges from the graph in order to protect nodes from this burning, knowing that it costs c_e to remove edge e and that we can spend at most B on removal. So, $Y \subseteq E$ protects node u from scenario I if Y contains an edge of every path connecting u and i, for all $i \in I$, and Y is a feasible solution of GP if $c(Y) \leq B$.

As previously stated, we will be mostly concerned with preventative action: we wish to remove edges *before* an ignition occurs so as to maximize the expected value protected by these edges. With this in mind, the objective of Graph Protection is to maximize the

expected value of protected vertices. For $Y \subseteq E$, this can be stated as

$$\max_{\substack{Y \subseteq E: \\ c(Y) \leq B}} \sum_{I \in \mathcal{I}} \sum_{\substack{u \in V: \\ Y \text{protects} \\ u \text{ from } I}} p_I w_u. \tag{1.1}$$

We call the above the *expected protected value* of Y.

This preventative, stochastic version of Graph Protection is the main problem investigated in this work. We call this problem the *single-stage stochastic Graph Protection* problem. As discussed in [40], one can also consider other reasonable variants of the problem. For instance, in the 2-stage version of the problem, actions are possible both before and after an ignition occurs. In such a situation, actions taken subsequent to a fire are representative of, for example, extinguishing the spread via firefighting resources. It is clearly advantageous to implement such actions, as there is certainty in knowing which nodes of which value are in danger. Therefore, this model becomes interesting when it is costlier to make these responsive decisions (often called recourse actions), and the problem then involves the trade-off between these actions and the less-expensive actions when only stochastic information is available. This problem is known as the *2-stage Graph Protection* problem, and is one of the main focuses of [40].

Another variation of the problem is when edges can only be removed after the break out of a fire, which may be the case if no probabilistic data is known, or no preventative resources prior to ignition are available. In other words, this version is the special case when there exists a single ignition scenario and the ignition probability of the scenario is equal to 1. We call this the *deterministic Graph Protection* problem.

When the ignition scenario contains exactly one node, single-stage deterministic Graph Protection is equivalent (in terms of exact optimization) to the Minimum-Size Bounded-Capacity Cut problem introduced by Hayrapetyan et al. [24]. They showed that this problem is NP-hard even when the input graph is a tree via a reduction from the Knapsack problem. Thus, it is unlikely that an exact, polynomial-time algorithm for GP exists. For such problems, a common approach is to construct *approximation algorithms* for them, where optimality is compromised for efficiency by finding, in polynomial-time, a suboptimal solution with value of provable quality (see e.g. [45]). Formally, for a maximization problem with optimal value OPT, an α-*approximation algorithm*, with $\alpha \in (0, 1)$, is a polynomial-time algorithm that returns a feasible solution with value at least αOPT; α is often called the *approximation ratio* or *approximation factor* of the algorithm. In the case of GP and similarly budgeted problems, we may also allow ourselves to violate the budget by a certain factor. We say than an algorithm is an (α, β)-*approximation algorithm*, where $\beta \geq 1$, if it always returns a solution of value at least αOPT and cost at most βB.

1.3 Our Contribution

In this thesis, we investigate the general stochastic single-stage Graph Protection problem where scenarios may contain an arbitrary number of nodes, a setting mostly left open by the work of Shmoys and Spencer [40]. While we focus on the case where the graph is a tree, some of our results also apply to general graphs (as pointed out below).

We consider two ways by which the underlying ignition scenarios may arise, which bring about two variants of Graph Protection. In the first setting, the scenario distribution is an explicitly given distribution over (a polynomial number of) arbitrary subsets of nodes; we call this problem the *Graph Protection with Polynomial Scenarios* (GPwPS) problem. In the second setting, each node has an independent chance of being an ignition point, and this implicitly specifies the scenario distribution (which now has support of exponential size); we call this the *Graph Protection with Independent Activation* (GPwIA) problem.

Our main positive result is the design of constant-factor approximation algorithms for GPwIA on trees. A notable aspect of these algorithms is a novel use of a component-based formulation of the problem. We develop a randomized rounding procedure that takes a fractional solution of the LP relaxation of this formulation, and produces an integral solution that achieves a bicriteria $(O(1), O(1))$-approximation guarantee with constant probability; that is, the expected value of the integer solution is a constant fraction of the expected value of the fractional solution, and its cost is a constant factor of the cost of the fractional solution. We then show that this can be combined with an enumeration scheme to obtain a unicriterion constant-factor approximation. Although our approximation algorithms only apply to the GPwIA problem on trees, this limitation is a result of the difficulty in solving the LP. The rounding scheme is oblivious to the underlying probability distribution or graph structure, and thus may have future applications to related versions of the problem.

In contrast, for GPwPS, we show that the problem becomes rather hard to approximate, even in the case when each scenario has 2 ignition nodes and the input graph is a tree. More precisely, we show that GPwPS is at least as hard as the Densest k-Subgraph problem, which is known in the literature as being notoriously difficult to approximate, by giving an approximation-preserving reduction from Densest k-Subgraph to GPwPS.

We note that, as in [40, 14], our results for GPwIA can be extended from trees to general graphs by using the cut-based tree decomposition of Räcke [35]. The main idea of this extension involves constructing a distribution of trees based on the given graph, such that the value of the cuts is approximately preserved. We then solve the problem on each tree of the distribution and choose the best solution. The cost of the partition on the graph is at most a $O(\log n)$ factor of the cost of the partition on the tree, and this yields a $(O(1), O(\log n))$-approximation.

1.4 Related Literature

As previously mentioned, the Graph Protection problem we investigate was proposed in [40]. In this section, we first discuss work related to GP in the area of optimization models for wildlife management, and then more broadly in network diffusion and stochastic optimization. Finally, in Section 1.4.3 we discuss the previously known results for GP and related problems.

The prevention and control of catastrophic wildfires is an increasingly important topic in forestry and ecosystem management, and accordingly there exists extensive literature pertaining to the area. In recent decades, the burning of wildlands has been uncharacteristically vast, severe, and destructive (see e.g. [1]), and is partly due to the seemingly contradictory modern practice of fire elimination, which causes an unnatural buildup of fuels within a landscape (see e.g. [2]). Various *fuel reduction treatments* have been implemented to control the spread of such wildfires (see e.g. [34, 39]), and optimization models have been proposed in the literature as a natural tool to gain insights in the strategic allocation and placement of these treatments in the face of limited resources (e.g. [18, 44]). Finney [18] introduces a model that interprets an optimal placement of treatments as those which delay the rate of wildfire spread the most. The placement of these treatments is focused on areas prone to "problem" fires, the fires that grow the largest and the fastest, as these fires are, they argue, more likely to be pervasive and destructive. A notably different model, given by Wei, Rideout, and Kirsch [44], splits the landscape into cells, and simulates fire growth via probabilistic information describing how likely a burning cell will ignite adjacent cells. With this model, they formulate the problem as a mixed integer program. As noted in [40], this formulation relies on a questionable assumption that the conditional probabilities of spread can be represented by linear functions.

The models of Finney and Wei et al. both account for uncontrollable environmental factors, such as wind direction and landscape composition. However, the models noticeably differ in their simulation of wildfire spread: Finney focuses on the spatial properties of growth, while Wei et al. prioritize the stochastic nature of transmission. Both of these aspects are clearly important when considering wildfire progression, and thus it may be beneficial to consider alternative models which combine these aspects in a more congruous way. Establishing such a model was the main motivation of the Graph Protection problem introduced by [40].

Graph Protection lies within the general area of *computational sustainability*, an interdisciplinary field that attempts to apply the tools of computer science and operations research to questions concerning sustainable development (see e.g. [21]). Computational sustainability has contributed to the understanding of a large number of environmental and economical questions, including global warming, invasive species control, poverty mitigation, and sustainable energy use. The spatial and stochastic properties of wildfire management generalize to many problems in conservation, and it is straightforward to

apply variants of the Graph Protection model in such cases. One comparable problem is that of fragmenting a landscape graph in order to establish conservation regions that are used to protect endangered species. This problem belongs to the broad class of spatial conservation prioritization (see e.g. [32]).

1.4.1 Network Diffusion

Graph Protection also belongs to a class of problems associated with graph *diffusion*, where the spread of an infectious process through a network is modeled within the language of graph theory. There exists a large body of related literature on diffusive graph processes, due in part to the variety of phenomena it models, including contagious diseases and word-of-mouth spread of ideas. The study of such models within the framework of approximation algorithms is also prevalent in the literature.

One well-studied problem similar to Graph Protection is the Firefighter problem (see e.g. the survey by Finbow and MacGillivray [17]). Like in Graph Protection, the goal of this problem is to prevent an infectious process (e.g. a fire) from spreading within a network. The problem begins with the ignition of a node (or set of nodes). However, unlike in GP, we protect adjacent nodes from burning by "vaccinating" such nodes. This vaccination takes place during a sequence of time steps. At each such step, nodes adjacent to a fire will ignite unless they are vaccinated, and this cascading burn continues until no more spread is possible. Anshelevich et al. [5] present a number of approximation results for various objectives within this model, including the maximization of nodes protected under a budget and the minimization of cost of protecting a given set of nodes. Recently, Spencer generalized these results to problems of computational sustainability involving the spread of invasive species [42].

Rather than preventing an infectious process, it is sometimes of interest to maximize the spread, for example if the goal is to have a large portion of a social network adopt a new product via viral marketing. Domingos and Richardson [12, 38] pose the problem of maximizing such a spread, where the idea is to pick a set of initial "seed" individuals so that their adoption influences others to adopt, resulting in a cascading effect of influence and usage. The challenge is to pick a set of seeds that maximizes this influence. Kempe, Kleinberg, and Tardos [28] showed that under a number of different cascading models, the objective function of this graph problem is submodular and thus the well-known greedy algorithm of Nemhauser, Wolsey, and Fisher [33] can be used to obtain a $(1 - 1/e)$-approximation.

1.4.2 Stochastic Optimization

As previously mentioned, Graph Protection is a problem in *stochastic optimization*, a paradigm of mathematical optimization that attempts to model problems involving uncertainty (see e.g. [9]). In these problems, part of the input data is typically given as a

probability distribution over a set of possible scenarios. A well-known model for stochastic optimization is the *2-stage recourse model*, where initial decisions are made based on probabilistic data of the input (the fist stage), and recourse actions are taken once a scenario is realized (the second stage). Recourse actions usually represent quick decisions made immediately following a realized scenario, and are hence more expensive than actions taken prior. Therefore, balancing decisions between the first and second stage is the main challenge of the problem. The goal is often to minimize the *expected* cost of a solution, which is the sum of decision costs in the first stage and the expected cost of decisions for all possible realized scenarios in the second stage. The 2-stage GP problem falls within this framework.

There are a number of ways the scenario distribution can be represented. A straightforward approach is to assume the entire distribution is given as input, say, as a list of scenarios paired with their non-zero probabilities of occurrence. However, this assumption is not always practical as the support is often of exponential size; for example, in Graph Protection, the scenario distribution is over subsets of nodes, and thus the support can easily be of size exponential in n. We are generally interested in algorithms that have running time polynomial in the non-stochastic input size, and therefore in order to have full knowledge of the distribution, the support size must also be such a polynomial. Although this assumption is quite restrictive, it has led to a number of results (see e.g. the survey by Swamy and Shmoys [43]). One relevant example is the GP variant when each ignition scenario is a single node, as studied in [40]. A more general way of modeling the distribution is the *black-box model* originally given by Gupta et al. [23], where the only access to the distribution is through an oracle which returns a scenario sampled according to the distribution.

In this work, we consider the stochastic model of Graph Protection where each node has an independent chance of ignition. Although this assumption fails to account for possible correlations between ignition scenarios (which may exist due to, for example, spatial properties of the landscape), it does allow for a compact representation of an exponential number of scenarios, and similar assumptions have appeared in the literature. For example, Möhring et. al [31] investigate a scheduling problem in which the given jobs have independently random processing times. Immorlica et al.[26] give approximation algorithms for a number of stochastic combinatorial optimization problems with independent distributions, including the stochastic variant of Vertex Cover. Recently, Agrawal et al. [4] investigated how independence of probabilities contrasts with correlated distributions, and show that in some cases assuming the former yields a good approximation of the latter. There are various other works in Computer Science that use the independent-activation model; see the Swamy-Shmoys survey [43] for examples.

Linear programs have proven useful in approximating a large number of stochastic optimization problems. This application is analogous to the fruitful approximations of deterministic problems provided by LP relaxations. However, often unlike their determin-

istic counterparts, 2-stage stochastic linear programs can be exponential in the size of the non-stochastic input, due to their encoding of all stochastic information of the problem. Because of this blowup in size, they are often more difficult to solve. In fact, Dyer and Stougie [13] showed that such LPs are #P-hard to solve in the black-box model. Despite this complexity, Shmoys and Swamy [41] provide a general framework that applies LP-based approximation algorithms of deterministic problems to their stochastic analogue.

1.4.3 Prior Work on Graph Protection and Related Problems

Shmoys and Spencer [40] mainly focus on the 2-stage Graph Protection problem on trees when each ignition scenario consists of a single ignition point. They present a bicriteria $\big(1-(1-\frac{1}{2d})^{2d}, 2\big)$-approximation for this specific problem, where $d \leq n$ denotes the diameter of the tree. (Recall that in the 2-stage problem, edges can be cut both before and after (at an inflated cost) a scenario realizes, and the total cost of edges cut in each scenario must be at most the given budget B.) Their result is based on rounding the solution of a natural LP-formulation of the problem using *pipeage rounding*, a technique introduced by Ageev and Sviridenko [3].

As observed by [40], the single-stage stochastic GP problem on trees (i.e. the above problem when there are no second-stage decisions), where each ignition scenario contains a single node, is actually a special case of the Maximum Coverage with a Knapsack Constraint (MCKC) problem, and thus applying Ageev and Sviridenko's result produces a $\big(1-(1-\frac{1}{d})^{d}\big)$-approximation [3]. They also observe that this algorithm can be combined with the deterministic single-node PTAS of Hayrapetyan et al. [24] in order to obtain a unicriterion 0.387-approximation for the 2-stage problem. This algorithm works by spending all of the budget in either the first or second stage and then choosing whichever solution yields the better value. We note that neither the reduction to MCKC for the single-stage problem, nor the above decoupling for the 2-stage problem, seem to work in the setting when ignition scenarios contain multiple nodes. In this setting, the protection of a node can now require multiple edges to be cut, and in the 2-stage setting a node may end up being protected due to edges being cut in both stages, making it unclear how to attribute protection to a specific decision.

We now discuss other budgeted cut problems that were considered in the approximation algorithm literature prior to [40], and are similar to, or special cases of, Graph Protection. Hayrapetyan et al. [24] introduced the Minimum-Size Bounded-Capacity Cut (MinSBCC) problem, wherein we seek an s-t cut of cost at most B in an undirected graph that minimizes the size of the s-side of the cut. It is easy to see that MinSBCC is essentially the deterministic GP problem with a single source, except that in MinSBCC we seek to *minimize* the number of *unprotected* nodes. Hayrapetyan et al. obtained a bicriteria approximation algorithm for MinSBCC as well as a PTAS for when the input graph has bounded treewidth. This implies a PTAS for the deterministic GP problem when the input graph has bounded

treewidth, as discussed in Theorem 1 of [40].

Engelberg et al. [14] introduced two budgeted versions of the Multiway Cut problem, both of which generalize MinSBCC (when one considers the objective of maximizing the size of the t-side of the cut). In particular, in the Weighted Budgeted Separating Multiway Cut (wBSMC) problem that they introduce, we are given edge costs, a set D of terminals, and a weight function $w : D \times D \to \mathbb{Z}^+$, and the goal is to remove a set of edges of total cost at most B so as to maximize the total weight of separated terminal pairs. Engelberg et al. give a 1/3-approximation algorithm for this problem on trees. Observe that the special case of stochastic single-stage GP where each scenario consists of a single node can be reduced to wBSMC as follows: we simply create for each node u and ignition scenario $\{i\}$, a terminal pair (u, i) with weight $p_i w_u$, where p_i is the probability of ignition of i and w_u is the value of node u.

Finally, using the tool of Räcke embeddings [35], the results for GP on trees and those of Engelberg et al. on trees can be extended to obtain bicriteria guarantees for general graphs, where the budget is violated by an $O(\log n)$ factor (in addition to any violation incurred by the algorithm on trees).

Chapter 2

Approximation Results for Arbitrary Ignition Scenarios

2.1 Introduction

In this chapter, we investigate the version of the single-stage stochastic Graph Protection problem when each ignition scenario may contain an arbitrary number of nodes (specifying the ignition points). Recall that the input of this problem is a graph $G = (V, E)$ with non-negative node values w_u, $u \in V$, and non-negative edge costs c_e, $e \in E$, a budget $B \geq 0$, as well as a description of the *scenario distribution*, that is, the ignition scenarios and their probabilities. The goal is to find a set of edges of total cost at most B so that the removal of such edges maximizes the expected value protected when a random subset of nodes ignites. We may assume that $c_e \leq B$ for all $e \in E$, as otherwise we can contract e. We focus mostly on the case where G is a tree, but some of our results also apply to arbitrary graphs (which we point out below).

An important detail left out of the above description is how we specify the scenario distribution. This work focuses on two different ways of doing this, which yield the following two variants of the Graph Protection problem.

- In the first model, an ignition scenario may be an arbitrary subset of V, and the scenario distribution is specified by explicitly listing these subsets paired with their probabilities of occurrence. As previously noted, while this model allows for ignition scenarios with arbitrary combinatorial structure, one noticeable drawback is that specifying a distribution with an exponential number of scenarios results in an exponential blow-up in the input size, rendering the notion of "polynomial-time" algorithm ineffective. This model is therefore restricted to the *polynomial-scenario model*, where the input distributions are only those with polynomial support. We call this variant of Graph Protection the *Graph Protection with Polynomial Scenarios* (GPwPS) problem.

- For the second model, we apply to Graph Protection the *independent-activation model*, which in this context means that each node has an independent chance of ignition. As previously mentioned, this model has often been considered in the Computer Science literature (e.g. [31, 26, 4]), and allows for a succinct capturing of an exponential number of scenarios. We call this resulting variant the *Graph Protection with Independent Activation* (GPwIA) problem.

Our main contribution is the design of constant-factor approximation algorithms for the GPwIA problem when the input graph is limited to a tree. As previously noted, the only results known prior [40] were for the cases where (i) each scenario consists of a *single* ignition point, or (ii) when there are a *constant* number of scenarios, each having a *constant* number of ignition points. In contrast, notice that in GPwIA, we deal with an *exponential* number of scenarios, and a scenario may contain an arbitrary number of ignition points. Given an instance of GPwIA, let OPT be the optimal expected protected value. Our results are summarized in the following theorems.

Theorem 2.1.1. *There exists a randomized bicriteria approximation algorithm for the GPwIA problem on trees that succeeds with constant probability and conditioned on success, returns a solution with expected protected value at least* $0.0095 \cdot \mathrm{OPT}$ *and cost at most* $2B$*.*

Theorem 2.1.2. *There exists a randomized approximation algorithm for the GPwIA problem on trees that, when given* $\epsilon \in (0, 1)$*, succeeds with constant probability and conditioned on success, returns a solution with expected protected value at least* $0.0023(1 - \epsilon)\mathrm{OPT}$ *and cost at most* B*.*

We remark that our goal was not to optimize the constants in Theorems 2.1.1 and 2.1.2 above, but was instead focused on keeping exposition simple.

We also show that a similar approximation for GPwPS on trees is unlikely, as such a result implies the same approximation for the notorious Densest k-Subgraph problem, which is believed to not exist. This result is formally given below.

Theorem 2.1.3. *An* α*-approximation algorithm for* GPwPS *on trees implies an* α*-approximation algorithm for* Densest k-Subgraph*.*

The outline of this chapter is as follows. In Section 2.2, we show that a natural linear program relaxation of GP (which is the one used by [40]) admits a large integrality gap even when there is only one scenario (i.e. the single-stage deterministic problem) containing 2 ignition points. This gap motivates a new configuration-style formulation of the problem (valid for all graphs), which interprets a solution as a set of disjoint connected components. Configuration linear programs have proven to be robust tools in the design of approximation algorithms (e.g. [27, 6, 11, 15, 19]), although their usage is somewhat sparse; a possible explanation for this is that the formulations may contain an exponential number of variables

and/or constraints due to their set-partitioning nature. We discuss this formulation and its LP relaxation in Section 2.3.

In Section 2.4, we design a randomized rounding scheme that rounds a fractional solution x of the component-based LP with objective value W to a random integer solution that satisfies, with constant success probability, the following bicriteria guarantee (conditioned on success): the expected protected value is $\Omega(W)$ and the expected cost is $O(B)$. This guarantee is formally given in Theorem 2.4.3. The design and analysis of our rounding algorithm builds upon some ideas from the work of Feige [15] in the disparate setting of combinatorial auctions. A notable benefit of our scheme is that no structure of the graph or underlining ignition probabilities is assumed, and thus if one can (approximately) solve the linear program for any given class of graphs and/or distribution model, our rounding result can be applied.

In Section 2.5, we focus on the GPwIA problem and show that we can obtain a PTAS for solving the component-based LP on trees. This result allows us to apply our rounding scheme, yielding the approximation algorithm of Theorems 2.1.1; we summarize this in Section 2.6. In Section 2.7 we discuss how to modify this algorithm to obtain the unicriterion approximation of Theorem 2.1.2. In Section 2.8, we prove the inapproximability result of Theorem 2.1.3 for GPwPS. Since, as noted above, our rounding algorithm works for any graph and any scenario distribution, this inapproximability result also means that it is hard to solve the component-based LP for GPwPS. This will also follow more directly from the reduction we describe.

2.2 Integrality Gap of Natural LP

For the Graph Protection problem when arbitrarily sized ignition scenarios are allowed, let $\mathcal{I} \subseteq 2^V$ be the set of these scenarios and let p_I be the ignition probability of scenario $I \in \mathcal{I}$. The natural linear program relaxation of this problem, assuming the input graph is a tree, is given below; it is the generalization of the single-source LP used in Shmoys and Spencer [40]. For each ignition scenario $I \in \mathcal{I}$ and each vertex $v \in V$, the variable $x_{v,I}$ represents whether v is protected when scenario I ignites. Each edge e is assigned a variable y_e which corresponds to whether e is part of the solution, i.e. whether e is removed from the graph for node protection. The set of edges in the path between nodes i and v is denoted as $P_{i,v}$.

$$\text{maximize} \quad \sum_{I,v} p_I w_v x_{v,I} \tag{LP1}$$

$$\text{subject to:} \quad \sum_{e \in P_{i,v}} y_e \geq x_{v,I} \quad \forall i \in I, I \in \mathcal{I}, v \in V \tag{2.1}$$

$$\sum_{e \in E} c_e y_e \leq B \tag{2.2}$$

$$1 \geq x_{v,I} \geq 0 \qquad \forall\, I \in \mathcal{I}, v \in V$$

$$y_e \geq 0 \qquad \forall\, e \in E$$

The following proposition shows that the integrality gap of the above LP is bounded below by order $|V| = n$, implying that it is inadequate for our use in approximating the problem. The example used to show this gap contains only one scenario that has two ignition points. Thus, it remains a bad example for any scenario-distribution model that is rich enough to capture this scenario.

Proposition 2.2.1. *The integrality gap of* (LP1) *is* $\Omega(n)$.

Proof. Consider the following instance of the problem, as shown in Figure 2.1. The graph consists of a single path on n vertices whose endpoints are the two ignition points i_1 and i_2 of the only ignition scenario $I = \{i_1, i_2\}$ with $p_I = 1$. Let $B = 1$, the value of all vertices be 1, and the single edges incident to i_1 and i_2 be labelled e_1 and e_2, respectively. For a single vertex $v \notin I$, let its two incident edges have cost $1/2$, and let all other edges have cost $(1+\varepsilon)/2$.

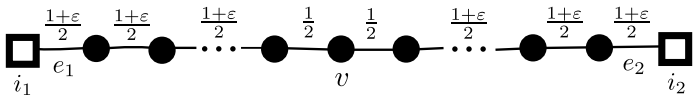

Figure 2.1: Instance of the stochastic single stage problem with $B = 1$ that exhibits an integrality gap of $\Omega(n)$.

Clearly the optimal integral solution consists of removing the two edges incident to v, as choosing any other edge makes all other edges unaffordable, resulting in no nodes being protected. This solution gives a protected value of 1. A feasible fractional solution, however, consists of setting y_{e_1} and y_{e_2} to $1/(1+\varepsilon)$, yielding an objective value of $(n-2)/(1+\varepsilon)$ which proves the claim. ■

2.3 Component Based Formulation

We now discuss a new configuration-style formulation of the Graph Protection problem, based on the interpretation of a solution as a set of disjoint connected components of the input graph; we emphasize that this formulation is valid for any arbitrary graph. Recall that a feasible solution of GP consists of a set of edges of total cost at most B. The removal

of such edges partitions the graph into a set of connected components; these components may contain nodes of different ignition scenarios. If a component contains a node of an ignition scenario, and this scenario ignites, all nodes contained in the component will burn. On the other hand, if no ignition occurs within a component, all nodes will be protected. Thus, the expected protected value of a single component is the weight of the component multiplied by the probability that no scenario intersecting with the component ignites.

We define this expected protected value of a connected component A as the following weight function, where I_A is the random event that no ignition scenario intersecting component A ignites.

$$\Pi(A) := \ w(A) \cdot \Pr[I_A]. \tag{2.3}$$

Let $Y \subseteq E$ be a feasible set of edges that achieves the optimal expected protected value OPT, and let $\mathcal{O}^*$ be the set of connected components of G formed by removing Y. Clearly, $\sum_{A \in \mathcal{O}^*} \Pi(A) = \text{OPT}$.

The *boundary* of a component A is the subset of edges of G with exactly one endpoint in A, and is denoted as $\delta(A)$. We call the cost of such a boundary, i.e. $c(\delta(A))$, the *boundary cost* of A. Let $\mathcal{C}$ be the set of components of G with boundary cost at most B. With our component-based interpretation of the optimal solution, we formulate the following linear program, where we have a variable x_A for every component $A \in \mathcal{C}$ indicating if A is a component of our solution.

$$\text{maximize} \quad \sum_{A \in \mathcal{C}} \Pi(A) x_A \tag{P}$$

$$\text{subject to:} \ \sum_{A \in \mathcal{C}} \left(\tfrac{c(\delta(A))}{2} \right) x_A \leq B \tag{2.4}$$

$$\sum_{A : v \in A} x_A \leq 1 \quad \forall\, v \in V \tag{2.5}$$

$$x_A \geq 0 \quad \forall\, A \in \mathcal{C}$$

Constraint (2.4) encodes the budget constraint of the problem; a set of components is a feasible integral solution only if their total boundary cost is at most $2B$ (the factor of 2 is a result of each edge being in exactly two component boundaries). Constraint (2.5) enforces an integral solution to consist of only disjoint components. Clearly, the optimal integral solution $\mathcal{O}^*$ is feasible for the above LP, and thus its value is at most the value of the optimal fractional solution.

We note that if the set of components are not constrained to those of $\mathcal{C}$, that is, if variables x_A are allowed for components A that have arbitrary boundary cost, then it can be shown that (P) has a similar integrality gap as (LP1). Thus it is crucial for us to consider components with boundary cost at most B. We also note that the number of variables of (P) is exponential in n, implying standard techniques cannot be used to solve

this LP in polynomial time. In Section 2.5 we discuss how to approximately solve (P) despite this setback.

In devising our rounding algorithm, we will exploit the following useful property of the weight function Π, which resembles that of *fractional subadditve utility functions* studied by Feige [15]. A form of this property appears earlier in game theory and economics literature as part of the classical Bondareva-Shapley Theorem [10, 30]. We show this property of Π by first extending the function to arbitrary subsets of V. For a given set $S \subseteq V$, let $C(S)$ be the set of connected components of the subgraph of G induced by S. Then,

$$\Pi(S) := \sum_{S' \in C(S)} \Pi(S').$$

Lemma 2.3.1. *Let $A \subseteq V$. Let z be a vector indexed by 2^A such that $z_S \geq 0$ for all $S \subseteq A$ and $\sum_{S:v \in S} z_S \geq 1$ for all $v \in A$. Then,*

$$\Pi(A) \leq \sum_{S \subseteq A} z_S \Pi(S).$$

Proof. Clearly for any component $S' \subseteq A$, all ignition scenarios that intersect S' will intersect A, and therefore $\Pr[I_{S'}] \geq \Pr[I_A]$. So,

$$\begin{aligned}
\sum_{S \subseteq A} z_S \Pi(S) &= \sum_{S \subseteq A} z_S \sum_{S' \in C(S)} w(S') \Pr[I_{S'}] \\
&\geq \sum_{S \subseteq A} z_S \sum_{S' \in C(S)} w(S') \Pr[I_A] \\
&= \Pr[I_A] \sum_{S \subseteq A} z_S \sum_{v \in S} w_v \\
&= \Pr[I_A] \sum_{v \in A} w_v \sum_{S: v \in S} z_S \\
&\geq \Pr[I_A] \sum_{v \in A} w_v \\
&= \Pi(A).
\end{aligned}$$

■

As previously stated, Lemma 2.3.1 shows that Π is a set function with a very similar property to *fractionally subadditve set functions* [15]. Note however that Π is *not* a monotone function (since adding an ignition point can decrease $\Pr[I_A]$). We leverage the above property of Π to build upon Feige's 1/2-approximation algorithm for maximizing

welfare in combinatorial auctions with fractionally subadditve valuations [15]. The following proposition is a restatement of Proposition 2.3 of Feige, applying it to our set function. The proof given below is essentially the same as the one found in [15], and is included for the sake of completeness.

Proposition 2.3.2 ([15])**.** *Let $k \geq 1$ be an integer. For a set $A \subseteq V$, consider a distribution over subsets of A such that each vertex of A is included in $S \subseteq A$ with probability at least $1/k$. Then, $\mathbb{E}[\Pi(S)] \geq \Pi(A)/k$.*

Proof. Let p_S be the probability (according to the distribution) that $S \subseteq A$ is selected. Note that $\sum_{S \subseteq A} p_S = 1$. The probability that v is included in S is exactly the sum of the probabilities of all sets containing v, and thus $\sum_{S:v \in S} p_S \geq 1/k$ and $\sum_{S:v \in S} k p_S \geq 1$ for all $v \in A$. Then from Lemma 2.3.1 we have that $\Pi(A) \leq \sum_{S \subseteq A} k p_S \Pi(S)$ and the proposition follows. ■

The algorithms of Theorems 2.1.1 and 2.1.2 essentially consist of two parts: finding a near-optimal fractional solution of (P), and rounding said solution to an integral solution in such a way that not too much protected value is lost. The methods, techniques, and tools used for actualizing these parts constitute a major portion of the remainder of this work.

2.4 Rounding Fractional Solutions

We now describe a randomized algorithm that rounds a feasible fractional solution of (P) to an integral solution. Let x be such a fractional solution and let W be its objective value; we assume that x is polynomial in size (we show later that when we apply the rounding routine, this will be the case). We will first give a randomized procedure that rounds x to an integral solution that *in expectation* has expected protected value $\Omega(W)$ and cost $O(B)$. However, this rounding does not guarantee that the actual integral solution meets *both* bounds simultaneously, so we modify it in Section 2.4.1. Along the way, we will introduce constants that govern the "success" of the algorithm. In Section 2.6, we instantiate these constants, yielding the approximation algorithm of Theorem 2.1.1.

The rounding of x will consist of two stages. In this first stage, we pick a set of tentative components based on x, some of which may intersect. Then, we convert this tentative set to an integral solution by assigning each intersecting region to a single component. Let $\mathcal{A}$ be the set of components corresponding to the entries of x. The initial rounding of the fractional solution is as follows: Given a scaling factor $\sigma \in (0, 1]$, add each $A \in \mathcal{A}$ to the collection of *tentative components* $\mathcal{T}$ independently with probability σx_A. Let X_A be the random variable indicating the value component A contributes to this initial

allocation. Then $X_A = \Pi(A)$ with probability σx_A and $X_A = 0$ otherwise. From linearity of expectation, the expected protected value of $\mathcal{T}$ is

$$\mathbb{E}\left[\sum_{A\in\mathcal{T}} \Pi(A)\right] = \sum_{A\in\mathcal{A}} \mathbb{E}[X_A] = \sigma \sum_{A\in\mathcal{A}} x_A \Pi(A) = \sigma W.$$

Let $\{A_1, \ldots, A_\ell\} = \mathcal{T}$ be the tentative components allocated with this randomized rounding. As previously noted, the components of $\mathcal{T}$ are not necessarily disjoint, and therefore may not be a feasible integral solution. Our goal is to modify these tentative components so that afterwards each vertex is contained in at most one of them. We say R is an *atomic region* of $\mathcal{T}$ if (i) $R \cap A_i \in \{\emptyset, R\}$ for all $i = 1, \ldots, \ell$; (ii) $R \subseteq A_i$ for some $i = 1, \ldots, \ell$; and (iii) R is a maximal set satisfying (i) and (ii).

The final part of the rounding scheme is to "assign" each atomic region R to a single tentative component. Let $\mathcal{T}(R)$ be the set of tentative components that contain R. Note that $|\mathcal{T}(R)| \leq n$ (i.e., it's polynomially bounded) because the atomic regions are disjoint. We assign R to a component of $\mathcal{T}(R)$ uniformly at random. This produces a set $S_i \subseteq A_i$, for each $i = 1, \ldots, \ell$, where S_i is the union of the atomic regions that were assigned to A_i. The final integral solution $\tilde{x}$ consists of the components compromising $S_1, \ldots, S_\ell$, that is, we set $\tilde{x}_A = 1$ for every component A contained in $S_1, \ldots, S_\ell$.

Lemma 2.4.1. *The expected value of the integral solution obtained by the above randomized assignment scheme is at least $\sigma W/2$.*

Proof. Following Feige [15], we interpret the above randomized algorithm in the following way. Assuming that A' is a tentative component, the final assignment of subset S' to A' can be viewed as the following randomized process: First, pick A' as a tentative component. Then, pick all other tentative components, and assign each atomic region of A' to a tentative component as described above. The atomic regions assigned to A' make up S'.

For $u \in A'$, let n_u be the number of tentative components that contain u. Then,

$$\begin{aligned}
\mathbb{E}[n_u \mid A' \in \mathcal{T}] &\leq 1 + \sum_{A: v\in A} \Pr\big[A \text{ is tentatively picked}\big] \\
&= 1 + \sigma \sum_{A: v\in A} x_A \\
&\leq 1 + \sigma, \qquad (2.6)
\end{aligned}$$

where the final inequality comes from constraint (2.5) of the LP. Recall that we assign an atomic region R to a tentative component that contains R uniformly at random. In other words, we assign a node $u \in R$ to a tentative component containing u uniformly at

random. So, conditioned on u being in A', u is assigned to A' with probability $1/n_u$. Thus,

$$\begin{aligned}\Pr\big[u \in S' \mid A' \in \mathcal{T}\big] &= \mathbb{E}[1/n_u \mid A' \in \mathcal{T}] \\ &\geq 1/\mathbb{E}[n_u \mid A' \in \mathcal{T}] \qquad (2.7) \\ &\geq 1/(1+\sigma) \\ &\geq 1/2,\end{aligned}$$

where the inequality (2.7) is due to Jensen's inequality. Then, applying the above to Proposition 2.3.2 implies that $\mathbb{E}[\Pi(S')] \geq \Pi(A')/2$. From this it is easy to see that the final integral solution will recover at least $1/2$ of the expected value of the tentative sets, which is σW. ■

Lemma 2.4.2. *The expected cost of the integral solution is at most $\sigma 2B$.*

Proof. Let Y_A denote the random variable for the cost A contributes to the tentative solution. So, $Y_A = c(\delta(A))$ with probability σx_A and $Y_A = 0$ otherwise. Then, similar to before,

$$\mathbb{E}\big[\text{cost of } \mathcal{T}\big] = \sigma \sum_{A \in \mathcal{T}} x_A c(\delta(A)) \leq \sigma 2B, \qquad (2.8)$$

where the inequality comes from (2.4) of (P).

Let $\mathcal{R}$ be the set of all atomic regions obtained by the first step of the rounding scheme. We claim that for every $R \in \mathcal{R}$, every $e \in \delta(R)$ is found in some $\delta(A)$ for $A \in \mathcal{T}(R)$. Assume for contradiction that there exists some $e = uv$ such that $e \in \delta(R)$ with $u \in R$, and that there exists no A where $u \in A$ and $v \notin A$. Then $(R \cup \{v\}) \cap A = R \cup \{v\}$, which contradicts R being maximal. If there exists no A such that $u \in A$, then R is not an atomic region by definition. Therefore, since every edge is in at most two atomic region boundaries, the expected cost of our final solution is:

$$\sum_{R \in \mathcal{R}} c(\delta(R)) \leq 2 \cdot c\left(\bigcup_{R \in \mathcal{R}} \delta(R)\right) \leq 2 \cdot \sum_{A \in \mathcal{T}} c(\delta(A)). \qquad (2.9)$$

Combining (2.8) and (2.9) proves the claim. ■

2.4.1 Probability of Success

From Lemmas 2.4.1 and 2.4.2 we know that the expected protected value and expected boundary cost of our integral solution is at least a fraction of W and at most a multiple of B, respectively. However, this does not guarantee that the actual solution returned by this randomized scheme achieves both these bounds. Our goal is to show that the solution does have this property with constant probability.

We show this by analyzing the tentative allocation obtained after the first step of the randomized algorithm. The analysis makes use of the following version of the Chernoff bound.

Theorem (Chernoff Bound)**.** *Let $X_1, \ldots, X_n$ be independent random variables such that $0 \leq X_i \leq M$ for all i, and let $X = X_1 + \cdots + X_n$. Then for any $\epsilon \geq 0$,*

$$\Pr\big[X \leq (1-\epsilon)\mathbb{E}[X]\big] \leq \exp\left(-\frac{\epsilon^2}{2} \cdot \frac{\mathbb{E}[X]}{M}\right).$$

We assume that for each $A \in \mathcal{C}$ with $x_A > 0$, the expected protected value of A is not a significant portion of the fractional solution value. That is, we may assume there exists a $\gamma \in (0, 1]$ such that $\Pi(A) \leq \gamma W$ for all $A \in \mathcal{A}$. This is because if a component existed with value greater than γW, choosing this single component as a solution already yields a good approximation. We discuss this idea in more detail when we present our approximation algorithm in its entirety.

As before, let X_A be the independent random variable representing the value A contributes to our rounded solution. So, $X_A = \Pi(A)$ with probability σx_A and $X_A = 0$ otherwise. Let $X = \sum_{A \in \mathcal{C}} X_A$; recall that $\mathbb{E}[X] = \sigma W$. Since $0 \leq X_A \leq \gamma W$ for all A, the Chernoff bound implies, for $\epsilon_1 \geq 0$,

$$\Pr\left[X \leq (1-\epsilon_1)\sigma W\right] \leq \exp\left(-\frac{\epsilon_1^2}{2} \cdot \frac{\sigma}{\gamma}\right). \tag{2.10}$$

Similarly, if Y_A is the random variable for the contributed boundary cost of component A and $Y = \sum_{A \in \mathcal{C}} Y_A$, then for $\kappa > 0$, Markov's Inequality together with (2.8) implies

$$\Pr\big[Y \geq \kappa B\big] \leq \frac{2\sigma}{\kappa}.$$

After the initial randomized rounding of the fractional solution, the algorithm detects if either of the above events occur. If so, the algorithm rejects the solution and returns failure. Otherwise, we continue to the second stage of the rounding. So, from the union bound, the probability of continuing is at least

$$\phi_1 = 1 - \exp\left(-\frac{\epsilon_1^2}{2} \cdot \frac{\sigma}{\gamma}\right) - \frac{2\sigma}{\kappa}. \tag{2.11}$$

One last detail that needs to be resolved is that because of this newly realized possibility of failure, the second part of the rounding algorithm (when atomic regions are assigned to tentative components) occurs conditioned on the probability of acceptance, and thus the analysis of Lemma 2.4.1 does not entirely apply. In other words, if Ω_1 is the event that the initial component allocation is accepted, and Ω_A is the event that component A is a tentative component, then for $u \in A$, instead of bounding $\mathbb{E}[n_u \mid \Omega_A]$ as we did in (2.6), it is necessary to bound $\mathbb{E}[n_u \mid \Omega_A, \Omega_1]$. Let Ω_1^C be the event that the algorithm returns failure. Then, because

$$\mathbb{E}\big[n_u \mid \Omega_A\big] = \Pr\left[\Omega_1 \mid \Omega_A\right] \cdot \mathbb{E}\big[n_u \mid \Omega_A, \Omega_1\big] + \Pr\left[\Omega_1^C \mid \Omega_A\right] \cdot \mathbb{E}\big[n_u \mid \Omega_A, \Omega_1^C\big]$$

we have

$$\Pr\left[\Omega_1 \mid \Omega_A\right] \cdot \mathbb{E}\big[n_u \mid \Omega_A, \Omega_1\big] \leq \mathbb{E}\big[n_u \mid \Omega_A\big]. \tag{2.12}$$

Our goal now is to find a lower bound for $\Pr[\Omega_1 \mid \Omega_A]$. Let $X' = \sum_{A' \in \mathcal{A} \setminus \{A\}} X_{A'}$ and $Y' = \sum_{A' \in \mathcal{A} \setminus \{A\}} Y_{A'}$. Then,

$$\Pr\big[\Omega_1 \mid \Omega_A\big] \geq 1 - \Pr\big[X' + \Pi(A) \leq (1-\epsilon_1)\sigma W\big] - \Pr\big[Y' + c(\delta(A)) \geq \kappa B\big]. \tag{2.13}$$

For the first probability, note that by linearity of expectation, $\mathbb{E}[X'] = \mathbb{E}[X] - \sigma x_A \Pi(A)$. Also, it is easy to see that

$$\begin{aligned}\Pr\big[X' \leq (1-\epsilon_1)\sigma W - \Pi(A)\big] &\leq \Pr\big[X' \leq (1-\epsilon_1)\sigma\left(W - x_A\Pi(A)\right)\big] \\ &= \Pr\big[X' \leq (1-\epsilon_1)\mathbb{E}[X']\big]. \end{aligned}\tag{2.14}$$

Then, by the Chernoff bound and the fact $x_A\Pi(A) \leq \Pi(A) \leq \gamma W$, we obtain

$$\begin{aligned}\Pr\big[X' \leq (1-\epsilon_1)\mathbb{E}[X']\big] &\leq \exp\left(-\frac{\epsilon_1^2}{2} \cdot \frac{\sigma(W - x_A\Pi(A))}{\gamma W}\right) \\ &\leq \exp\left(-\frac{\epsilon_1^2}{2} \cdot \frac{\sigma(1-\gamma)}{\gamma}\right). \end{aligned}\tag{2.15}$$

For the latter probability of (2.13), if $\kappa > 1$, then again from Markov's inequality we have

$$\begin{aligned}\Pr\big[Y' \geq \kappa B - c(\delta(A))\big] &\leq \frac{\mathbb{E}[Y']}{\kappa B - c(\delta(A))} \\ &\leq \frac{\mathbb{E}[Y]}{(\kappa-1)B} \\ &\leq \frac{2\sigma}{\kappa - 1}. \end{aligned}\tag{2.16}$$

Therefore, combining (2.12)–(2.16) with (2.6) and setting

$$\phi_2 = 1 - \exp\left(-\frac{\epsilon_1^2}{2} \cdot \frac{\sigma(1-\gamma)}{\gamma}\right) - \frac{2\sigma}{\kappa - 1} \tag{2.17}$$

implies that

$$\mathbb{E}\big[n_u \mid \Omega_A, \Omega_1\big] \;\leq\; \frac{1}{\phi_2} \cdot \mathbb{E}\big[n_u \mid \Omega_A\big] \;\leq\; \frac{1+\sigma}{\phi_2}.$$

Thus, $u \in A$ gets assigned to the final $S \subseteq A$ with probability at least $\phi_2/(1+\sigma)$, and Proposition 2.3.2 implies that $\mathbb{E}[\Pi(S)] \geq \phi_2\Pi(A)/(1+\sigma)$.

These results are summarized in the following theorem.

Theorem 2.4.3. *Let x be a fractional solution of* (P) *with objective value W. Let ϵ_1, κ, γ, and σ be constants such that $\epsilon_1 \geq 0$, $\kappa > 1$, $\gamma, \sigma \in (0, 1]$, and ϕ_1 and ϕ_2, defined as*

$$\phi_1 = 1 - e^{-\epsilon_1^2 \sigma/(2\gamma)} - 2\sigma/\kappa, \quad \phi_2 = 1 - e^{-\epsilon_1^2 \sigma(1-\gamma)/(2\gamma)} - 2\sigma/(\kappa - 1)$$

are greater than 0. *Then, we can round x to a random integer solution $\tilde{x}$ with objective value $\tilde{w}$ and boundary cost $\tilde{c}$, such that letting Ω_1 denote the event that the algorithm returns success, we have:*

(i) $\Pr[\Omega_1] \geq \phi_1$,

(ii) $\mathbb{E}\big[\tilde{w} \;\big|\; \Omega_1\big] \geq \dfrac{\sigma(1-\epsilon_1)\phi_2}{(1+\sigma)} W$,

(iii) $\Pr\big[\tilde{c} \leq \kappa B \;\big|\; \Omega_1\big] = 1$.

As previously stated, in Section 2.6 we give values that make both ϕ_1 and ϕ_2 positive. Before this, we discuss in the following section how we actually find a fractional solution of the linear program.

2.5 Solving the Linear Program

As previously noted, the linear program (P) has an exponential number of variables, and thus cannot be directly solved in polynomial time. However, as there are only a polynomial number of constraints (namely, $n + 1$), we can solve the dual of the LP with the *ellipsoid method* in polynomial time, as long as we meet certain criteria. The dual (D) is given below, where the dual variable α corresponds to the budget constraint (2.4) and the variable β_u corresponds to the constraint (2.5) for each $u \in V$.

$$\text{minimize} \quad B\alpha + \sum_{u \in V} \beta_u \tag{D}$$

$$\text{subject to:} \;\; \alpha \cdot \tfrac{c(\delta(A))}{2} + \sum_{u \in A} \beta_u \geq \Pi(A) \quad \forall A \in \mathcal{C} \tag{2.18}$$

$$\alpha,\, \beta_u \geq 0 \qquad \forall\, u \in V$$

The ellipsoid method makes use of a *separation oracle* to check feasibility of a linear program. A separation oracle is a procedure that, when given a potentially feasible solution of a linear program, checks whether this solution is indeed feasible. Additionally, if the solution is infeasible, the separation oracle returns a constraint of the LP that is violated. The benefit of this method is that it finds an optimal solution in time independent of the number of constraints. This property is formally described as the theorem below.

Theorem 2.5.1 ([22]). *Consider the following linear program for $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, and $a_i \in \mathbb{R}^n$, $i = 1, \ldots, m$,*

$$\min_{x \in \mathbb{R}^n} \left\{ c^T x \ : \ a_i^T x \geq b_i \ \forall \, i = 1, \ldots, m, \ x \geq 0 \right\}.$$

Suppose there exists a separation oracle for the above LP with running time N, and suppose that the encoding length of each constraint $a_i^T \geq b_i$ is at most M. Then, the ellipsoid method finds an optimal solution of the LP in time $\text{poly}(n, N, M)$. *Furthermore, this process yields an equivalent linear program whose constraints are the polynomial number of violated inequalities found by the separation oracle.*

Thus, the ellipsoid method can solve (D) in polynomial time as long as we have a separation oracle for the constraints (2.18) that runs in polynomial time. To achieve this, we first define the following *density* ρ for a component A for a given candidate solution α, β:

$$\rho(A) := \frac{\mathit{\Pi}(A) - \beta(A)}{c(\delta(A))/2}, \tag{2.19}$$

where $\beta(A) := \sum_{u \in A} \beta_u$. Note that α, β is feasible if and only if $\alpha \geq \rho(A)$ for all $A \in \mathcal{C}$. Furthermore, if there exists any constraint that is violated, say $\rho(A') > \alpha$ for some A', then the constraint corresponding to the component with maximum ρ is also violated. Thus, if we can solve $\max_{A \in \mathcal{C}} \{\rho(A)\}$ in polynomial time, we can check whether the maximum density is at most α, resulting in a polynomial-time separation oracle.

Unfortunately, since ρ is essentially an arbitrary set function when we have arbitrarily sized ignition scenarios, this maximization problem is NP-hard. However, in the special case when the input graph is a tree and each node has an independent chance of igniting (i.e. the GPwIA problem), after suitably scaling the input data, we can efficiently solve the problem of maximizing ρ via dynamic programming; we describe this is Section 2.5.1. However, for more general scenario-distributions, even on trees, both the Graph Protection problem (i.e. GPwPS on trees) and the above density-maximization problem become Densest k-Subgraph hard, which we show in Section 2.8.

2.5.1 Independent Node Ignition on Trees

We now focus on the *Graph Protection with Independent Activation* (GPwIA) problem, which is the special case when the ignition of each node is an independent event. Our focus is also on the case when the input graph is a tree $T = (V, E)$. Let p_u be the probability that u ignites, and let $q_u = 1 - p_u$ be the probability that u does not ignite. Then, the probability that no ignition occurs in a component A is $q(A) := \prod_{u \in A} q_u$, and our $\mathit{\Pi}$ function becomes $\mathit{\Pi}(A) = w(A)q(A)$.

A main advantage of having independent probabilities over a distribution of ignition scenarios $\mathcal{I} \subseteq 2^V$ is that we can better approximate the maximum density $\rho(A)$ over

all components A. Intuitively, this is because smaller sub-problems are capable of being merged, as the probability of ignition in one component is independent of the probability of ignition in another, disjoint component. We will be confining the input graph of GPwIA to trees, and as such, we can use a dynamic program (DP) to search for components containing a node u by merging components of the subtrees of u. The following section is concerned with this algorithm.

Dynamic Program for Maximizing Density

Given an instance of GPwIA such that the input graph is a tree $T = (V, E)$, the dynamic programming algorithm works as follows. First, pick a root $r \in V$ of the tree, and direct all edges away from r. We process the nodes in a bottom-up fashion, processing all children of a node before considering the node itself. We will argue that the maximum-density component can be computed by keeping track of a bounded number of component-specifications (i.e. tuples of parameter values) at each node; these component-specifications can be updated using component-specifications of the children of the node. The subproblem at a node u is to determine which component-specifications are achievable by a component contained in T_u, the subtree rooted at u. More precisely, we will assign a table D_u to each node u such that each entry of D_u corresponds to a component-specification, and the existence of such an entry implies there exists a component in T_u that meets these specifications. Moreover, these specifications will allow us to compute the density of a component meeting these specifications. We say a component is *rooted at* u if u is the vertex in A_u closest to r (i.e. $u \in A_u$ and $A_u \subseteq T_u$). As done above, we will signify that a component A is rooted at u by placing u in the subscript of A.

The table D_u will have three dimensions, indexed by the weight, probability of non-ignition, and boundary cost of components rooted at u. So, a non-`null` entry $D_u\big[w' \,\big|\, q' \,\big|\, c'\big]$ signifies that there exists a component A_u such that $w(A_u) = w'$, $q(A) = q'$, and $c(\delta(A_u)) = c'$. The value of each table entry is the *minimum* β-value of a component that has the corresponding values. That is, if $D_u\big[w' \,\big|\, q' \,\big|\, c'\big] = \beta'$, then out of all components A_u with $w(A_u) = w'$, $q(A_u) = q'$, and $c(\delta(A_u)) = c'$, the one with minimum β-value has $\beta(A_u) = \beta'$. We call a set of components *comparable* if they are all rooted at u and have the same weight, non-ignition probability, and boundary cost.

The algorithm starts by assigning to each D_u the component-specification corresponding to when u is the single vertex of the component. So, for all $u \in V$, we set

$$D_u\big[w_u \,\big|\, q_u \,\big|\, c(\delta(u))\big] = \beta_u.$$

We can then build all other valid components by merging these entries (starting with the leaves), described in the following way. Notice that for $s \in V$, any component A_s can be viewed as the composition of s and a set of components rooted at the children $u_1, \ldots, u_k$ of s, say $A_{u_1}, \ldots, A_{u_k}$. Given that we have populated the entries of $D_{u_1}, \ldots, D_{u_k}$

properly, such a component can be "created" by considering $u_1, \ldots, u_k$ in some order, and "merging" the appropriate entries of D_{u_i} and D_s. In the algorithm, when considering u_i, each entry $D_{u_i}\big[w' \;\big|\; q' \;\big|\; c'\big] = \beta'$, corresponding to say component B_{u_i}, is "merged" with each entry $D_s[w'' \;\big|\; q'' \;\big|\; c''] = \beta''$, corresponding to say a component B_s. The new entry is $D_s\big[w' + w'' \;\big|\; q' \cdot q'' \;\big|\; c' + c'' - 2c_{su_i}\big] = \beta' + \beta''$, which corresponds to the component $B_s \cup B_{v_i}$; see Algorithm 2.1 for more detail.

For the above example with s and children $u_1, \ldots, u_k$, suppose we consider $u_1, \ldots, u_k$ in the order $1, \ldots, k$. After considering u_1, we have an entry in D_s corresponding to $A_{u_1} \cup \{s\}$ (or something comparable with better β-value); when we consider u_2, this will create an entry in D_s for $A_{u_2} \cup A_{u_1} \cup \{u\}$, and so on. Claim 2.5.3 makes this argument rigorous. The complete algorithm is presented as Algorithm 2.1.

Lemma 2.5.2. *Algorithm 2.1 returns the value of maximum $\rho(A)$ out of all components $A \in \mathcal{C}$. That is, the algorithm returns*

$$\max_{A \in \mathcal{C}} \frac{\Pi(A) - \beta(A)}{c(\delta(A))/2}. \tag{2.21}$$

Furthermore, the running time of the algorithm is $O(nM^2)$, where M is the maximum number of entries in a D_u table.

Proof. We prove this by showing through induction that for each $u \in V$, there exists an entry D_u for each component of T rooted at u that has minimum β-value out of all comparable components rooted at u. This is stated as the following claim.

Claim 2.5.3. *Let A_u be a component of T rooted at u and let $w' = w(A_u)$, $q' = q(A_u)$, and $c' = c(\delta(A_u))$. If*

$$A_u = \operatorname*{arg\,min}_{\substack{A \text{ rooted at } u: \\ w(A)=w',\, q(A)=q', \\ c(\delta(A))=c'}} \beta(A),$$

then Algorithm 2.1 assigns the entry $D_u\big[w'\big|q'\big|c'\big] = \beta(A_u)$ to the table.

Proof. For the base case, this claim is clearly true for the leaves of the tree, as the only component rooted at u of a leaf u is the node u itself. Thus for each leaf u, the table D_u will have only one entry (which is assigned at line 3 of the algorithm). Now, pick some non-leaf node s, and assume our claim is true for all nodes below s. Let A_s be a component rooted at s that has minimum β-value out of all comparable components. Note that A_s is a component made up of the node s as well as a set of components, each of which is rooted at a child node. Let such children be $u_1, \ldots, u_j$ and the corresponding components be $A_{u_1}, \ldots, A_{u_j}$. The values of A_s can clearly be computed via the values of these components as follows.

1. $w(A_s) = w_s + w(A_{u_1}) + \cdots + w(A_{u_j})$

Algorithm 2.1

Input: $T = (V, E), w, c, q, \beta$

1: Pick an arbitrary $r \in V$ as the root of the tree.
2: **for each** $u \in V$, processed in descending order of distance from r, **do**
3: $\quad D_u\big[w_u \mid q_u \mid c(\delta(u))\big] \leftarrow \beta_u$.
4: $\quad$ **for each** child v of u **do**
5: $\quad\quad$ **for each** non-`null` $D_v\big[w(A_v) \mid q(A_v) \mid c(\delta(A_v))\big]$ **do**
6: $\quad\quad\quad$ **for each** non-`null** $D_u\big[w(A_u) \mid q(A_u) \mid c(\delta(A_u))\big]$ **do**
7: $\quad\quad\quad\quad w' \leftarrow w(A_u) + w(A_v)$
8: $\quad\quad\quad\quad q' \leftarrow q(A_u) \cdot q(A_v)$
9: $\quad\quad\quad\quad c' \leftarrow c(\delta(A_u) + c(\delta(A_v)) - 2c_{uv}$
10: $\quad\quad\quad\quad$ **if** $D_u\big[w' \mid q' \mid c'\big] =$ `null` **then**
11: $\quad\quad\quad\quad\quad D_u\big[w' \mid q' \mid c'\big] \leftarrow \beta(A_v) + \beta(A_u)$
12: $\quad\quad\quad\quad$ **else**
13: $\quad\quad\quad\quad\quad D_u\big[w' \mid q' \mid c'\big] \leftarrow \min\big\{D_u\big[w \mid q' \mid c'\big],\ \beta(A_v) + \beta(A_u)\big\}$
14: $\quad\quad\quad\quad$ **end if**
15: $\quad\quad\quad$ **end for**
16: $\quad\quad$ **end for**
17: $\quad$ **end for**
18: **end for**
19: Searching through all $u \in V$, find the non-`null` entry

$$D_u\big[w(A') \mid q(A') \mid c(\delta(A')\big] = \beta(A')$$

that maximizes

$$\frac{q(A')w(A') - \beta(A')}{c(\delta(A'))/2} \tag{2.20}$$

such that $c(\delta(A')) \leq B$.

Output: The maximum value of (2.20) obtained in step 19.

2. $q(A_s) = q_s q(A_{u_1}) \dots q(A_{u_j})$

3. $c(\delta(A_s)) = c(\delta(s)) + c(\delta(A_{u_1})) + \dots + c(\delta(A_{u_j})) - 2\sum_{i=1}^{j} c_{su_i}$

4. $\beta(A_s) = \beta_s + \beta(A_{u_1}) + \dots + \beta(A_{u_j})$

So, to prove our claim, we will show that there exists a table entry in D_s with the above values.

We know that for each $i = 1, \dots, j$, there exists the table entry

$$D_{u_i}\big[\, w(A_{u_i}) \,\big|\, q(A_{u_i}) \,\big|\, c(\delta(A_{u_i}))\,\big] = \beta(A_{u_i})$$

because of our induction hypothesis. We know that A_{u_j} exists, and if a comparable component to A_{u_j} existed with lesser β-value, then component A_s would have had such a component as a subcomponent instead of A_{u_j} (because per our assumption A_s has minimum β-value). From line 3 we know that the table entry $D_s[w_s, q_s, c(\delta(s))] = \beta_s$ also exists.

Now, consider the stage of the algorithm when s is the current node (line 2) and we are processing each child of s (line 4), and assume we process the children of s in order $u_1, \dots, u_j$. Note that there could be other children of s whose subtrees do not intersect with A_s. These nodes will not affect the outcome, so without loss of generality assume they are all processed after u_j. The iterations of the loop will proceed as follows. In this first iteration, when u_1 is the current child, the entry corresponding to A_{u_1} will be found at line 5, and the entry $D_s\big[\, w_s \,\big|\, q_s \,\big|\, c(\delta(s))] = \beta_s$ will be found at line 6. In lines 10–13 the following entry will be created:

$$D_s\big[\, w_s + w(A_{u_1}) \;\big|\; q_s \cdot q(A_{u_1}) \;\big|\; c(\delta(s)) + c(\delta(A_{u_1})) - 2c_{su_1} \,\big] = \beta_s + \beta(A_{u_1}) \tag{2.22}$$

If $j = 1$, we are clearly done. Otherwise, during the second iteration, when u_2 is the current child, the entry corresponding A_{u_2} will be found at line 5, and the entry corresponding to $A_s \cap T_{u_1}$ (the entry defined in (2.22)) will be found at line 6. In lines 10–13 the following entry will be created:

$$D_s\left[w_s + \sum_{i=1}^{2} w(A_{u_i}) \;\middle|\; q_s \cdot \prod_{i=1}^{2} q(A_{u_i}) \;\middle|\; c(\delta(s)) + \sum_{i=1}^{2} c(\delta(A_{u_i})) - 2\sum_{i=1}^{2} c_{su_i} \right]$$

$$= \beta_s + \sum_{i=1}^{2} \beta(A_{u_i})$$

It is easy to see that this pattern repeats for the remaining $j - 1$ children, and that after the jth iteration, an entry corresponding to A_s will be in D_s, which proves the claim. ■

Thus, for each $u \in V$, there exists an entry D_u for each component rooted at u with minimum β-value out of all comparable components. The component of $\mathcal{C}$ that achieves the maximum density is such a component and therefore from line 19, we will find the maximum density value and return it. ■

Scaling Input Values

Now, our goal is to scale the input values for node weight, edge cost, and ignition probabilities so that the table D created by Algorithm 2.1 has size polynomial in the input size. In our final algorithm, we will scale the values *before* solving (P) and rounding the fractional solution, allowing us to solve the LP in polynomial-time; we will argue that a near-optimal solution to GPwIA with the scaled data remains a near-optimal solution to GPwIA with the original data.

For scaling the vertex weights, we first guess[1] a value W such that $\frac{1}{2}\text{OPT} \leq W \leq \text{OPT}$. Then, for a fixed $\epsilon > 0$, let $\mu = \epsilon W/n$, and for each $u \in V$ we scale w_u to $\hat{w}_u = \lfloor w_u/\mu \rfloor$. Similarly, for the edge costs, let $\nu = \epsilon B/n$ and set $\hat{c}_e = \lceil c_e/\nu \rceil$ and $\widehat{B} = \lfloor B/\nu + n \rfloor$ (recall that we may assume $c_e \leq B$). For scaling the probabilities of non-ignition q_u, first let $q_{\min}$ be the smallest non-zero non-ignition probability. We scale all $q_u \in (0,1)$ to $q_{\min}(1+\epsilon/n)^i$, where i is the largest integer such that $q_{\min}(1+\epsilon/n)^i$ is less that q_u. Essentially, the scaling partitions $[0,1]$ into the $\ell - 1$ intervals

$$\big[\, 0,\ q_{\min}(1+\epsilon/n)\,\big),\ \big[\, q_{\min}(1+\epsilon/n),\ q_{\min}(1+\epsilon/n)^2\,\big),\ \ldots\ ,\big[\, q_{\min}(1+\epsilon/n)^\ell,\ 1\,\big).$$

Then, for all $q_u \in (0,1)$, if $q_u \in \big[q_{\min}(1+\epsilon/n)^i,\ q_{\min}(1+\epsilon/n)^{i+1}\big)$, the new $\hat{q}_u$ is set to $q_{\min}(1+\epsilon/n)^i$. If $q_u \in \{0,1\}$, we leave it as is.

Lemma 2.5.4. *Algorithm 2.1 can be run in time polynomial of the input size when the values are scaled as described above. That is, the running time of the algorithm is $O(nM^2)$, and when the values are scaled, M is polynomial in the input size.*

Proof. In order to show that the algorithm runs in time polynomial of the scaled input, it is enough to bound the size of the table D_u. Note that when the values are unscaled, the maximum node value stored in the table is $\sum_{v\in V} w_v$. Recall that $W \leq \text{OPT}$. Clearly OPT is bounded above by $\sum_{u\in V} w_u$, and so the largest scaled value stored is:

$$\sum_{u\in V} \hat{w}_u \leq \sum_{v\in V} \frac{w_u}{\epsilon W/n} \leq \sum_{v\in V} n/\epsilon = O(n^2/\epsilon).$$

[1]We can "guess" the correct value of W as follows. For each power of 2 not exceeding $\sum_{u\in V} w_u$, we set W to this power, scale the input values, and run Algorithm 2.1. Lemma 2.5.4 tells us that for the correct W, the algorithm runs in polynomial-time. For arbitrary W, this may not be the case, but we can detect if this running time is exceeded, and if so, stop and return a `null` solution. Then, running the algorithm for all W still has polynomial running time, and we can return the best non-`null` solution.

Similarly, the largest boundary cost stored in the table is $\sum_{e\in E} c_e$, and so the largest scaled edge cost stored is $\sum_{e\in E} \hat{c}_e \leq \sum_{e\in E}(\frac{c_e}{\epsilon B/n}+1) = O(n^2/\epsilon)$.

For the scaled probabilities of non-ignition, recall that each $q_u \in (0,1)$ is set to $q_{\min}(1+\epsilon/n)^i$ for some positive integer i, and since we are multiplying these values, the only values that are stored in the tables are of the form $q_{\min}^j(1+\epsilon/n)^i$ where $1 \leq j \leq n$. This means that the dimension of the table can be indexed by these powers i and j. Therefore, in order to show that this dimension of the table is polynomial in the input size, it is sufficient to show that the largest power i is bounded by a polynomial of the input size. Let ℓ be such that $q_{\min}(1+\epsilon/n)^\ell < 1 \leq q_{\min}(1+\epsilon/n)^{\ell+1}$. It is easy to see that this implies $\ell = O(n/\epsilon \cdot \log(1/q_{\min}))$ which is clearly polynomial in the input size.

Thus, the maximum size of a table is $M = O(n^6/\epsilon^3 \log(1/q_{\min}))$, and the lemma follows. ■

Lemma 2.5.5. *Let $\widehat{\text{OPT}}$ be the optimal expected protected value of the problem when the data is scaled as described above, and assume we have an algorithm for the scaled problem such that the solution returned has expected value at least $\eta\widehat{\text{OPT}}$ and cost at most $\zeta\widehat{B}$. Then, the same solution with unscaled values has expected protected value at least $(1-2\epsilon)\eta\text{OPT}$ and cost at most $(1+\epsilon)\zeta B$.*

Proof. Let $\hat{\mathcal{O}}$ be an optimal set of components with the scaled values, and let $\mathcal{O}^*$ be an optimal set of components when the values are *not* scaled (so the expected value protected with $\mathcal{O}^*$ is OPT). First note that setting $\hat{w}_u = \lfloor w_u/\mu \rfloor$ implies that $\mu\hat{w}_u \leq w_u \leq \mu(\hat{w}_u+1)$ and thus $\mu\hat{w}_u \geq w_u - \mu$. Likewise, for all $u \in V$, $q_u \geq \hat{q}_u \geq q_u/(1+\epsilon/n)$. For a component A, these scaled values together imply:

$$\mathit{\Pi}(A) = \prod_{v\in A} q_v \sum_{v\in A} w_v \;\geq\; \prod_{v\in A} \hat{q}_v \sum_{v\in A} w_v \;\geq\; \mu \prod_{v\in A} \hat{q}_v \sum_{v\in A} \hat{w}_v. \tag{2.23}$$

Assuming we have an algorithm for the scaled problem such that the solution returned has value at least $\eta\widehat{\text{OPT}}$, (2.7.3) implies that the unscaled value of the solution is at least $\eta\mu\widehat{\text{OPT}}$. Also note that for any component A,

$$\prod_{u\in A} \hat{q}_u \;\geq\; \prod_{u\in A} \tfrac{1}{1+\epsilon/n} \cdot q_u \;\geq\; \tfrac{1}{(1+\epsilon/n)^n} \prod_{u\in A} q_u \;\geq\; (1-\epsilon) \prod_{u\in A} q_u. \tag{2.24}$$

Let Y^* be the set of boundary edges of the optimal solution, that is $Y^* = \bigcup_{A\in\mathcal{O}^*} \delta(A)$. Then, because $\widehat{B} = \lfloor B/\nu + n \rfloor$, we have $B \leq \nu(\widehat{B} - n + 1)$, and setting $\hat{c}_e = \lceil c_e/\nu \rceil$ implies that $\hat{c}_e \leq c_e/\nu + 1$. Together these imply that $\mathcal{O}^*$ is a feasible solution with the scaled

costs and scaled budget $\widehat{B}$, as shown below:

$$\begin{aligned}\sum_{e\in Y^*}\hat{c}_e &\le \sum_{e\in Y^*}\left(\frac{c_e}{\nu}+1\right)\\ &\le \frac{1}{\nu}\sum_{e\in Y^*}c_e+(n-1)\\ &\le \frac{1}{\nu}B+(n-1)\\ &\le \frac{1}{\nu}\left(\nu\left(\widehat{B}-n+1\right)\right)+(n-1)=\widehat{B}.\end{aligned}$$

Then,

$$\begin{aligned}\mu\cdot\widehat{\mathrm{OPT}} &= \sum_{A\in\hat{\mathcal{O}}}\prod_{u\in A}\hat{q}_u\sum_{u\in A}\mu\hat{w}_u\\ &\ge \sum_{A^*\in\mathcal{O}^*}\prod_{u\in A^*}\hat{q}_u\sum_{u\in A^*}\mu\hat{w}_u \qquad (2.25)\\ &\ge (1-\epsilon)\sum_{A^*\in\mathcal{O}^*}\prod_{u\in A^*}q_u\sum_{u\in A^*}(w_u-\mu)\\ &\ge (1-\epsilon)\left(\sum_{A^*\in\mathcal{O}^*}\prod_{u\in A^*}q_u\sum_{u\in A^*}w_u-\sum_{A^*\in\mathcal{O}^*}\sum_{u\in A^*}\mu\right)\\ &= (1-\epsilon)\left(\mathrm{OPT}-n\mu\right)\\ &= (1-\epsilon)\left(\mathrm{OPT}-\epsilon W\right)\\ &\ge (1-\epsilon)\left(\mathrm{OPT}-\epsilon\mathrm{OPT}\right)\\ &\ge (1-2\epsilon)\mathrm{OPT}, \qquad (2.26)\end{aligned}$$

where line (2.25) is a result of $\mathcal{O}^*$ being a feasible solution with the scaled costs and budget.

Now, for the cost of the solution, we know that $c_e \le \hat{c}_e\nu$ and $\widehat{B}\le B/\nu+n$. Then, if $\widehat{Y}$ is the set of boundary edges of the optimal scaled solution, i.e. $\widehat{Y}=\bigcup_{A\in\hat{\mathcal{O}}}\delta(A)$, the cost of the unscaled solution is:

$$\sum_{e\in\widehat{Y}}c_e\le\nu\sum_{e\in\widehat{Y}}\hat{c}_e\le\nu\left(\zeta\widehat{B}\right)\le\zeta\nu(B/\nu+n)=\zeta(1+\epsilon)B. \qquad (2.27)$$

This completes proof. ■

2.5.2 More General Distributions

Before describing our approximation algorithms, we note that there exist slightly more general probability distributions that allow for a maximum-density approximation via a

similar dynamic program as the one above. This distribution, introduced by Hayrapetyan, Swamy, and Tardos [25], is defined by a *distribution tree*, a rooted tree whose leaves are the nodes of a given graph G, such that each edge e of the tree is labelled with $p_e \in [0, 1]$. A distribution is defined on the nodes of G by having each edge "activated" independently with probability p_e; this generates a scenario consisting of the nodes that are reachable by the root of the tree using only activated edges. Note that GPwIA is the case where the distribution tree is a star. Although this type of distribution is out of scope of this work, we note that a similar DP to Algorithm 2.1 is possible, where the running time is exponential in the depth of tree; this is work in progress.

2.6 Putting everything together: a bicriteria approximation for GPwIA on trees

We now present an approximation algorithm for GPwIA on trees that meets the claims of Theorem 2.1.1. Given an instance of GPwIA, the main parts of the algorithm are as follows. First, choose $\epsilon > 0$, and scale the given node values, edge costs, and non-ignition probabilities as described in Section 2.5.1. Then, using the ellipsoid method (Theorem 2.5.1) with Algorithm 2.1 as the separation oracle, solve the dual linear program (D). This produces a linear program with a polynomial number of constraints, whose dual is a linear program equivalent to (P) with a polynomial number of variables. Solving this LP yields a solution x^* with value $\widehat{\text{OPT}}$. We then round as described in Section 2.4 to obtain a (random) integer solution.

Assuming we have chosen suitable constants as described in Theorem 2.4.3 such that $\phi_1, \phi_2 > 0$, we can boost this probability by repeating the experiment multiple times. Then, this theorem combined with Lemma 2.5.5 implies our final solution with original values and costs will have expected protected value at least $(1-2\epsilon)(1-\epsilon_1)\sigma\phi_2/(1+\sigma) \cdot \text{OPT}$ and expected edge cost at most $(1+\epsilon)\kappa B$.

Now, we apply Theorem 2.4.3 and we give values for the constants which prove the claims of Theorem 2.1.1. If we are allowed to exceed the budget by a factor of 4, setting $\epsilon_1 = 1/2$, $\kappa = 4/(1+\epsilon)$, $\sigma = 1/(2-4\epsilon)$, and $\gamma = 1/10$ (with sufficiently small ϵ) yields $\phi_1 \approx 0.2147$ and $\phi_2 \approx 0.0969$, resulting in a $(\phi_2/6, 4)$-approximation algorithm. If instead we only want a factor of 2 above the budget, then having $\epsilon_1 = 1/2$, $\kappa = 2/(1+\epsilon)$, $\sigma = 1/(4-8\epsilon)$, and $\gamma = 1/30$ results in $\phi_1 \approx 0.3583$ and $\phi_2 \approx 0.0959$ and a $(\phi_2/10, 2)$-approximation algorithm; this proves Theorem 2.1.1.

2.7 Unicriterion Approximation for GPwIA on Trees

For the algorithm presented above, one may be tempted to contrive constants that result in an approximate solution that satisfies the budget. However, the initial scaling of edge

costs still prevents us from a unicriterion solution. To elaborate, recall that we scale the edge costs in order to ensure that the DP described in Section 2.5.1 runs in polynomial-time and hence, the component-based LP (P) can be solved in polynomial-time. Thus, our components of consideration are those with scaled boundary cost at most $\widehat{B}$. While we can ensure, by choosing constants suitably in the rounding process, that the rounded solution also has *scaled boundary cost at most* $\widehat{B}$, we cannot escape the fact that due to the edge-cost scaling, the actual boundary cost may be $(1+\epsilon)B$. We also note that it is not necessary to scale the edge costs before solving the LP; one can apply such scaling as part of the separation oracle of the ellipsoid method. However, this results in only an approximate separation oracle, meaning that potential solutions may pass feasibility without actually being within budget.

In this section, we explain how to convert this bicriteria approximation to an approximation algorithm that does not exceed the budget. The main idea behind this approach is that in an optimal solution there can only be a constant number of components with relatively large boundary cost, and because of this, we can search for such components separately. Then, we can apply the main algorithm to the components that have relatively low boundary cost. The key insight here is that we can extract a subset of the low-boundary cost components whose total value is close to the total value of all the low-boundary cost components, and whose total boundary cost is a fraction of B. Thus, by applying our bicriteria approximation algorithm, we now obtain a near-optimal solution that does not violate the budget.

Formally, consider an optimal solution of components $\mathcal{O}^*$. For some $\lambda \in (0,1)$, let $\mathcal{O}_1^*$ be the components of $\mathcal{O}^*$ with boundary cost at least λB, and let $\mathcal{O}_2^*$ be the components of $\mathcal{O}^*$ with boundary cost less than λB. Then because

$$\sum_{O_1 \in \mathcal{O}_1^*} \mathit{\Pi}(O_1) + \sum_{O_2 \in \mathcal{O}_2^*} \mathit{\Pi}(O_2) = \mathrm{OPT},$$

we know that either $\mathcal{O}_1^*$ or $\mathcal{O}_2^*$ will contribute at least $\frac{1}{2}\mathrm{OPT}$ to the optimal solution. Also, notice that $|\mathcal{O}_1^*| \leq \frac{2}{\lambda}$, which is a constant. Our goal will be to find an optimal solution when restricted to components with either relatively large or small boundary cost. Then, the better of those two solutions will be at least $\frac{1}{2}\mathrm{OPT}$.

Section 2.7.1 deals with the latter case, where the goal is to find an optimal set of components when we are restricted to only components with cost at most λB. We apply a slightly modified algorithm to the one above. In Section 2.7.3, we consider the problem of obtaining a maximum-value solution consisting of a constant number of components, and show that this problem admits a PTAS. Running both the PTAS and this new algorithm, and then choosing the better of the two solutions, results in an approximation algorithm for GPwIA that returns a solution of cost within budget. We discuss this in Section 2.7.2, which proves Theorem 2.1.2.

2.7.1 Approximation for Inexpensive Components

We now assume each component has boundary cost at most λB. This assumption allows us to modify the algorithm of Theorem 2.1.1 by restricting the variables of the linear program (P) to only such components, which can be achieved by modifying line 19 of the dual separation oracle Algorithm 2.1. This stronger bound on boundary cost has a number of advantages. First, we can guarantee that the components of the solution have unscaled boundary cost at most B. By limiting the search to components with scaled boundary cost at most $\lambda\widehat{B}$, we obtain components with unscaled boundary cost at most $(1+\epsilon)\lambda B$. So, if λ is at most $1/(1+\epsilon)$, each potential component will be within budget.

Another advantage is that since the boundary cost of each component is only a small fraction of the total boundary cost $2B$, limiting the budget by some fraction still results in a relatively good approximation. This idea is given below as Lemma 2.7.1, where $\mathrm{OPT}_2 = \sum_{A\in\mathcal{O}_2^*} \Pi(A)$.

Lemma 2.7.1. *There exists a subset of $\mathcal{O}_2^*$ with total boundary cost at most $2B/\alpha + \lambda B$, for $\alpha \geq 1$, such that the expected protected value of these components is at least OPT_2/α.*

Proof. We show this by interpreting $\mathcal{O}_2^*$ as an instance of Knapsack. Let $|\mathcal{O}_2^*| = k$, and let $\{A_1, \ldots, A_k\} = \mathcal{O}_2^*$ be the items of the knapsack, each item A_i having profit $p_i = \Pi(A_i)$ and cost $c_i = c(\delta(A_i))$, such that they are sorted by the ratio of their values to their costs, i.e. $p_1/c_1 \geq p_2/c_2 \geq \cdots \geq p_k/c_k$. Let the budget of this Knapsack problem be $2B/\alpha$.

Recall that an optimal fractional solution of Knapsack can be constructed by greedily picking, in the above order, the first ℓ items completely as well as a fraction of the $(\ell+1)$-th item, such that the total cost equals the budget. That is, we choose the first $A_1, \ldots, A_\ell$ items such that $\sum_{i=1}^{\ell} c_i \leq 2B/\alpha$, and then choose a fraction θ of item $A_{\ell+1}$ such that $\theta = (2B/\alpha - \sum_{i=1}^{l} c_i)/c_{\ell+1}$. Since $\sum_{i=1}^{k} p_i = \mathrm{OPT}_2$, it is easy to see that $\sum_{i=1}^{\ell} p_i + \theta p_{\ell+1} \geq \mathrm{OPT}_2/\alpha$. Since each component has boundary cost at most λB, this greedy solution with *all* of item $A_{\ell+1}$ has boundary cost at most $2B/\alpha + \lambda B$, and thus $\{A_1, \ldots, A_{\ell+1}\}$ meets the claims of the lemma. ■

Note that by selecting a subset of $\mathcal{O}_2$, it is not guaranteed that the total *edge* cost of this subset is half of the total *boundary* cost. We will discuss below that our goal is to produce a fractional solution with total edge cost at most $B/2$ (so total component cost at most B), and we therefore apply Lemma 2.7.1 with $\alpha = \frac{2}{1/2-\lambda}$ and obtain a solution with protected expected value at least $\frac{1-2\lambda}{4}\mathrm{OPT}_2$.

We now describe our algorithm for this problem, which is similar to that of Section 2.6. Given an instance of GPwIA, we first replace the budget B with $B/2$. Then, as before, we choose $\epsilon > 0$, and scale the given node values, edge costs, and non-ignition probabilities as described in Section 2.5.1. When using the ellipsoid method with Algorithm 2.1, we now only search for components with boundary cost at most $\lambda\widehat{B}$. Solving the compact LP

(P) yields a fractional solution x^*, and from Lemma 2.7.1, we know the objective value of this solution will be at least $\frac{1-2\lambda}{4}\widehat{\mathrm{OPT}}_2$ (where $\widehat{\mathrm{OPT}}_2$ is the fraction of the scaled optimal solution value where each component has boundary cost at most $\lambda\widehat{B}$). We then round x^* as described in Section 2.4 to obtain a (random) integer solution.

Now, we apply Theorem 2.4.3 and give values for the constants which yield a unicriterion approximation algorithm. Notice that because the components of this solution have boundary cost at most $B/2$, setting $\kappa = 2/(1+\epsilon)$ now gives an integral solution within budget. So, with small enough ϵ and λ, having $\epsilon_1 = 1/2$, $\kappa = 2/(1+\epsilon)$, $\sigma = 1/(4(1-2\epsilon)(1-2\lambda))$, and $\gamma = 1/30$ results in $\phi_1 \geq 0.3583$ and $\phi_2 \geq 0.0959$. Then, combining this with the above yields a $(\phi_2/40)$-approximation of the optimal set of inexpensive components. This result is summarized in the below theorem.

Theorem 2.7.2. *Given an instance of GPwIA on trees, let $\mathcal{O}^*$ be an optimal set of components. Let $\mathcal{O}_2^* \subseteq \mathcal{O}^*$ be the set of those components with boundary cost at most λB, for fixed $\lambda \in (0,1)$, and let $\sum_{A\in\mathcal{O}_2^*} \Pi(A) = \mathrm{OPT}_2$. Then, there exists a randomized algorithm for that succeeds with constant probability and conditioned on success, returns a solution with expected protected value at least $0.0023 \cdot \mathrm{OPT}_2$ and cost at most B.*

We remark that the analysis can be significantly improved by using Chernoff bounds to analyze the total boundary cost, due to the fact that each random variable representing a component's boundary cost is now bounded above by λB. As previously mentioned, this work is not concerned with finding the best constants.

2.7.2 Proof of Theorem 2.1.2

Now, consider the components $\mathcal{O}_1^*$, which recall are the components of $\mathcal{O}^*$ with boundary cost at least λB. Our goal is to find a set of components that has expected protected value close to $\mathrm{OPT}_1 = \sum_{A\in\mathcal{O}_1^*} \Pi(A)$. Because $|\mathcal{O}_1^*| \leq \frac{2}{\lambda}$, which is a constant, we can achieve this goal by applying the following theorem.

Theorem 2.7.3. *Given an instance of* GPwIA *and a constant k, there exists a PTAS for the problem of finding a maximum-value solution that uses at most k components and whose total cost is at most B.*

We defer the presentation and analysis of this algorithm to Section 2.7.3, and now show how combining this algorithm with the algorithm in the previous section results in a constant-factor unicriterion approximation algorithm for GPwIA.

Our algorithm works as follows: Given an instance of GPwIA, we choose a $\lambda \in (0,1)$, and run the algorithm of Theorem 2.7.3 with this instance, a fixed $\epsilon \in (0,1]$, and $k = \lfloor 2/\lambda \rfloor$. This produces a solution $\mathcal{A}_1$ with value at least $(1-\epsilon)\mathrm{OPT}_1$. We also run the algorithm of Theorem 2.7.2, and obtain a solution $\mathcal{A}_2$ with value at least $\phi_2/40 \cdot \mathrm{OPT}_2$,

where $\phi_2 \approx 0.0959$. We emphasize that both of these solutions have total cost that does not exceed the budget. Then, we return whichever solution has the larger objective value.

Note that because $\text{OPT}_1 + \text{OPT}_2 = \text{OPT}$, for each instance of GPwIA there exists an $\alpha \in [0, 1]$ such that $\text{OPT}_1 = (1 - \alpha)\text{OPT}$ and $\text{OPT}_2 = \alpha\text{OPT}$. Since we are returning the solution with larger objective value, the worst-case solution will be when

$$(1 - \alpha)(1 - \epsilon)\text{OPT} = \frac{\phi_2}{40}\alpha\text{OPT},$$

which is achieved when $\alpha = \frac{40(1-\epsilon)}{\phi_2+40(1-\epsilon)}$. Thus, this algorithm has an approximation factor of $\frac{\phi_2(1-\epsilon)}{\phi_2+40(1-\epsilon)} \geq 0.0023(1 - \epsilon)$, which proves Theorem 2.1.2.

2.7.3 PTAS for Constant Number of Expensive Components

We now prove Theorem 2.7.3, that is, we devise a PTAS for the problem of finding a maximum-value solution of GPwIA consisting of a constant number of components. We will use a dynamic programming algorithm to find such a set of components, taking advantage of the tree structure of the input graph. The dynamic program is similar to, and an extension of, the one presented in Section 2.5.1. The main idea is that components within the subtree T_u, $u \in V$, can be obtained by components within the subtrees of u's children nodes.

Like the dynamic program given as Algorithm 2.1, we pick a root $r \in V$ of the tree and process the vertices of T in a bottom-up manner starting from the leaves, so that we process the children of a node u before processing u. The subproblem concerning node u will consist of finding sets of at most k components in the subtree rooted at u. Each u will be assigned a table F_u that will store the specifications of all possible k-tuples of components from T_u; moreover, the specification for each k-tuple will contain enough information so that we can calculate the expected protected value from this k-tuple. That is, if $A_1, \ldots, A_k$ are a set of components in T_u, if there exists an entry for this set in F_u, this entry will contain the data needed to compute $\sum_{i=1}^{k} \Pi(A_i)$.

Each entry of F_u will be indexed by the following values. The first index will be an integer ℓ where $0 \leq \ell \leq k$, which represents the number of components this entry encodes. The three other entries will be indexed by the lists $\mathcal{W}$, $\mathcal{Q}$, and f, each of length ℓ. These three lists will store information for each component in a consistent way, so that the ith entry of $\mathcal{W}$, $\mathcal{Q}$, and f will correspond to the same component. The list $\mathcal{W}$ will contain the values of the components; the list $\mathcal{Q}$ will contain the probabilities of non-ignition of the components; and the list f will contain Boolean values indicating which, if any, component contains u.

Given a node u, a set of components $\mathcal{O}$ is *compatible* with the entry $F_u\big[\ell \;\big|\; \mathcal{W} \;\big|\; \mathcal{Q} \;\big|\; f\big]$ if $\mathcal{O}$ consists of exactly ℓ components whose values, non-ignition probabilities, and u-membership have a 1-to-1 correspondence with $\mathcal{W}$, $\mathcal{Q}$, and f, respectively. We say that

two sets of components are *comparable in* T_u or T_u*-comparable* if they are both compatible with the same F entry. Intuitively, sets of components are comparable if they contain the same number of components and protect the same expected value. However, we emphasize that this notion of comparability is dependent on u: if exactly one of the two sets contains a component containing u, these sets are never T_u-comparable.

Now, the information stored in each table entry will be the set of boundary edges of a $(\leq k)$-tuple of components in T_u that is compatible with the table entry and that has minimum total boundary cost among all comparable sets. For example, suppose we have the following entry:

$$F_u\Big[\ \ell \ \Big|\ \mathcal{W} = (w_1, \ldots, w_\ell) \ \Big|\ \mathcal{Q} = (q_1, \ldots, q_\ell) \ \Big|\ f = (1, 0, \ldots, 0)\Big] = E'.$$

This implies that in T_u there exists at least one set of $\ell \leq k$ components $A_1, \ldots, A_\ell$ with $u \in A_1$ such that $w(A_i) = w_i$ and $q(A_i) = q_i$, for all $i = 1, \ldots, \ell$. Furthermore, E' is the set of boundary edges with minimum cost out of all such sets of components comparable in T_u. Clearly, it is possible to compute the expected value protected by these components: $\sum_{i=1}^{\ell} \Pi(A_i) = \sum_{i=1}^{\ell} w_i q_i$.

We note that the table entries are not confined to only the components that have boundary cost at least λB. This is fine, as $\mathcal{O}_1^*$ (or a set of components comparable to $\mathcal{O}_1^*$ with smaller boundary cost) is a feasible entry as described above. The optimal set of components will protect an expected amount of value at least that of $\mathcal{O}_1^*$, regardless of their boundary cost.

The algorithm works by creating all valid table entries of T_u by combining the table entries of the children nodes of u. Essentially we will be creating new sets of components in T_u by merging smaller sets of components that are in the subtrees of T_u. To better understand this process, consider the following scenario. For a non-leaf node u, let v be a child of u. Suppose we have ℓ_1 components $\mathcal{U} = \{U_1, \ldots, U_{\ell_1}\}$ in T_u such that none of the components intersect with T_v. We also have of ℓ_2 components $\mathcal{V} = V_1, \ldots, V_{\ell_2}$ in T_v. Assume also that $\ell_1 + \ell_2 \leq k + 1$. Let $w_i^u = w(U_i)$, $q_i^u = q(U_i)$ for all $i = 1, \ldots, \ell_1$, and similarly define w_i^v, q_i^v for v.

We now give rules for how to "merge" the two sets $\mathcal{U}$ and $\mathcal{V}$ into a single set. There are a few cases to consider, depending on whether u and/or v are part of a component.

1. If u is not in a component of $\mathcal{U}$, or v is not in a component of $\mathcal{V}$, then if $\ell_1 + \ell_2 \leq k$, our new set will simply be $\mathcal{U} \cup \mathcal{V}$.

2. If both u and v are in components, assuming without loss of generality that $u \in U_1$ and $v \in V_1$, we will create the following two sets of components. One set will be the same as case 1, $\mathcal{U} \cup \mathcal{V}$, if $\ell_1 + \ell_2 \leq k$. The other will be similar, except U_1 and V_1 will be merged into a single component, i.e. the second set will be $\{U_1 \cup$

$V_1, U_2, \ldots, U_{\ell_1}, V_2, \ldots, V_{\ell_2}\}$. Note that in this case, even if $\ell_1 + \ell_2 = k + 1$, the resulting set of components is of cardinality k.

If we think of this situation as the creation of new entries for F_u by combining known entries of F_u and F_v, it is straightforward to update the new values. The dynamic programming algorithm for generating these values is given as Algorithm 2.2. In the algorithm, we assume that the indexing of sets adheres to the notion of comparability, i.e. if two sets of components are comparable in T_u, they will map to the same entry in F_u. Without loss of generality, we also assume that if a component contains the root of the subtree, that component's values are first in the corresponding lists.

Before analyzing Algorithm 2.2, we note a subtle scenario that may occur when the algorithm assigns a set of edges to an entry of the table. If an entry already corresponds to a set of components, the edges within this entry will be replaced only if there exists a comparable set with *smaller* boundary cost. So, if there exists multiple comparable sets with equal boundary cost, the table entry corresponding to these sets will contain the boundary edges of the set that was processed first. We say that a table entry *equivalently corresponds* to a set of components if the entry either corresponds to the set or corresponds to a comparable set with equal boundary cost.

Claim 2.7.4. *Let n_u be the number of nodes in T_u and $\mathcal{C}_u$ be the set of all components in T_u. Then, for each $u \in V$, and for each set of components $\mathcal{O} = \{A_1, \ldots, A_{\ell_u}\} \subseteq \mathcal{C}_u$, $\ell_u = 1, \ldots, \min\{n_u, k\}$, such that $\mathcal{O}$ has minimum boundary cost among all T_u-comparable sets of components, F_u will contain an entry that equivalently corresponds to $\mathcal{O}$.*

Proof. For the base case, this claim is clearly true for the leaves of the tree. If u is a leaf, then the only valid set of components in $\mathcal{C}_u$ is the single component consisting of the single vertex u. This component will have value w_u, probability of non-ignition q_u, and the single boundary edge $\delta(u)$. The entry corresponding to this data is assigned to F_u in line 4 of the algorithm.

For the induction hypothesis, given a node s, we assume that this claim is true for all children of s. Let $\mathcal{O} = \{A_1, \ldots, A_{\ell_s}\} \subseteq \mathcal{C}_s$, $1 \leq \ell_s \leq \min\{n_s, k\}$, be a set of components that has minimum boundary cost among all T_u-comparable sets of components. Our goal is to show that there exists an entry of F_s that equivalently corresponds to $\mathcal{O}$. There are two cases to consider: when s is in a component of $\mathcal{O}$, and when s is in no component of $\mathcal{O}$. We first assume the latter.

Because s is contained in no component of $\mathcal{O}$, all components will exist in subtrees of the children of s. Let $s_1, \ldots, s_j$ be the children of s that contain at least one of these components, and assume without loss of generality that the algorithm processes these children in the given order. Furthermore, let $\mathcal{O}_i$ be the set of components contained within T_{s_i}, that is $\mathcal{O}_i = \mathcal{O} \cap \mathcal{C}_{s_i}$. Note that for any s_i, there does not exist a set of components

Algorithm 2.2

Input: Instance of $\mathsf{GPwIA}(T=(V,E), w, c, q, B)$, $\lambda \in (0,1)$

$k \leftarrow \lfloor 2/\lambda \rfloor$
Pick an arbitrary $r \in V$ as the root of the tree.
for each $u \in V$, processed in descending order of distance from r, **do**
  $F_u\big[1 \mid (w_u) \mid (q_u) \mid (1)\big] \leftarrow \delta(u)$
  $F_u\big[0 \mid \emptyset \mid \emptyset \mid \emptyset\big] \leftarrow \emptyset$
  for each child v of u **do**
    for each non-`null` $F_v\big[k'' \mid (w_1^v, \ldots, w_{k''}^v) \mid (q_1^v, \ldots, q_{k''}^v) \mid f_v\big] = E_v$ **do**
      for each non-`null` $F_u\big[k' \mid (w_1^u, \ldots, w_{k'}^u) \mid (q_1^u, \ldots, q_{k'}^u) \mid f_u\big] = E_u$ **do**
        if $k' + k'' \le k$ **then**
          $\mathcal{W}_1 \leftarrow (w_1^u, \ldots, w_{k'}^u, w_1^v, \ldots, w_{k''}^v)$
          $\mathcal{Q}_1 \leftarrow (q_1^u, \ldots, q_{k'}^u, q_1^v, \ldots, q_{k''}^v)$
          $E_1 \leftarrow E_u \cup E_v$
          $f_1 \leftarrow (0, \ldots, 0)$
          if f_u contains a 1-entry **then**
            $f_1 \leftarrow (1, 0, \ldots, 0)$
          end if
          $Y' \leftarrow F_u\big[k' + k'' \mid \mathcal{W}_1 \mid \mathcal{Q}_1 \mid f_1\big]$
          if $Y' =$ `null` **or** $c(E_1) < c(Y')$ **then**
            $F_u\big[k' + k'' \mid \mathcal{W}_1 \mid \mathcal{Q}_1 \mid f_1\big] \leftarrow E_1$
          end if
        end if
        if $k' + k'' \le k + 1$ **and** both f_u and f_v contain a 1-entry **then**
          $\mathcal{W}_2 \leftarrow (w_1^u + w_1^v, w_2^u, \ldots, w_{k'}^u, w_2^v, \ldots, w_{k''}^v)$
          $\mathcal{Q}_2 \leftarrow (q_1^u \cdot q_1^v, q_2^u, \ldots, q_{k'}^u, q_2^v, \ldots, q_{k''}^v)$
          $E_2 \leftarrow (E_u \cup E_v) \setminus \{uv\}$
          $Y'' \leftarrow F_u\big[k' + k'' - 1 \mid \mathcal{W}_2 \mid \mathcal{Q}_2 \mid (1, 0, \ldots, 0)\big]$
          if $Y'' =$ `null` **or** $c(E_2) < c(Y'')$ **then**
            $F_u\big[k' + k'' - 1 \mid \mathcal{W}_2 \mid \mathcal{Q}_2 \mid (1, 0, \ldots, 0)\big] \leftarrow E_2$
          end if
        end if
      end for
    end for
  end for
end for
Searching through all $u \in V$, find the non-`null` table entry $F_u\big[\ \ell \mid (w_1, \ldots, w_\ell) \mid (q_1, \ldots, q_\ell) \mid f\big] = E'$, that maximizes $\sum_{i=1}^{\ell} w_i q_i$ subject to $c(E') \le B$, and let $A_1, \ldots, A_{|E'|+1}$ be the components defined by removing E'.

Output: $A_1, \ldots, A_{|E'|+1}$

comparable to $\mathcal{O}_i$ whose boundary cost is less than that of $\mathcal{O}_i$. Otherwise, this lower-boundary cost set together with the other $\mathcal{O}_j$'s would give a set of components comparable to $\mathcal{O}$ with boundary cost less than that of $\mathcal{O}$, which is a contradiction. Thus, by the induction hypothesis, there exists an entry in F_{s_i} that equivalently corresponds to $\mathcal{O}_i$.

Consider the stage of the algorithm when we are building F_s, i.e. s is the current node of the loop at line 3. We will discuss the j iterations of when the children of s are processed, i.e. at the loop of line 6. During the iteration when we consider s_1, the algorithm will eventually consider the entry I_1 of F_{s_1} that equivalently corresponds to $\mathcal{O}_1$, as well as the entry $F_s\big[0 \bigm| \emptyset \bigm| \emptyset \bigm| \emptyset\big] = \emptyset$ (at lines 7 and 8, respectively). Note that the latter entry exists via assignment to F_s on line 5. As there exists at most k components in $\mathcal{O}$, the number of components of I_1 are at most k, and so the algorithm passes the "if" of line 9. Since this entry's f_s-index is $\emptyset$, this iteration will keep the new f-index as the 0 vector, i.e. f_1 is set to (0) at line 13, and since this iteration fails the "if" statement of line 14, f_1 remains (0) (also note that it will fail the "if" of line 22). Then, the merging of these two entries (lines 10–12) simply results in a copy of I_1, and this copy will be assigned as an entry to F_s (at line 19).

If T_{s_1} is the only child subtree containing components of $\mathcal{O}$, we are clearly done. Otherwise, during the second iteration of the loop when s_2 is the current child, the algorithm will eventually consider the entry I_2 of F_{s_2} that equivalently corresponds to $\mathcal{O}_2$, as well as the copy of I_1 in F_s. The merging of these two entries results in a new entry $I_{1,2}$ that equivalently corresponds to $\mathcal{O}_1 \cup \mathcal{O}_2$. Note that per our definition of $\mathcal{O}$, if there already exists an entry in F_s corresponding to $\mathcal{O}_1 \cup \mathcal{O}_2$, the edges of this entry have cost at least the boundary cost of $\mathcal{O}_1 \cup \mathcal{O}_2$ (otherwise, the components corresponding to this entry, together with $\mathcal{O} \setminus (\mathcal{O}_1 \cup \mathcal{O}_2)$, form a set T_s-comparable to $\mathcal{O}$ with smaller boundary cost). Therefore, either $I_{1,2}$ is assigned to F_s or an equivalently corresponding entry remains in F_s, and thus at the end of the iteration, F_s will always contain an entry equivalently corresponding to $\mathcal{O}_1 \cup \mathcal{O}_2$. It is easy to see that this general pattern repeats for each of the $j-2$ remaining iterations, and thus at the end of the jth iteration, F_s will contain an entry that equivalently corresponds to $\mathcal{O} = \mathcal{O}_1 \cup \cdots \cup \mathcal{O}_j$.

Now, we consider the case when s is contained in a single component of $\mathcal{O}$, say A_1. We proceed similarly as before, defining the previous notation slightly differently in order to mitigate the fact that A_1 is no longer contained in a single child's subtree. Let $s_1, \ldots, s_j$ be the child nodes of s whose subtrees contain at least one component of $\mathcal{O}$ *or* intersect with A_1. Note that if T_{s_i} intersects with A_1, then $T_{s_i} \cap A_1$ is a component of $\mathcal{C}_{s_i}$ and s_i is contained in this component. Let $\mathcal{O}_i$ be the set of components of $\mathcal{O}$ that are contained in T_{s_i} plus the subcomponent of A_1 that intersects with T_{s_i} (if such a subcomponent exists). So, $\mathcal{O}_i = (\mathcal{O} \cap \mathcal{C}_{s_i}) \cup \{T_{s_i} \cap A_1\}$. As before, the induction hypothesis implies that for each i, there exist an entry in F_{s_i} equivalently corresponding to $\mathcal{O}_i$. Note that for non-empty $T_{s_i} \cap A_1$, the edge $s_i s$ is in the boundary of $\mathcal{O}_i$ but not in the boundary of $\mathcal{O}$. However, since this edge will also be in the boundary of any set T_{s_i}-comparable to $\mathcal{O}_i$, this edge has

no effect on the minimum comparable set.

Like before, consider the stage of the algorithm when we are building F_s and processing child node s_1. During this iteration, the algorithm eventually considers the entry I_1 of F_{s_1} that equivalently corresponds to $\mathcal{O}_1$, as well as the entry $F_s[1 \mid (w_s) \mid (q_s) \mid (1)] = \delta(s)$. Note that the latter entry (call it J) exists via assignment to F_s on line 4. The values of I_1 depend on whether T_{s_1} intersects with A_1; we consider these cases separately.

1. If $T_{s_1} \cap A_1 \neq \emptyset$, then because $|\mathcal{O}_1| \leq k$, this entry along with the J entry will correspond to at most $k+1$ sets, and so the first check of the "if" statement at line 22 goes through. Also, $s_1 \in T_{s_1} \cap A_1$ (a component of $\mathcal{O}_1$) and so the f-index of I_1 will contain a 1-entry. Since the f-index of J also contains a 1-entry, the second check of the "if" statement at line 22 goes through. Then, a new entry $I_{1,s}$ of F_s is created by merging I_1 and J, which clearly equivalently corresponds to a modified $\mathcal{O}_1$ where the component containing A_1 now also contains s.

2. If, on the other hand, $T_{s_1} \cap A_1 = \emptyset$, then $\mathcal{O}_1$ must contain at most $k-1$ components (otherwise, $\mathcal{O}$ would contain more than k components, a contradiction). Thus, we pass the "if" statement at line 9, and a new F_s entry $I'_{1,s}$, formed by merging I_1 and J, will be created at lines 10–12, and because J's f-index is (1), this new entry will also have an f-index containing a 1 (by line 15). Thus, $I'_{s,1}$ equivalently corresponds to $\{O_1\} \cup \{s\}$, as desired. Note that because $T_{s_1} \cap A_1 = \emptyset$, the f-index of I_1 will *not* contain a 1-entry, and thus will not pass the "if" statement of line 22.

A similar argument can be made for the remaining $j-1$ iterations, the result of which implies that F_s contains an entry that equivalently corresponds to $\mathcal{O}$, as desired. ■

Scaling Values

In order for Algorithm 2.2 to have running time polynomial in the input size, we scale the node values and non-ignition probabilities as we did in Section 2.5.1. Let OPT_1 be the optimal value of this problem. Clearly, $\mathrm{OPT}_1 \leq \sum_{u \in V} w_v$. We first guess a value W such that $W \leq \mathrm{OPT}_1$. Then, for some $\epsilon > 0$, let $\mu = \epsilon W/n$, and for each $u \in V$, let the new node weight be $\hat{w}_u = \lfloor w_u/\mu \rfloor$. For $q_u \in (0,1)$, we round down to $\hat{q}_u = q_{\min}(1+\epsilon/n)^i$, where i is the largest integer such that $q_u \geq \hat{q}_u$, and keep $\hat{q}_u = q_u$ the same if $q_u \in \{0,1\}$.

Lemma 2.7.5. *Algorithm 2.2 runs in polynomial time with the scaled values as described above.*

Proof. Consider the table entry $F_u[k \mid \mathcal{W} \mid \mathcal{Q} \mid f]$ for $u \in V$. Let $\hat{w} = \sum_{u \in V} \hat{w}_v$ be the sum of all node values. Clearly, the maximum value of an entry in $\mathcal{W}$ is $\hat{w}$, and thus the maximum size of the table in the $\mathcal{W}$ dimension is at most $\hat{w}^k$. Then,

$$\hat{w}^k \leq \left(\sum_{v \in V} \frac{w_u}{\epsilon W/n} \right)^k \leq \left(\sum_{v \in V} n/\epsilon \right)^k = O(n^{2k}/\epsilon).$$

For the $\mathcal{Q}$ index, recall from the proof of Lemma 2.5.4 that since each $\hat{q}(A)$ is of the form $q_{\min}^{j}(1+\epsilon/n)^{i}$, we can keep track of these values by indexing each entry of $\mathcal{Q}$ by indices i and j. A single entry of $\mathcal{Q}$ was shown to have a polynomial maximum size, and thus the maximum size of $\mathcal{Q}$ (the kth power of this polynomial) is also polynomial. Then, since the possible number of values of f is constant, we conclude that Algorithm 2.2 is a polynomial-time algorithm with these scaled values. ■

Lemma 2.7.6. *Let $\widehat{\mathrm{OPT}_1}$ be the optimal value of the problem when the initial values are scaled as described above. Assuming we have an algorithm that computes such an optimal solution $\widehat{\mathcal{O}}_1$, this solution with unscaled values has expected protected value at least $(1-2\epsilon)\mathrm{OPT}_1$.*

Proof. This proof is nearly identical to the proof of the similar result in Lemma 2.5.5. First, for any component A,

$$\Pi(A) \;\geq\; \mu \prod_{v\in A} \hat{q}_v \sum_{v\in A} \hat{w}_v.$$

Then,

$$\begin{aligned}
\mu \cdot \widehat{\mathrm{OPT}} &= \sum_{A\in\widehat{\mathcal{O}}_1} \prod_{u\in A} \hat{q}_u \sum_{u\in A} \mu\hat{w}_u \\
&\geq \sum_{A^*\in\mathcal{O}_1^*} \prod_{u\in A^*} \hat{q}_u \sum_{u\in A^*} \mu\hat{w}_u \\
&\geq (1-\epsilon)\left(\sum_{A^*\in\mathcal{O}_1^*} \prod_{u\in A^*} q_u \sum_{u\in A^*} w_u - \sum_{A^*\in\mathcal{O}_1^*}\sum_{u\in A^*} \mu\right) \\
&\geq (1-\epsilon)\left(\mathrm{OPT}_1 - n\mu\right) \\
&= (1-\epsilon)\left(\mathrm{OPT}_1 - \epsilon W\right) \\
&\geq (1-2\epsilon)\mathrm{OPT}_1,
\end{aligned}$$

which completes the proof. ■

Lemmas 2.7.5 and 2.7.6 together with Claim 2.7.4 proves Theorem 2.7.3.

2.8 Approximability of Polynomial Distribution

In this section, we consider the *Graph Protection with Polynomial Scenarios* (GPwPS) problem, where the ignition probabilities of scenarios are given as a distribution over subsets $I \subseteq V$ such that each I has a probability p_I of ignition. Currently, there is no known

approximation algorithm for this problem; our contribution is an inapproximability result that partly resolves this lack of knowledge. Namely, we show that even in the simple case when the input graph is a tree and each scenario has exactly two vertices, the problem reduces to Densest k-Subgraph, a problem notorious for its evasiveness to approximation.

In the version of Densest k-Subgraph (DkS) we consider, we are given a graph G and integer k, and the goal is to find a subgraph of G with at most (or exactly) k vertices that maximizes the number of edges. This problem is clearly NP-hard, as it generalizes the Maximum Clique problem. It is an important problem not only in applied settings (e.g. finding communities within social networks), but also in the study of approximation algorithms, as there exists a large gap between the current best known lower and upper approximation bounds. The best known algorithm, due to Bhaskara et al. [7], achieves an approximation factor of $O(n^{1/4+\epsilon})$. This result was the first improvement over Feige et al.'s $O(n^{1/3-\epsilon})$-approximation [16] from nearly 10 years prior. For inapproximability, Khot [29] showed that assuming a standard complexity assumption (that NP does not have subexponential-time randomized algorithms) there does not exist a PTAS for the problem. More recently, Raghavendra and Steurer's Small Set Expansion Conjecture was shown to imply NP-hardness of approximating DkS to any constant factor [36, 37], and Bhaskara et al. [8] provided evidence suggesting DkS may be even harder to approximate than Small Set Expansion or Unique Games.

We now prove Theorem 2.1.3, which states that an α-approximation algorithm for GPwPS on trees implies an α-approximation algorithm Densest k-Subgraph.

Proof of Theorem 2.1.3. We prove this by first showing that DkS reduces to GPwPS. Let $G = (V, E)$ be an instance of DkS. We convert this to an instance of GPwPS as follows. The vertices of the tree V' are defined as one new vertex s as well as all vertices V, so $V' = \{s\} \cup V$. For the edge set E', we connect s to each $u \in V$. So our instance $T = (V', E')$ is a star graph with center s and leaves V. For each $u \in V$, we set the node value $w_u = 0$ and the cost of each edge to $c_{su} = 1$. We set $w_s = 1$ and the budget to $B = k$. Finally, the ignition scenario subsets are defined by the edge set E, with each scenario having the same probability of ignition. That is, for each $uv \in E$, the subset $\{u, v\}$ is a scenario which occurs with probability $p_{\{u,v\}} = 1/|E|$.

Consider a feasible solution to this GPwPS instance. Clearly the only component that will contribute any value to the solution will be the one containing s. Let this component be A, and let S be the set of vertices not in A. Then, the value of this solution is equal to $\Pi(A)$, and

$$\begin{aligned}\Pi(A) &= w_s \cdot \Pr\big[\text{no scenario in } A \text{ ignites}\big] \\ &= 1 \cdot \Pr\big[\text{a scenario strictly contained in } S \text{ ignites}\big].\end{aligned}$$

An ignition scenario $\{u, v\}$ is strictly contained in S if both endpoints of the edge uv are

contained in S, i.e. if $u_1u_2 \in E(S)$. Thus,

$$\Pr\big[\text{a scenario strictly contained in } S \text{ ignites}\big] = \sum_{uv \in E(S)} p_{\{u,v\}} = \frac{|E(S)|}{|E|}.$$

Each removed edge of the GPwPS instance has cost 1, and removing an edge results in adding a node (the endpoint which is not s) to S. Thus, if the GPwPS solution is within budget k, the cardinality of S will be at most k. Therefore, feasible solutions of GPwPS correspond to feasible solutions of DkS, and (since $|E|$ is independent of S) their objective values are proportional to each other. It follows that an α-approximate solution to GPwPS yields an α-approximate solution to DkS, and vice versa. ■

We note that although this result helps explain why no approximation algorithm for GPwPS was found, we believe that the above reduction is interesting on its own merit. When one considers a problem on graphs, it is often the case that confining the input to trees simplifies the problem. In the case of GPwPS, however, even just allowing ignition scenarios of size 2 results in such a confinement having no real benefit, as the underlying structure encodes a significantly hard problem regardless.

We also note that the above inapproximability result, coupled with the fact that our LP-rounding algorithm works for any scenario distribution, implies that the same inapproximability result holds for solving the linear program (P), and thus also for finding a maximum-density component (as defined in Section 2.5). We can also infer the inapproximability of the maximum-density component problem from the reduction of Theorem 2.1.3. With the instance created in the proof of the theorem, we see that the maximum-density component problem is equivalent to finding a set S with $|S| \leq k$ that maximizes $|E(S)|/|S|$; this problem is known as Densest at-most-k-Subgraph, and Goldstein and Langberg [20] showed that an α-approximation algorithm for it yields a 4α-approximation for DkS.

Chapter 3

Conclusion and Future Work

In this thesis, we presented approximation results for the Graph Protection problem when ignition scenarios may contain arbitrary numbers of nodes. In particular, we devised a rounding algorithm that, when given a fractional solution of a novel component-based LP with value W, returns an integral solution with expected protected value $\Omega(W)$ and expected cost $O(B)$. We showed that a near-optimal fractional solution of the LP can be obtained for GPwIA on trees, and applying the rounding algorithm to such a solution yields the approximation algorithms of Theorems 2.1.1 and 2.1.2. We also showed that a similar approximation is unlikely to exist for the GPwPS problem, due to an approximation-preserving reduction from Densest k-Subgraph.

There are a number of directions for future work on Graph Protection. As previously noted, we did not focus on optimizing the constants of the approximation algorithms of Theorems 2.1.1 and 2.1.2. Thus, it is unknown what the best factors achievable with our methods are, and whether there exists different methods that produce better factors. Furthermore, many open problems remain for Graph Protection when the input consists of an *arbitrary* graph (and not exclusively a tree). Almost all results known for general graphs use the cut-based tree decomposition of Räcke [35], which extends results on trees to general graphs, but in doing so, violates the budget by an $O(\log n)$-factor. To the best of our knowledge, the only result that does not use this technique is that of Hayrapetyan et al. [24], although this is for the deterministic, single-source problem when we seek to *minimize* the number of *unprotected* nodes. Another direction for future work is to consider similar problems in the stochastic *2-stage recourse model* (instead of the stochastic single-stage model we consider). For example, even the development of an approximation algorithm for 2-stage GPwIA on trees is an interesting open problem.

Bibliography

[1] M. A. Adams. Mega-fires, tipping points and ecosystem services: Managing forests and woodlands in an uncertain future. *Forest Ecology and Management*, 294:250–261, 2013. 5

[2] J. K. Agee and C. N. Skinner. Basic principles of forest fuel reduction treatments. *Forest Ecology and Management*, 211:83–96, 2005. 5

[3] A. A. Ageev and M. Sviridenko. Pipage Rounding: A New Method of Constructing Algorithms with Proven Performance Guarantee. *Journal of Combinatorial Optimization*, 8(3):307–328, 2004. 8

[4] S. Agrawal, Y. Ding, A. Saberi, and Y. Ye. Price of Correlations in Stochastic Optimization. *Operations Research*, 60(1):150–162, 2012. 7, 11

[5] E. Anshelevich, D. Chakrabarty, A. Hate, and C. Swamy. Approximability of the firefighter problem: Computing cuts over time. *Algorithmica*, 62(1-2):520–536, 2012. 6

[6] N. Bansal and M. Sviridenko. The Santa Claus problem. In *Proceedings of the 38th ACM Symposium on Theory of Computing (STOC'06)*, pages 31–40, 2006. 11

[7] A. Bhaskara, M. Charikar, E. Chlamtac, U. Feige, and A. Vijayaraghavan. Detecting High Log-Densities: an $O(n^{1/4})$ Approximation for Densest k-Subgraph. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC'10)*, pages 201–210, 2010. 41

[8] A. Bhaskara, M. Charikar, A. Vijayaraghavan, V. Guruswami, and Y. Zhou. Polynomial Integrality Gaps for Strong SDP Relaxations of Densest k-subgraph. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'12)*, pages 388–405, 2012. 41

[9] J. R. Birge and F. V. Louveaux. *Introduction to Stochastic Programming*. Springer, 2nd edition, 2011. 6

[10] O. N. Bondareva. Some applications of linear programming to cooperative games. *Problemy Kybernetiki*, 10:119–139. 15

[11] S. Dobzinksi, N. Nisan, and M. Schapira. Approximation Algorithms for Combinatorial Auctions with Complement-Free Bidders. *Mathematics of Operations Research*, 35(1):1–13, 2010. 11

[12] P. Domingos and M. Richardson. Mining the Network Value of Customers. In *Proceedings of the Seventh ACM International Conference on Knowledge Discovery and Data Mining (KDD'01)*, pages 57–66, 2001. 6

[13] M. Dyer and L. Stougie. Computational complexity of stochastic programming problems. *Mathematical Programming*, 106(3):423–432, 2006. 8

[14] R. Engelberg, J. Könemann, S. Leonardi, and J. Naor. Cut problems in graphs with a budget constraint. *Journal of Discrete Algorithms*, 5(2):262–279, 2007. 4, 9

[15] U. Feige. On Maximizing Welfare when Utility Functions are Subadditive. *SIAM Journal of Computing*, 39:122–142, 2009. 11, 12, 15, 16, 17

[16] U. Feige, G. Kortsarz, and D. Peleg. The Dense k-Subgraph Problem. *Algorithmica*, 29:410–421, 2001. 41

[17] S. Finbow and G. MacGillivray. The Firefighter Problem: A survey of results, directions and questions. *Australasian Journal of Combinatorics*, 43:57–77, 2009. 6

[18] M. A. Finney. A Computational Method for Optimizing Fuel Treatment Locations. *International Journal of Wildland Fire*, 16:702–711, 2007. 5

[19] Z. Friggstad and C. Swamy. Approximation Algorithms for Regret-Bounded Vehicle Routing and Applications to Distance-Constrained Vehicle Routing. In *Proceedings of the 46th ACM Symposium on Theory of Computing (STOC'14)*, pages 744–753, 2014. 11

[20] D. Goldstein and M. Langberg. The Dense k Subgraph problem. *CoRR*, arXiv:0912.5327, 2010. 42

[21] C. P. Gomes. Computational Sustainability: Computational Methods for a Sustainable Environment, Economy, and Society. *The Bridge, the National Academy of Engineering*, 39(4):5–13. *Winter Issue on Frontiers of Engineering.* 5

[22] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981. 22

[23] A. Gupta, M. Pál, R. Ravi, and A. Sinha. Boosted sampling: Approximation algorithms for stochastic optimization. In *Proceedings of the 36th ACM Symposium on Theory of Computing (STOC'04)*, pages 417–426, 2004. 7

[24] A. Hayrapetyan, D. Kempe, M. Pál, and Z. Svitkina. Unbalanced Graph Cuts. In G. Brodal and S. Leonardi, editors, *13th Annual European Symposium on Algorithms (ESA)*, volume 3669 of *Lecture Notes in Computer Science*, pages 191–202. Springer, Berlin, 2005. 3, 8, 43

[25] A. Hayrapetyan, C. Swamy, and E. Tardos. Network design for information networks. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'05)*, pages 933–942, 2005. 30

[26] N. Immorlica, D. Karger, M. Minkoff, and V. S. Mirrokni. On the Costs and Benefits of Procrastination: Approximation Algorithms for Stochastic Combinatorial Optimization Problems. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'04)*, pages 691–700, 2004. 7, 11

[27] N. Karmarkar and R. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proceedings of the 23rd Annual IEEE Symposium on the Foundations of Computer Science (FOCS'82)*, pages 312–320, 1982. 11

[28] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the Ninth ACM International Conference on Knowledge Discovery and Data Mining (KDD'03)*, pages 137–146, New York, NY, 2003. 6

[29] S. Khot. Ruling Out PTAS for Graph Min-Bisection, Densest k-Subgraph and Bipartite Clique. In *Proceedings of the 44th Annual IEEE Symposium on the Foundations of Computer Science (FOCS'04)*, pages 136–145, 2004. 41

[30] S. S. Lloyd. On balanced sets and cores. *Naval Research Logistics Quarterly*, 14:453–460. 15

[31] R. H. Möhring, A. S. Schulz, and M. Uetz. Approximation in Stochastic Scheduling: The Power of LP-based Priority Policies. *Journal of the ACM*, 46(6):924–942, 1999. 7, 11

[32] A. Moilanen, K. A. Wilson, and H. Possingham, editors. *Spatial Conservation Prioritization: Quantitative Methods and Computational Tools.* Oxford University Press, 2009. 6

[33] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of the approximations for maximizing submodular set functions. *Mathematical Programming*, 14(1), 1978. 6

[34] S. J. Prichard, D. L. Peterson, and K. Jacobson. Fuel treatments reduce the severity of wildfire effects in dry mixed conifer forest, Washington, USA. *Canadian Journal of Forest Research*, 40:1615–1626, 2010. 5

[35] H. Räcke. Optimal Hierarchical Decompositions for Congestion Minimization in Networks. In *Proceedings of the 40th ACM Symposium on Theory of Computing (STOC'08)*, pages 255–264, 2008. 4, 9, 43

[36] P. Raghavendra and D. Steurer. Graph Expansion and the Unique Games Conjecture. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC'10)*, pages 755–764, 2010. 41

[37] P. Raghavendra, D. Steurer, and M. Tulsiani. Reductions Between Expansion Problems. In *Proceedings of the 2012 IEEE Conference on Computational Complexity (CCC'12)*, pages 64–73, 2012. 41

[38] M. Richardson and P. Domingos. Mining Knowledge-Sharing Sites for Viral Marketing. In *Proceedings of the Eighth ACM International Conference on Knowledge Discovery and Data Mining (KDD'02)*, pages 61–70, 2002. 6

[39] H. D. Safford, J. T. Stevens, K. Merriam, M. D. Meyer, and A. M. Latimer. Fuel treatment effectiveness in California yellow pine and mixed conifer forests. *Forest Ecology and Management*, 274:17–28, 2012. 5

[40] D. B. Shmoys and G. Spencer. Approximation Algorithms for Fragmenting a Graph Against a Stochastically-Located Threat. *Theory of Computing Systems*, 56(1):96–134, 2015. 1, 2, 3, 4, 5, 7, 8, 9, 11, 12

[41] D. B. Shmoys and C. Swamy. An Approximation Scheme for Stochastic Linear Programming and Its Application to Stochastic Integer Programs. *Journal of the ACM*, 53(6):978–1012, 2006. 8

[42] G. Spencer. Robust Cuts Over Time: Combatting the Spread of Invasive Species with Unreliable Biological Control. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, pages 377–383, 2012. 6

[43] C. Swamy and D. B. Shmoys. Algorithms Column: Approximation Algorithms for 2-Stage Stochastic Optimization Problems. *ACM SIGACT News*, 37(1):33–46, 2006. 7

[44] Y. Wei, D. Rideout, and A. Kirsch. An optimization model for locating fuel treatments across a landscape to reduce expected fire losses. *Canadian Journal of Forest Research*, 38(4):868–877, 2008. 5

[45] D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011. 3