Stability in Multi-Agent Environments and Approximation Algorithms for NP-Hard Graph Problems

Roe Engelberg
Stability in Multi-Agent Environments and Approximation Algorithms for NP-Hard Graph Problems

Research Thesis

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Roee Engelberg

Submitted to the Senate of the Technion — Israel Institute of Technology
Adar 5769 Haifa March 2009
The research thesis was done under the supervision of Prof. Seffi Naor in the Computer Science Department.

I wish to express my gratitude to Seffi Naor who was much more than an advisor. Seffi made my research journey a pleasant one, despite many difficulties. He welcomed the development of new ideas and motivated me even when things seemed to go nowhere. While I was away from home, Seffi’s help, support and friendship were priceless.

I thank Danny Raz. Officially, he was my advisor for one year, practically even longer. Danny showed me that while we should enjoy the beauty of the world of ideas, we must not forget the world of practicality, and that even if combining the worlds might seem hard, it is possible.

I would also like to thank the Computer Science and Engineering Department at the University of Washington, who had me as a visiting grad student for a wonderful year. Special thanks to Anna Karlin and Richard Ladner, without whom my visit would never have been possible, and who welcomed me warmly. Anna was also a great academic mentor, energized with excitement about new ideas that could spark my heart and imagination. It was my pleasure to be part of a reading group along with Anna Karlin, Seffi Naor, Ben Birnbaum and Allan Borodin. I thank all of them for making it the great experience it was, especially Allan with his charm and experienced insights.

I thank Microsoft Research Theory Group in Redmond, WA for their hospitality. My visits there were both pleasant and academically inspiring. I thank the Computer Science Department, Technion, for making my adventures possible and being my home for a significant part of my academic life. Special thanks to Eli Biham and Yardena Kolet for their great help.

It was my honor to work with my co-authors. Special thanks go to Michael Schapira who reminded me in the very last steps of this work why I started it in the first place. Michael makes research fascinating as it should be and more than anything else, Michael is a great friend.

Other great friends that were a significant part of my academic life and whom I thank deeply are Jonathan Laserson (Joni) and Ido Tal. Joni, as my office mate and my academic better half for all these years, has been the first person to listen to my unprocessed ideas. Much of the growth of my thoughts into actual results should be credited for Joni’s feedback. While Joni was away, Ido Tal assisted me with turning immature concepts into valid questions. Ido was also my algebra and \LaTeX{} guru, and served as my hands and mouth in the Technion while I was far away. My PhD experience would have never been the same without Yaniv Hamo, Guy Flysher and Ron Rubinstein. I thank them for making the hard times of the PhD studies enjoyable.

My ideas were just a note on a page without the feedback from long fruitful discussions with many researchers. I would like to thank everybody and specifically Noam Nisan, Amos Fiat, Ariel Orda, Yuval Rabani, Moshe Tenenholtz, Gabriel Scalosub, David Amzallag, Niv Buchbinder and Liane Lewin-Eytan. Last but definitely not least, I would like to thank my dear family and all my friends for their support along the years.

The generous financial support of the Technion is gratefully acknowledged.
# Contents

## Abstract

## Abbreviations and Notations

### 1 Introduction

1.1 Definitions

1.1.1 Game Theoretic Terms

1.1.2 Approximation Algorithms

1.2 Organization of this Dissertation

### 2 Equilibria in Online Games

2.1 Introduction

2.1.1 Online Games

2.1.2 Examples

2.1.3 Our Results

2.1.4 Organization

2.2 Preliminaries

2.2.1 Randomized Strategies

2.3 The Paging Game

2.3.1 Algorithm LRU

2.3.2 Algorithm FIFO

2.3.3 Algorithm SIM - an algorithm with a bounded competitive ratio

2.4 The Static List Accessing Game

2.4.1 Lower Bounds

2.4.2 Algorithm Move-to-Front

2.4.3 Algorithm TIMESTAMP

2.4.4 Algorithm BIT

2.5 Metrical Task Games

2.5.1 The Model

2.5.2 The Non-malleability Principle

2.5.3 The Optimistic Method

2.6 The Caching Game

---

Technion - Computer Science Department - Ph.D. Thesis PHD-2009-06 - 2009
2.7 The Online Generalized Steiner Game ........................................ 29
  2.7.1 Algorithm $BC'$.......................................................... 31
  2.7.2 Analysis of Algorithm $BC'$......................................... 31
2.8 Conclusions and Future Research .............................................. 32

3 Convergence to Equilibria in Interdomain Backup Routing .......... 34
  3.1 Introduction ................................................................. 34
    3.1.1 Is Backup Interdomain Routing Safe?.......................... 34
    3.1.2 Our contribution ..................................................... 35
    3.1.3 Policy Guidelines for Backup Routing ......................... 36
    3.1.4 Related Work ......................................................... 37
    3.1.5 Organization of the Chapter ...................................... 37
  3.2 The Model ................................................................. 37
    3.2.1 Illustrative Example and Some Background .................... 38
    3.2.2 A Stochastic Model of Interdomain Routing .................. 39
  3.3 Existence of Stable States in Backup Interdomain Routing ....... 41
  3.4 Backup Interdomain Routing is Safe ................................ 46
  3.5 Other Aspects of Our BGP Safety Result ............................ 51
    3.5.1 Four Merits .......................................................... 51
    3.5.2 Routing Anomalies .................................................. 52
  3.6 Discussion: Enabling More Expressive Routing Policies .......... 53
    3.6.1 Generalized Next Hop .............................................. 53
    3.6.2 Policy-Consistency ................................................ 55
    3.6.3 Set-Avoiding Policies ............................................ 55
    3.6.4 Shortest-Path Routing with Backup Connections ............. 56
  3.7 Conclusions and Future Research ..................................... 57

4 Cut Problems in Graphs with a Budget Constraint ..................... 58
  4.1 Introduction ............................................................... 58
    4.1.1 Our Results ......................................................... 59
    4.1.2 Related Work ....................................................... 60
    4.1.3 Organization ......................................................... 61
  4.2 Preliminaries ............................................................. 61
  4.3 The Budgeted Isolating Multiway Cut Problem ....................... 62
    4.3.1 Hardness Results .................................................. 63
    4.3.2 A Greedy Approximation Algorithm for BIMC ................ 64
    4.3.3 Approximation Algorithms for the Weighted BIMC Problem ... 65
  4.4 The Weighted Budgeted Separating Multiway Cut Problem .......... 68
    4.4.1 Hardness Results .................................................. 68
    4.4.2 Approximation Algorithms for Weighted BSMC in Trees ....... 71
    4.4.3 An Approximation Algorithm for Weighted BSMC on General Graphs ... 73
  4.5 The Budgeted Graph Disconnection Problem .......................... 74
4.6 Further Discussion ................................................................. 75
   4.6.1 The Budgeted Sparsest Cut Problem ............................... 75
   4.6.2 Linear Programming Formulation Revisited ..................... 75
   4.6.3 Node-Weighted BSMC ..................................................... 77

5 Improved Approximation Algorithms for the Spanning Star Forest Problem 80
   5.1 Introduction ................................................................. 80
       5.1.1 Our Results and Techniques ....................................... 81
   5.2 Preliminaries .............................................................. 82
   5.3 An LP-Based Algorithm .................................................. 83
       5.3.1 Application of ROUNding-AlG to Dominating Set ............... 85
   5.4 An Approximation Algorithm for the Unweighted Spanning Star Forest Problem ............... 86
   5.5 An Approximation Algorithm for the Node-Weighted Spanning Star Forest Problem ......... 90
   5.6 Hardness of Approximation ............................................... 90
   5.7 Future Research .......................................................... 94
List of Figures

3.1  BGP oscillations with backup routing (based on DISAGREE in [40, 39])  38
5.1  Illustration of Graph $G$ (left) and $G^*$ (right)  87
5.2  The approximation ratios for ORACLE-ALG and ROUNING-ALG  89
5.3  The approximation ratios for TRIVIAL-ALG and ROUNING-ALG  91
Abstract

In this thesis we study decision making under two different types of limitations.

In the first part of this work, we concentrate on games where several players, or agents, continually interact. The players are selfish and have their own goals, but no single player has full control over its performance as the outcome depends on the choices of all of the involved agents. The solution concept we study is the equilibrium of the game, which can be thought of as a status quo in which no player has an incentive to deviate from its current choices. Specifically, we study the following topics:

**Equilibria in Online Games:** Online games are non-cooperative games that model scenarios combining online decision making with interaction between non-cooperative agents. Roughly speaking, an online game captures systems in which independent agents serve requests that arrive in an online fashion in a common environment. The players have to choose as a strategy an online algorithm. After presenting our model, we focus on characterizing the equilibria in online games.

**Convergence to Equilibria in BGP Games:** The Border Gateway Protocol (BGP) [67] establishes routes between the Autonomous Systems (ASes) that make up today’s Internet. Since each AS has its own preferences as for the routing outcome, and at the same time the outcome depends on the choices made by all of the ASes, BGP is, in fact, a game. Informally, BGP requires ASes to continuously choose their most preferred routes (given their local preferences), out of the most recently announced routes learned from neighboring ASes. This route selection process can be viewed as a best response dynamic of the underlying game. The convergence of best response dynamics of BGP is referred to as BGP safety. BGP safety is of high importance since in the resulted equilibrium the routing outcome is stable. We study different types of AS’s preferences for which BGP safety is guaranteed. Our techniques rely on the substitution of the traditional worst-case analysis of BGP dynamics with a more realistic probabilistic approach.

In the second part of this work we study NP-hard optimization problems in graphs. It is commonly believed that the optimal solution for these problems is intractable and thus cannot be found efficiently. Instead, we look for the best solution that can be computed efficiently by designing approximation algorithms, while also considering the approximation hardness of these problems. Specifically, we study the approximability of the following graph problems:

**Cut Problems with a Budget Constraint:** A cut in a connected graph is a set of edges whose removal from the graph disconnects the vertex set. We study budgeted variants of classical cut problems: the Multiway Cut problem, the Multicut problem, and the k-Cut problem.
The Spanning Star Forest Problem: A star graph is a tree of diameter at most two. A star forest is a graph that consists of node-disjoint star graphs. In the spanning star forest problem, given an unweighted graph $G$, the objective is to find a star forest that contains all the vertices of $G$ and has the maximum number of edges. We also consider a node-weighted version of this problem.
Abbreviations and Notations

\[ G = (V, E) \] — A graph with \( V \) as the set of vertices and \( E \) as the set of edges

\( OPT \) — Either the optimal solution for the problem at hand or its value

\( s_{-i} \) — The vector resulted by omitting the \( i \)th element from the vector \( s \)
Chapter 1

Introduction

Computing challenges are at the heart of computer science. While dealing with computing tasks, different challenges are faced according to the constraints of the computing task at hand. In this dissertation we concentrate on two aspects of computing that introduce different constraints and hence different challenges.

The first part of this work focuses on non-cooperative multi agents environments. The common scenarios in such environment introduce lack of coordination to the computation process of the different agents. Game theoretic approaches suggest possible concepts through which one could understand the expected interaction between the agents. The main theme that we are looking at is scenarios in which the interaction between the agents is ongoing, or continual. The solution concept we are interested in is the equilibrium, which is, informally, a stable state in which no agent has an incentive to change its choices. Once the system reaches equilibrium, future interactions between the agents are expected to be implied directly by the current choices of the players, and hence is predictable and stable.

In the second part of this dissertation we consider hard graph problems in which the optimal solution cannot be computed efficiently unless $P = NP$ (which commonly believed to be unlikely), where efficient computing refers to computing in polynomial time. As resources, and specifically computing time, are limited in many applications in the real world, the challenge is to come up with a good solution, possibly not optimal, in polynomial time. To this end we study approximation algorithms, which guarantee good worst-case performance.

1.1 Definitions

In this section we formally define the key terms that will be used throughout this dissertation. The section is divided into two subsection, one subsection for each of the parts of our work.

1.1.1 Game Theoretic Terms

In a game, $N$ players interact by choosing their actions out of a possible set of strategies. Player $i$ picks a strategy from the set $S_i$, its strategy space. Let $s$ be the vector of strategies picked by the players, where $s_i \in S_i$. The vector $s$ induces the outcome of the game. It is possible also that the outcome will be influenced by an external agent (not one of the players), usually referred to as nature. Each player has a preference
function over the possible outcomes of the game, and we denote the preference relation of the $i$th player by $\geq_i$. A strategy $s_i \in S_i$ is a best response of player $i$ to $s_{-i}$ if for every $s' \in S_i$, $(s_{-i}, s_i) \geq_i (s_{-i}, s')$. Notice that the best response of a player need not to be unique. A vector of strategies, $s$, is an equilibrium of the game, if for every player $i$, it holds that $s_i$ is a best response of player $i$ to $s_{-i}$. Notice that once the system reaches an equilibrium, no agent has an incentive to deviate by changing its strategy. The resulted stability is of central importance and is a main source of motivation for the study of equilibria.

Reaching an equilibrium is not trivial. Different dynamics can be considered and the simplest one (and arguably the most natural one) is presented here: Given a strategy vector $s$, a best response of the players in $A$, where $A$ is a nonempty subset of players, results a strategy vector $s'$ in which $s'_i$ is a best response of player $i$ to $s_{-i}$ for a player $i \in A$ and $s'_i = s_i$ for $i \notin A$. Informally, the players in $A$ try to improve their performance greedily according to the current state while the choices of rest of the players are kept fixed. A best response dynamic is a sequence of strategy vectors $s_0, s_1, \ldots$ such that each vector $s^j$ (except $s^0$) is resulted by a best response of some players from the vector $s^{j-1}$. When the set of players $A$ that move in each step is of size one, the best response dynamic is usually easier to analyze, and hence we sometime focus on such dynamics. Notice that if a best response dynamic reaches an equilibrium, the dynamic converges and no further changes of strategies are expected.\footnote{1We usually assume that a player changes its strategy only if it strictly prefers a different strategy.}

1.1.2 Approximation Algorithms

In a combinatorial optimization problem, we are given an instance of the input for which a set of feasible outputs is defined. Each feasible output has a value (usually a non negative integer), and our goal is to compute a feasible solution with the optimal value among the possible feasible solutions. In a maximization problem the optimal value is the maximum value, while in a minimization problem the minimum value is optimal.

An NP-hard optimization problem is a problem for which the optimal solution cannot be found efficiently (in polynomial time) unless $P = NP$. For an NP-hard optimization problem we usually compromise for a nearly optimal solution, or an approximate solution. Algorithms that have some guarantee on the value of their solution with respect to the value of the optimal solution are called approximation algorithms. Specifically, given a maximization problem, we say that an algorithm is an $\alpha$-approximation for $0 < \alpha \leq 1$, if for every input instance the algorithm produces a solution whose objective value is at least $\alpha$ times that of the optimal solution for that instance.

Many optimization problems can be described by integer programs. In an integer problem we have a set of integer variables that are linearly constrained (by equalities or inequalities) and a target linear function. A feasible solution is a set of values for the variables that satisfy all of the constraints. An optimal solution is a feasible solution with the minimum/maximum value among all the feasible solutions. The problem of solving integer programs is NP-hard and hence integer programs are usually relaxed to linear programs (LPs). In an LP relaxation of an integer program the variables are not constrained to take integer values only and may take any real values. There are efficient algorithms for solving LPs (see for example, [53]). Computing the solution for an LP relaxation of a given integer program may shed light on good solutions for the original integer program. Specifically, notice that every solution for the integer program is a solution
for its LP relaxation. Hence the value of the optimal solution of the LP is at least as good as the value of the optimal solution of the original integer program. Indeed, LPs were found to be very useful in the design and analysis of approximation algorithms (see for example, [76]).

1.2 Organization of this Dissertation

In the first two chapter we study equilibria of different games. Chapter 2 introduces the model of online games, which capture scenarios in which independent agents serve requests that arrive in an online fashion in a common environment. After defining the formal framework for the study of our new model, we focus on characterizing the equilibria of online games. In Chapter 3 we consider best response dynamics of an interdomain backup routing game. We prove that under some policy guidelines, convergence to an equilibrium is guaranteed in a probabilistic model, and we discuss the implications of our results.

The next two chapters deal with different NP-hard optimization problems in graphs. In Chapter 4 we study budgeted variants of well studied cut problems. The problem of the spanning star forest problem is presented in Chapter 5 along with lower and upper bounds for its approximability.
Chapter 2

Equilibria in Online Games

In this chapter we introduce the model of online games, and lay the foundations for studying this model. Roughly speaking, an online game captures systems in which independent agents serve requests in a common environment. Since the agents are independent, it is unlikely that some central authority can enforce a policy or an algorithm (centralized or distributed) on them, and thus, the agents can be viewed as selfish players in a non-cooperative game. In this game, the players have to choose as a strategy an online algorithm according to which requests are served. To further facilitate the game theoretic approach, we suggest the measure of competitive analysis as the players’ decision criterion. As the expected result of non-cooperative games is an equilibrium, the question of finding the equilibria of a game is of central importance, and thus, it is the central issue we concentrate on in this chapter. We study some natural examples for online games; in order to obtain general insights and develop generic techniques, we present an abstract model for the study of online games generalizing metrical task systems. We suggest a method for constructing equilibria in this model and further devise techniques for implementing it.\(^1\)

2.1 Introduction

Consider a scenario in which several processes share a common cache according to the following rules. Over time, the processes receive requests to access pages, where each page request is designated to be accessed by a specific process (note that as the accessed data is also shared, different processes can request the same page). Each time a process is requested to access a page, the page must be cached first. If the cache is full, the process that needs to access the page decides which page will be evicted from the cache. Naturally, the processes seek to minimize the number of page faults incurred by their requests. Assuming that the processes are independent, they are expected to exhibit selfish behavior. Hence, each process strives to use as his paging policy an online algorithm which is the best response to the policies used by the other processes. In what respect should this response be the best? Equivalently, according to which decision criterion should the response be chosen? As the requests arrive in an online manner, and since the processes have no prior information about the requests (e.g. requests distribution), we concentrate on competitiveness, which by

\(^1\)Most of the results of this chapter were presented in a conference paper in [24].
now is the standard measure used for evaluating online algorithms, as the measure for the performance of possible paging policies. That is, a process chooses the paging policy with the best competitive ratio, given the policies of the other processes.

Suppose, for example, that all the processes are advised to use a least recently used (LRU) policy when evicting pages from the cache. Can any process guarantee itself a better competitive ratio by choosing a different (deterministic) policy? In other words, does the LRU policy constitute an equilibrium? Such questions come up in the context of many other scenarios that share similar attributes. However, it appears that previous work has either concentrated on the inherent uncertainty that comes from the online nature of the problem, or on game-theoretic aspects of selfish agents operating in a distributed system. In this chapter we develop a formal model for such scenarios in order to initiate the study of the issues just raised.

At first, consider a general setting in which there are several independent agents operating jointly in a common environment. The environment is likely to include some common infrastructure or resources that the agents may use, but as the agents are not necessarily identical, they may have different requirements or different access to the given resources. The agents need to satisfy a sequence of tasks, where each task is allocated to a specific agent. The tasks are given in an online fashion; the agents need to serve the tasks upon arrival, by making decisions based on the current state of the system, without knowing future events (specifically, there is no known distribution on the possible requests). Each agent runs, independently, an algorithm minimizing the cost it pays for serving the tasks.

Traditionally, such scenarios are modeled as an online problem, where the goal is to design an algorithm that minimizes the total cost incurred by the actions of all the agents. Essential to modeling the above as an online problem is the assumption that all the agents are controlled and managed by the same authority. However, in general, this assumption is problematic as the agents are independent, and hence, it is not likely that there exists a central authority that can enforce a policy or an algorithm (centralized or distributed) on the agents. Furthermore, the independence between the agents means that each agent is concerned about improving its own performance, while being indifferent to the performance of other agents. This immediately implies a possible lack of coordination between the agents, leading to selfish behavior, implying that an agent might deviate from a given algorithm if such a deviation improves its performance.

Such a deviation is crucial in many cases. For example, when designing protocols to be used as standards, no matter how efficient a protocol may be, it cannot be used as a standard if users have an incentive to deviate from it. Hence, a protocol used as a standard should be designed to induce an equilibrium among its users.

We conclude that modeling systems as a centralized online problem fails to capture a fundamental attribute, thus motivating the use of a game theoretic approach that takes into consideration the lack of coordination between agents. Such an approach can be used to predict the behavior of agents in a system, as well as to design stable standards that define equilibria. Recently, there has been a growing literature at the interface of game theory and algorithms, and in particular concerning the stability of standards (e.g. [2, 22, 60]). Nevertheless, so far, scenarios having an online nature were not addressed using that approach.\footnote{Some works ([13, 41, 42, 55, 57, 58]) combine online decision making with mechanism design, but the online decision-making role is limited to the mechanism itself, while the role of the agents is merely to supply the input to the online algorithm.}

8
2.1.1 Online Games

We now explain how to model a system as an online game. Basically, an online game is a pre-Bayesian game (equivalently, a game in informational form or an incomplete information game with strict type of uncertainty; for example, see [47]). In what follows we informally define the relevant components of the game: the players, their strategy spaces and cost (utility) functions, and possible states.

In an online game there are $N$ players denoted by $\{1,\ldots,N\}$. As each player controls a single agent in the system, the strategy space of player $i$ is the set of legal (defined below) online algorithms that determine the actions of the $i$th agent. Each possible state consists of a request sequence. The requests arrive in an online fashion into the system, and the agents run the online algorithms chosen by the players. Each request $r_j$ is allocated to a subset of the agents $A_j \subseteq \{1,\ldots,N\}$. Each agent in $A_j$ has a set of possible actions he may take to serve the request. The players in $A_j$ specify one of their possible actions simultaneously. Accordingly, an online algorithm for a player is considered legal if, for every request sequence, and for each request allocated to a player’s agent, the algorithm specifies a feasible possible action in response to the request, before the next request arrives to the system. The actions chosen by all the players in $A_j$ determine the cost that request $r_j$ incurs for each of the players.

In this chapter we concentrate on task allocation online games - online games with the property that for every request $r_j$, $|A_j|=1$. In other words, each request $r_j$ is allocated to a specific agent and each player is charged for the cost of all the operations of its agent. For convenience, in the rest of the chapter we use the term online game to refer to a task allocation online game.

The stable outcomes of the interactions of non-cooperative selfish agents correspond to the equilibria of the underlying game, that is, the points where unilateral deviation does not help any user improve its performance. An equilibrium of an online game is a vector of strategies, one for each player, from which no player has an incentive to deviate. While in strategic games the incentive for deviation is simply a lower cost, this is not the case in pre-Bayesian games due to the inherent uncertainty regarding the realized state. Moreover, as the players have no prior information about the realized state, the definition of a best response strategy traditionally relies on decision criteria that stem from worst case considerations (see [47, 6]). Some examples for such decision criteria are minimizing the maximum cost (taken over all possible states), minimizing the maximum regret (the difference between the cost paid and the minimum possible cost for the same state), and minimizing the competitive ratio.

Accordingly, the best response in online games is also likely to be defined primarily according to worst case decision criteria. We emphasize that this is a straight-forward implication of the inherent uncertainty in online games, and it is not clear whether there is a different reasonable alternative definition avoiding worst case considerations. Nevertheless, one might consider further refinement of the best response definition in which some additional criteria characterize the better strategy among the strategies having the same performance with respect to a primary worst case measure.

We focus on the competitive ratio as the decision criterion. Roughly speaking, it means that players measure the performance of their strategy with respect to an optimal offline strategy determined by an adversary that “knows” in advance both the request sequence and the players’ strategies. Specifically, given the strategies of the other players, a player minimizes the worst case ratio (taken over all possible request sequences) between the cost it pays and the cost paid by the adversary. This ratio is called the competitive ratio, although notice that a strategy might have different competitive ratios with respect to different
strategies of the other players.

Accordingly, we study the competitive ratio equilibrium, which is a vector of strategies, one for each player, in which no player can achieve a better competitive ratio by changing its strategy. As the best response strategy might be hard to design, implement, or compute, the equilibrium notion is relaxed, and we also consider approximate equilibria in which every player achieves a competitive ratio that is within a known factor away from the best competitive ratio attainable by any strategy. Identifying the set of solutions of an online game which are in (approximate) equilibrium is at the heart of the analysis of the game we define, and comprises the basis for performance evaluation in our system.

We note that the players in an online game may not necessarily “know” what are the strategies of the other players, however, as in a Nash equilibrium of any strategic game, the strategy of each player is a best response to the other players’ strategies. Moreover, as each strategy is actually an online algorithm, such a strategy can capture different responses to different events that might occur while the players are serving the request sequence.

2.1.2 Examples

We now turn to special cases that demonstrate the general settings discussed above. There are many scenarios that involve interaction between different agents: processors in a distributed system; processes in a computer; users accessing a server on the network; routers in a network; users in a peer-to-peer systems; and many more. For clarity, we concentrate on scenarios that were already modelled and studied as (centralized) online problems, yet the inherent selfishness of the users in these scenarios was not taken into account. Thus, the concept of an online game better models these examples.

We have already seen the paging game that generalizes the classic online paging problem. As a second example we consider file caching in distributed systems, e.g., peer-to-peer caches and web caches. In this caching game, there are \( N \) servers and each controls its own cache. The servers get requests for files and should serve them either by caching the file to their own cache or by accessing a remote replica of the file that is cached by another server. Caching a file incurs some (possibly constant) cost and might be constrained by the capacity of the cache, while accessing a remote replica incurs a cost which is typically proportional to the distance between the accessing server and the location of the replica.

Such scenarios have been previously examined and analyzed using different approaches. Early studies modelled the above scenario as an online problem (e.g., the distributed paging problem studied by Bartal, Fiat and Rabani in [10]). As can be expected, such models tend to ignore the selfish behavior of the servers in distributed systems. Recent works (e.g. [20]) indeed use a game-theoretic approach to overcome this problem, but they model caching scenarios as offline problems, while most of these scenarios are inherently online. We argue that the online caching game better describes these scenarios as it combines the advantages of the above approaches and captures the essential properties of such systems - both selfishness and online nature.

We consider the online generalized Steiner game where there are ISP’s operating in a common environment (which includes potential users) modelled as an edge-weighted undirected graph. Requests for connecting pairs of nodes in the graph arrive online. A request is served when all the edges of a path between the given nodes belong to the solution. Each request is designated to a specific ISP, that serves the request and pays for the edges that it adds to the solution. An edge which is added to the solution can be
re-used without being paid for again.

As these examples indicate, many scenarios have both an online nature and interaction between different agents, and thus are better described as online games. We thus turn to formally discuss the model of online games in its full generality, so as to establish the foundations for studying these scenarios. This discussion is followed by the study of specific online games that arise in many contexts.

2.1.3 Our Results

We present the new model of online games, which is used to study scenarios that are intrinsically online and in which different agents interact. We suggest the measure of competitive analysis as the players’ decision criterion, and introduce the appropriate basic concepts of the model, including the equilibria and the approximate equilibria of the game. We point out the question of finding equilibria as being of central importance. We also pay special attention to the incorporation of randomness in the model, as it differs from the theory of classic strategic games. We further introduce two desired properties of equilibria - efficiency and sub-game perfect equilibrium. Informally, the efficiency property excludes equilibria in which the players use “unfriendly” strategies that prevent the achievements of a good competitive ratio. Sub-game perfect equilibria are more stable than other equilibria.

In order to obtain general insights and develop generic techniques, we present an abstract model for the study of online games - the metrical task game. This model generalizes metrical task systems, a well-known abstract model for studying online problems. We identify a general useful principle for deriving equilibria in metrical task games, the non-malleability principle, which, roughly speaking, suggests that if the interaction between the agents is nullified, an equilibrium is easier to derive. We further develop the optimistic method that is aimed to implement the non-malleability principle by having each player assume a certain behavior on the part of the other players. Relying on that method, we devise techniques to construct equilibria (or even sub-game perfect equilibria) in some classes of metrical task games. According to these techniques, given an online game, some induced online problems are examined and competitive algorithms for these problems are designed and further exploited for constructing the equilibrium. The use of these techniques is demonstrated as they are applied in our study of some online games, e.g. the online generalized Steiner game.

Our discussion is followed by the study of a few natural examples for online games, some of which were introduced earlier. For the paging game we show that LRU defines an efficient sub-game perfect equilibrium with respect to deterministic strategies, while “plain” FIFO is not even competitive with respect to the individual player. Interestingly, this result gives another justification for preferring LRU over FIFO. For the caching game we design a strategy which yields an approximate equilibrium with respect to deterministic strategies, while for the online generalized Steiner game we show how to exploit the algorithm of [11] for the corresponding online problem to construct an efficient approximate equilibrium of the online game.

As this chapter introduces the new concept of online games, we establish the foundations needed for the study of this new area of research. While doing so, the study of online games is far from being complete. Thus, many questions are left open, and we note some possible further research directions.
2.1.4 Organization

The rest of this chapter is organized as follows. In Section 2.2 we formally define the central terms and notions used in the study of the online games. In Section 2.3 we study the paging game, while in Section 2.4 we study the list accessing game. Section 2.5 introduces the abstract model of metrical task games, and some general techniques for constructing an equilibrium. These techniques are used in Section 2.6 where the file caching game is discussed and in Section 2.7 where we study the online generalized Steiner game. We conclude in Section 2.8 with open questions and future research directions.

2.2 Preliminaries

We now turn to formalize the basic notions of online games. We note that the following definitions are stated with respect to cost minimization online games only, while the analogous definitions for profit maximization online games can be easily obtained.

We refer the reader to [14, Chapter 7] for the formal definition of an online problem. This definition can be generalized to formally describe the model of online games in the following way. Consider an online game with $N$ players. Instead of one set of feasible requests and actions, we are given a set of feasible requests and actions for each player. A strategy for a player is an online algorithm, that is, a function that assigns an action for each feasible history that ends with a request that is designated for that player. Notice that here the difference between an online game and an online problem is that a history in an online game includes additional information about the identity of the players for each request/action. Accordingly, when an online game generalizes a known online problem, we can use known online algorithms as strategies in a straightforward manner, that is, given the function from histories to actions that describes the algorithm, the corresponding strategy is a simple generalization that ignores the additional information about the players given in the history.

We use the standard notation for the strategy vector of the players, i.e. we denote by $s = (s_1, s_2, \ldots, s_N)$ a strategy vector where $s_i$ is the strategy (the online algorithm) played by the $i$th player. We denote by $s_{-i}$ the vector of strategies played by all the players except for the $i$th player. Given a request sequence $\sigma = r_1, r_2, \ldots, r_m$, let $\text{pref}_j(\sigma)$ be the prefix of $\sigma$ of length $j$, i.e. $\text{pref}_j(\sigma) = r_1, r_2, \ldots, r_j$. We denote by $p(r_i)$ the player that serves request $r_i$.

Given an online deterministic algorithm $A$ for the $i$th player, we use $A_{s_{-i}}(\sigma)$ to denote the cost incurred by $A$ when serving the $i$th player requests in the request sequence $\sigma$, while other players play the strategies given by $s_{-i}$. We denote by $OPT$ an optimal (offline) strategy for the $i$th player, i.e. a strategy that while serving the $i$th player’s requests incurs the minimum possible cost. We note that by this definition $OPT$ is assumed to know in advance both the entire request sequence $\sigma$ and the strategy vector $s_{-i}$.

**Definition 2.1** A deterministic algorithm $A$ is said to be $c$-competitive for the $i$th player with respect to a strategy vector $s_{-i}$ if there exists an $\alpha$ such that for every request sequence $\sigma$, $A_{s_{-i}}(\sigma) \leq c \cdot OPT_{s_{-i}}(\sigma) + \alpha$. The infimum over the set of all values $c$ such that $A$ is $c$-competitive for the $i$th player with respect to the strategy vector $s_{-i}$ is called the competitive ratio of $A$ for the $i$th player with respect to $s_{-i}$, and is denoted by $R_{s_{-i}}(A)$. 

12
In many of the scenarios modelled as online games the players are almost identical. Naturally, in such cases, the players will tend to use the same strategy in equilibrium, especially when such a strategy is agreed upon as a standard. Moreover, in cases of inherent symmetry between the players, e.g., symmetric games, it is easier to design equilibria that consist of one identical strategy, motivating the following definition.

**Definition 2.2 (Self-Competitiveness)** A strategy \( A \) is said to be \( c \)-self-competitive if for every \( j \) it is \( c \)-competitive for the \( j \)th player with respect to the strategy vector \( s_{-j} = (A, \ldots, A) \).

When considering deterministic strategies in an online game, a competitive ratio equilibrium (or simply an equilibrium) is a strategy vector \( s = (s_1, \ldots, s_N) \) such that for every \( i \), and for every possible deterministic strategy \( s'_i \), \( R_{s_{-i}}(s_i) \leq R_{s_{-i}}(s'_i) \). As equilibria are sometimes hard to construct or compute, the notion of approximate equilibria is also considered. A strategy vector \( s \) (of deterministic strategies) is an \( \alpha \)-equilibrium with respect to deterministic strategies, if for every \( i \) and every possible deterministic strategy \( s'_i \), \( R_{s_{-i}}(s_i) \leq \alpha \cdot R_{s_{-i}}(s'_i) \). We note that, by definition, every \( c \)-self-competitive strategy \( A \) gives rise to the \( c \)-equilibrium \( (A, \ldots, A) \). Moreover, for a \( c \)-self-competitive strategy \( A \), the vector \( (A, \ldots, A) \) might even be \( c' \)-equilibrium for \( c' < c \). To prove such a result, one should prove a lower bound on \( \min_B R_{s_{-i}}(B) \) where \( s_{-i} = (A, \ldots, A) \).

We now elaborate on several desired properties that equilibria solutions should satisfy.

**Efficiency.** In some games, an equilibrium can be constructed using strategies in which players mutually prevent the achievement of a good competitive ratio. Such equilibria are unlikely to get realized, as players tend to have very poor performance in these situations. This motivates defining measures for evaluating the efficiency of an equilibrium in an online game. We use the competitive ratio of each of the players in a given equilibrium as a measure that reflects the individual efficiency of each of the players. We also evaluate the overall efficiency of the system, including all agents, as follows. Consider a strategy vector as an online algorithm for the “social” online problem defined by the online game when all agents are controlled by a single entity. Let the competitive ratio of the strategy vector be its competitive ratio as an online algorithm for the appropriate “social” online problem. We use this competitive ratio as a measure for the overall efficiency of equilibria. Note that an optimal (competitiveness-wise) online algorithm that has centralized control of all the agents can be referred to as a social optimum of the online game, and accordingly, the closer the competitive ratio of an equilibrium to that of the social optimum, the better that equilibrium is. This motivates the study of the price of stability in online games, a topic that is deferred for further research.

**Sub-game Perfect Equilibrium.** The continual nature of online games implies that players might rethink their strategy as they serve the request sequence. Given a strategy vector defining an equilibrium, the players have no incentive to change their strategy while serving the requests, as long as no deviations from the given strategy vector happen\(^3\). However, in case a deviation occurs, there may be an incentive to one (or more) of the players to further deviate, and this process may justify itself, even though it probably should not have

\(^3\)In fact, one also needs to assume that the strategies of the players are optimized with respect to every prefix - otherwise, given a non-worst case prefix, a player might have an incentive to change its strategy. We note that in most online games there are no non-worst case prefixes as every prefix can be completed to a worst case request sequence (and hence such incentives are irrelevant), and moreover, the notion of sub-game perfect equilibria captures stability even in the presence of such prefixes.
happened in the first place. Hence, an equilibrium is considered to be more stable if it is resistant to some “local” deviations. Equilibria possessing this property are called sub-game perfect equilibria. Formally, a history of an online game \( G \) is a legal sequence of requests and reactions of the players. Accordingly, every history \( h \) induces a sub-game of \( G \) which is the online game defined according to \( G \) with the history \( h \) being initially realized. Finally, a sub-game perfect equilibrium of an online game \( G \) is a vector of strategies \( s \) such that for every sub-game \( G' \) of \( G \) it holds that the strategy vector induced by \( s \) in \( G' \) is an equilibrium of \( G' \).

### 2.2.1 Randomized Strategies

One of the differences between strategic games in the normal form and pre-Bayesian games (including online games) is the implications of randomization. A mixed strategy of player \( i \) in a strategic game (or a randomized strategy) is a probability distribution over the deterministic strategies of player \( i \). The same definition holds for pre-Bayesian games, and a mixed strategy in an online game is what we can intuitively interpret as a randomized online algorithm (strategy). Allowing mixed strategies in an online game gives rise to a new online game, which is referred to as the mixed extension of the given online game. To complete the definition of the mixed extension of an online game, note that given a vector of mixed strategies and a request sequence, the cost that a player pays is the expected cost, where the expectation is computed according to the given distributions over the strategies.

As the analysis of the mixed extension of an online game should comply with the decision criterion of competitive analysis, we extend the definition of competitive ratio accordingly. Notice that due to the worst case attribute of this decision criterion, the competitive ratio of a mixed strategy is not the expectation of the competitive ratios of the underlying deterministic strategies. To better understand this subtle issue, one can think of competitive analysis of randomized online algorithms as the concept we apply. As the competitive ratio of randomized online algorithms is determined with respect to a specific kind of adversary, we have a different set of definitions for each type of adversary. Currently, we focus on oblivious adversaries.

**Definition 2.3** A randomized algorithm \( A \) is said to be \( c \)-competitive for the \( i \)th player with respect to a strategy vector \( s_{-i} \) (which can include randomized strategies), if there exists \( \alpha \) such that for every request sequence \( \sigma \),

\[
E_r[A_{s_{-i}}(\sigma)] \leq c \cdot E_r[OPT_{s_{-i}}(\sigma)] + \alpha,
\]

where \( r \) denotes the coin tosses of all the players.

Here, assuming an oblivious adversary, \( OPT \) does not know in advance the coin tosses of the players, but rather only knows their actions (responses) till the time it has to serve its requests. Now, the competitive ratio of \( A \) for the \( i \)th player with respect to \( s_{-i} \) (oblivious adversary), can be defined similarly to Definition 2.1, and hence denoted by \( R_{s_{-i}}(A) \). The definition of self-competitiveness follows similarly to Definition 2.2.

While the mixed extension of a strategic game preserves some of the properties of the original game (played without randomization), extending an online game (and in general, a pre-Bayesian game) to allow randomized strategies does not have this property. Particularly, one of the important properties is that in a strategic game, every Nash equilibrium of the game is also a Nash equilibrium of its mixed extension. This is not the case for online games, due to the worst case attribute of the decision criterion used by the players. Hence, when considering both deterministic strategies and randomized strategies, the appropriate formal
definitions of *equilibrium* and *approximate equilibrium* of an online game follow from the definitions with respect to the mixed extension.

### 2.3 The Paging Game

In the paging game a cache of size \( k \) and a set of pages are shared between \( N \) processes. The processes get requests for pages, and each page has to be retrieved to the cache before being accessed by the processes. When a process needs to cache a page and the cache is full, it should choose which page to evict from the cache. If request \( r_i \) incurs a page fault, player \( p(r_i) \) is charged for the page fault. In what follows we prove the interesting difference between the known paging policies LRU and FIFO: while the former constructs an efficient sub-game perfect equilibrium, the latter results in a non-equilibrium strategy vector in which the players have an unbounded competitive ratio. Notice that in general, the known lower bounds for the online paging problem are still valid here, in particular the lower bound of \( k \) on the competitiveness of any deterministic online algorithm, and the lower bound of \( H_k \) on the competitiveness of any randomized online algorithm (see [14]).

#### 2.3.1 Algorithm LRU

Algorithm LRU for online paging is known to have competitive ratio of \( k \) (see [14]). A player in the paging game can use LRU as a strategy in one of two possible ways. The first, referred to as LRU\(_{self}\), is to ignore the requests served by the other players, and in case of a page fault to evict its least recently used page. The second way is by taking into consideration all requests, including those served by other players. Notice that this is the straightforward generalization of the online algorithm, hence we denote this strategy by LRU, and note that it requires that the players know the time of the recent accesses to the pages in the cache. Although one might expect that LRU\(_{self}\) is a better strategy than LRU for the selfish player, this is not the case: LRU\(_{self}\) is not \( c \)-self-competitive for every \( c \), while LRU is \( k \)-self-competitive. This can be easily understood by recalling that the accessed pages are shared among all the processes. Hence, no process has a set of pages that might be considered as its “own” pages and which it prefers to keep in the cache.

**Lemma 2.1** LRU\(_{self}\) is not \( c \)-self-competitive for every \( c \).

**Proof.** For simplicity, assume that there are two players, \( k = 2 \), and that there are 3 different pages, and consider the following (arbitrary long) request sequence: \( r_1 = 1, p(r_1) = 1; r_2 = 1, p(r_2) = 2; r_3 = 2, p(r_3) = 1; r_4 = 2, p(r_4) = 2; \forall i > 2, r_{2i−1} = 3, p(r_{2i−1}) = 1 \) and \( r_{2i} = 1, p(r_{2i}) = 2 \). If both players use LRU\(_{self}\) as their strategy, there will be a page fault in every request starting from the 5th request. A better strategy for the first player would be to evict page 2 when it serves the 5th request, and this would incur no further page faults. 

**Theorem 2.2** LRU is \( k \)-self-competitive.

**Proof.** We use the notion of \( k \)-phase partition (see [14, p. 37]). Fix a user \( i \) and a request sequence \( \sigma \), and let \( s_i = (LRU, \ldots, LRU) \). Given a strategy \( ALG \) for the \( i \)th player, we denote by \( S_j(ALG) \) the set...
of pages in the cache immediately after the first \( j - 1 \) requests were served and before the \( j \)th request is revealed.

We divide the request sequence into charging phases. Each charging phase is a non-empty subsequence of \( \sigma \). The first phase starts at the first request, while the other phases start right after the previous phase ends; all the phases, except maybe the last phase, end one request before the later of the following two events happens: There are requests for \( k + 1 \) different pages since the beginning of the phase; \( \text{OPT} \) evicts a page from the cache as a response to a request that is not the first request in the phase.

We now show that in each charging phase, \( \text{LRU} \) will pay at most \( k \). Fix a phase \( \rho \). If the phase does not terminate according to the second condition, then the claim follows from the definition of \( \text{LRU} \) (after a page is requested for the first time in \( \rho \), it will not be evicted during \( \rho \)). Otherwise, let \( r_j \) be the first request for the \( (k + 1) \)st different page in \( \rho \). As before, \( \text{LRU} \) will pay at most \( k \) until \( r_j \). Then, \( S_{j+1}(\text{LRU}) = S_{j+1}(\text{OPT}) \), as both caches will include all the \( k \) most recently used pages (as \( \text{OPT} \) does not evict pages and all the other players use \( \text{LRU} \) as their strategy). Now, as long as the second condition does not hold, both caches will contain the same set of pages. Obviously, if \( S_i(\text{LRU}) = S_i(\text{OPT}) \), and \( r_i \) results in a page fault for \( \text{LRU} \), then \( r_i \) results in a page fault for \( \text{OPT} \) too. Thus, the above implies that starting at \( r_j \), \( \text{LRU} \) will not evict any page till \( \rho \) ends.

The second termination condition for a phase implies that for every phase, except, perhaps, for the last one, we can match a different page eviction performed by \( \text{OPT} \), and the theorem follows.

We conclude that \((\text{LRU}, \ldots, \text{LRU})\) is an equilibrium with respect to deterministic strategies. Notice it is optimal (with respect to deterministic strategies) according to both our efficiency criteria: the competitive ratio of the players is the lowest possible, and so is the competitive ratio of the equilibrium itself, as the interaction between the strategies results in the same behavior of the online (centralized) \( \text{LRU} \) paging policy. Moreover, the proof of Theorem 2.2 can be generalized to prove that this equilibrium is in fact a sub-game perfect equilibrium.

### 2.3.2 Algorithm FIFO

Algorithm \( \text{FIFO} \) for online paging is \( k \)-competitive [14]. To derive an appropriate strategy for the paging game from \( \text{FIFO} \), assume that each page in the cache has a tag specifying the last time it was brought to the cache. A player that uses the \( \text{FIFO} \) strategy evicts the oldest page in cache.

**Theorem 2.3** \( R((\text{FIFO}, \ldots, \text{FIFO}))(\text{FIFO}) = \infty \).

**Proof.** We construct an arbitrarily long sequence of requests such that the cost paid by the first player when using \( \text{FIFO} \) is arbitrarily large compared to the cost it pays when using a different strategy.

For convenience, in what follows we assume that \( k \) is even. Assume that initially the cache is empty, and that there are \( \frac{3}{2} k \) different pages which will be denoted by \( \{1, \ldots, \frac{3}{2} k\} \). The request sequence is defined by:

\[
\begin{align*}
  r_i = \begin{cases} 
  i & i = 1, \ldots, k + 1 \\
  j & i = k + 2j, \ k > j \geq 1 \\
  r_{i-(2k-1)} & i = k + 2j, \ j \geq k \\
  k + j + 1 & i = k + 1 + 2j, \ \frac{k}{2} > j \geq 1 \\
  r_{i-(k+1)} & i = k + 1 + 2j, \ j \geq \frac{k}{2}
  \end{cases}
\end{align*}
\]

and

\[ p(r_i) = \begin{cases} 
2 & i = 1, \ldots, k \\
1 & i = k + 1 \\
1 & i = k + 2j, j \geq 1 \\
2 & i = k + 1 + 2j, j \geq 1 
\end{cases} \]

The following tables sketch the different phases of the sequence.

<table>
<thead>
<tr>
<th>Initial phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i )</td>
</tr>
<tr>
<td>( r_i )</td>
</tr>
<tr>
<td>( p(r_i) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>First intermediate phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i )</td>
</tr>
<tr>
<td>( r_i )</td>
</tr>
<tr>
<td>( p(r_i) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Second intermediate phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i - (k + 1) )</td>
</tr>
<tr>
<td>( r_i )</td>
</tr>
<tr>
<td>( p(r_i) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>The first part of the final phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i - (2k - 1) )</td>
</tr>
<tr>
<td>( r_i )</td>
</tr>
<tr>
<td>( p(r_i) )</td>
</tr>
</tbody>
</table>

**Lemma 2.4** If all the players are using the \( FIFO \) strategy, after the \( k \)th request and till the end of the sequence, the queue contains the last \( k \) requested pages according to the order they have been requested. Specifically, every \( k + 1 \) consequent requests are for different pages and \( r_i \) is not in the cache.

**Proof.** By induction. Obviously, after the \( k \)th request is served, the queue contains the pages 1, \ldots, \( k \) according to the order they have been requested. Hence \( r_{k+1} = k + 1 \) is not in the cache.

Assume that the claim holds after \( r_{i-1} \) is served. There are four different cases.

- \( i = k + 2j, k > j \geq 1 \) - as the previous request for \( r_i \) is \( r_j \), it follows from the induction hypothesis that \( r_i \) is not in the cache.
- \( i = k + 1 + 2j, \frac{k}{2} > j \geq 1 \) - as \( r_i \) is the first request for this page, it is obviously not in the cache.
• \( i = k + 1 + 2j, \ j \geq \frac{k}{2} \) - by the induction hypothesis, all the pages in the set \( \{r_{i-j}\}_{j=1}^{k+1} \) are different. Moreover, the cache includes \( \{r_{i-j}\}_{j=1}^{k} \) when \( r_i \) is invoked, and hence the claim follows.

• \( i = k + 2j, \ j \geq k \) - we need to show that \( r_i \notin \{r_{i-j}\}_{j=1}^{k} \). By definition it is enough to prove that \( r_{i-(2k-1)} \notin \{r_{i-j}\}_{j=1}^{k} \). Firstly, notice that \( r_{i-(2k-1)} \in \{r_{i-k+1-j}\}_{j=1}^{k-2} \), and by the induction hypothesis
\[
\begin{align*}
  r_i = r_{i-(2k-1)} \notin \{r_{i-j}\}_{j=1}^{k-3}.
\end{align*}
\]
Specifically, it means that \( r_i = r_{i-(2k-1)} \neq r_{i-k} \). Similarly,
\[
\begin{align*}
  r_i = r_{i-(2k-1)} \notin \{r_{i-j}\}_{j=1}^{k-1}.
\end{align*}
\]
Next, as \( i \geq 3k \), it holds that \( i - (k-1) \geq 2k + 1 \) and hence \( r_{i-j} = r_{i-j-(k+1)} \) for every odd \( 1 \leq j \leq (k-1) \). Thus, for \( j = k-1 \) we have that \( r_{i-(k-1)} = r_{i-(k-1)-(k+1)} = r_{i-2k} \) and, by (2.2), \( r_i \neq r_{i-2k} = r_{i-(k-1)} \). For every odd \( 1 \leq j \leq (k-3) \) we have that by (2.1), \( r_i \neq r_{i-(k-j-1)} = r_{i-j} \). It is left to prove that \( r_i = r_{i-(2k-1)} \neq r_{i-j} \) for even \( 2 \leq j \leq k-2 \). We distinguish between two cases:

1. \( i-j \geq 3k \) - it holds that \( r_{i-j} = r_{i-j-(2k-1)} \), and by (2.2), \( r_i \neq r_{i-j-(2k-1)} = r_{i-j} \).
2. \( i-j < 3k \) - it holds that \( 3k \leq i < 3k+j \) and \( k+1 \leq i-(2k-1) < 3k+j-2k+1 = k+j+1 < 2k-1 \) hence \( r_i = r_{i-(2k-1)} \geq k+1 \). At the same time, since \( 2k+2 \leq i-j < 3k \), \( r_{i-j} = \frac{i-j-k}{2} < k \) and the claim follows.

\[ \blacksquare \]

**Lemma 2.5** If the first player evicts page \( k \) when serving \( r_{k+1} \), then he has no further page faults.

**Proof.** Assuming that the first player evicts page \( k \) when serving \( r_{k+1} \), the FIFIFO queue contains \( 1, \ldots, k-1, k+1 \) in this order (1 is the next to be taken out) after \( r_{k+1} \) is served. Obviously, the next \( \frac{k}{2} - 1 \) requests that are served by the second player are for pages that are not in the cache (and won’t be in the cache when they are requested), and hence the queue is updated accordingly. However, for \( 1 \leq j \leq \frac{k}{2} \) the first player needs to access page \( j \) at time \( k+2j \). Since till this time there has been only \( j-1 \) page faults of the second player, and according to the order of the pages in the queue, it follows that \( r_{k+2j} \) will not incur a page fault. Accordingly, after \( r_{2k} \) is served, the pages \( \frac{k}{2}, \frac{k}{2}+1, \ldots, k-1, k+1, k+2, \ldots, \frac{3k}{2} \) are in the queue in this order.

This is the beginning of the second intermediate phase. Again, the next \( \frac{k}{2} \) requests that are served by the second player \( (k, 1, 2, \ldots, \frac{k}{2} - 1) \) are for pages that are not in the cache (and won’t be in the cache when they are requested), and hence the queue is updated accordingly. Also, for \( \frac{k}{2} \leq j < k \) the first player needs to access page \( j \) at time \( k+2j \). Since till this time there has been only \( j-\frac{k}{2} \) page faults of the second player in this phase, and according to the order of the pages in the queue at the beginning of the phase, it follows that \( r_{k+2j} \) will not incur a page fault. Accordingly, after \( r_{3k-1} \) is served, the pages \( k+1, k+2, \ldots, \frac{3k}{2}, k, 1, 2, \ldots, \frac{k}{2} - 1 \) are in the queue in this order. Thus, \( r_{3k} \) does not incur a page fault, but \( r_{3k+1} \) does.

Now, we have the following proposition which completes the proof.
Proposition 2.6 After $r_{3k+1}$ is served and till the end of the sequence, the cache contains the $k$ most recent requests that were designated to the second player. Specifically, these requests are for different pages and no page faults are incurred by the requests designated to the first player.

Proof. By induction. After $r_{3k+1}$ is served the claim holds by the above discussion.

Notice that every request $r_i$, $i > 3k$, that is designated to the first player is for $r_{i-(2k-1)}$, which is one of the $k$ most recent requests that were designated to the second player. Hence, by the induction hypothesis, it is in the cache and incurs no page fault.

As for the requests $r_i$ that are designated to the second player, $i > 3k+1$, it holds that $r_i = r_{i-(k+1)}$.

By Lemma 2.4, it holds that $r_{i-(k+1)} \notin \{r_{i-j}\}_{j=2}^{k}$ and $r_{i-(k+j)} \notin \{r_{i-(k+j)}\}_{j=2}^{k}$. As the $k$ most recent requests that were designated to the second player are $r_{i-j}$ for even $2 \leq j \leq 2k$, the claim follows.

2.3.3 Algorithm $SIM$ - an algorithm with a bounded competitive ratio

In this section we present an algorithm that achieves a bounded competitive ratio, regardless of the strategies of the other players. We use this algorithm to conclude that algorithms with unbounded self-competitive ratio (such as $FIFO$) do not comprise an equilibrium, as they are not the best response for themselves.

The idea behind algorithm $SIM$ is to simulate every possible adversary. The algorithm works in phases. In each phase we charge each adversary with at least one page fault while algorithm $SIM$ will be charged with at most $F = \binom{T}{k}$ page faults, where $T$ is the total number of pages in the system.

Assume that player $i$ is using algorithm $SIM$ as its strategy, and the strategies of the other players are fixed to be $s_{-i}$. Algorithm $SIM$ maintains a data structure that holds $F$ entries, one for each of the possible initial configurations of the cache at the beginning of the phase (so each entry stands for a possible adversary). Each entry also holds a flag which can be marked either ‘charged’ or ‘uncharged’, and the current simulated content of the cache for that entry (the simulated content of the cache is initialized according to the configuration that corresponds to that entry). In the beginning of a new phase, all entries are marked ‘uncharged’ and their simulations are reset to be the $F$ different cache configurations. As requests arrive along the phase, for each uncharged configuration, the algorithm updates the simulated content of the cache (so we have the content of the cache as it would be if the phase started with the configuration that corresponds to that entry). Notice that in order to simulate the content of the cache (given a starting configuration), the algorithm needs to use the knowledge of $s_{-i}$. As long as no page faults were introduced for the $i$th player during the simulation, the simulation is successful regardless of the adversary’s choices (as it had no choices to make so far in that phase for that simulation). Once a simulation for an entry results a page fault for the $i$th player, that entry is marked ‘charged’, and the algorithm stops simulating it. Once all the configuration are charged, the phase is over and a new phase starts.

As for the actual page eviction choices of algorithm $SIM$, each phase is divided into steps. In each step, the algorithm picks an entry that is uncharged and updates the content of the cache according to the simulated content of the cache for that entry and that moment. Once algorithm $SIM$ has a page fault, the step ends and a new step begins (a new uncharged entry is picked and so on).

Theorem 2.7 For any strategy vector $s_{-i}$, algorithm $SIM$ has a competitive ratio of at most $F$. 

19
**Proof.** Consider a specific phase. Each time algorithm SIM has a page fault a step ends. At the beginning of that step an uncharged entry was chosen and the content of the cache was updated accordingly by algorithm SIM. It follows that the simulation of the chosen entry resulted a page fault for the \( i \)th player when the algorithm had the page fault that ended the step. So, in each step at least one entry is marked ‘charged’. There are \( F \) entries, so there are at most \( F \) steps in a phase and hence algorithm SIM has at most \( F \) page faults in each phase.

Now consider a specific phase and the content of the cache at the beginning of that phase when OPT is playing. There exists an entry that correspond to that content, and that entry will have an updated simulation of the content of the cache as long as OPT has no page faults. Only once OPT has a page fault, the entry is marked ‘charged’. Since the phase is not over till all the entries are charged, by the time the phase ends OPT had at least one page fault during that phase.

To conclude, in each phase algorithm SIM has at most \( F \) page faults and OPT has at least one page fault. The theorem follows.

It follows that \((FIFO, \ldots, FIFO)\) is not an equilibrium: player \( i \) can use algorithm SIM and achieve a bounded competitive ratio according to Theorem 2.7, while playing FIFO results an unbounded competitive ratio according to Theorem 2.3.

### 2.4 The Static List Accessing Game

In this section we study another simple online game which generalizes the well known static list accessing online problem. In this online game a list is shared between \( N \) players (for example, it is located on a server and \( N \) different players access it). Given a request \( r_i \), player \( p(r_i) \) should access the given element in the list. Each player is paying for his actions (accessing and paid transpositions).

#### 2.4.1 Lower Bounds

In general, the known lower bounds for the online problem are all valid for the online game, including the lower bound of \( 2 - \frac{2}{\ell + 1} \) on the competitiveness of any deterministic online algorithm (see [14]), and the lower bound of \( \frac{3}{2} - \frac{2}{\ell + 5} \) on any randomized algorithm against oblivious adversaries ([74]).

#### 2.4.2 Algorithm Move-to-Front

One of the well-known online algorithms for the static list accessing problem is Move-to-Front (MTF), which is also a a valid strategy in the online game. When using this algorithm, after every access to an item, the item is brought to the front of the list.

**Theorem 2.8** \( MTF \) is \((2 - \frac{1}{\ell})\)-self-competitive where \( \ell \) is the length of the given list.

**Proof.** The proof follows the same ideas as the proof of the competitiveness of \( MTF \) (see [14]). We sketch the proof. At the heart of the proof of \( MTF \) is a potential function, \( \Phi \), which is defined to be the number of inversions between the list of the player playing \( MTF \) and the list of the player playing OPT.

Fix a player \( k \), and assume that all other players use \( MTF \) as their strategy. Similarly to the proof of the competitiveness of \( MTF \), for each request designated to the \( k \)th player, it holds that the cost paid by
\( MTF \) plus the change in the potential function is bounded by, roughly speaking, twice the cost paid by \( OPT \). For each request designated to a player different from \( k \), the cost for the \( k \)th player is 0, regardless of his strategy, and the value of the potential function can only decrease. Hence, by the potential function argument, the total cost paid by \( MTF \) is at most \((2 - \frac{1}{2})\) times the cost paid by \( OPT \).

The lower bounds in the general case imply that \((MTF, \ldots, MTF)\) is a \((1 + \Theta(\frac{1}{2}))\)-equilibrium with respect to deterministic strategies, and a \((\frac{4}{3} + \Theta(\frac{1}{2}))\)-equilibrium against an oblivious adversary.

### 2.4.3 Algorithm TIMESTAMP

We now turn to examine another well known deterministic online algorithm, Algorithm TIMESTAMP, which is known to be 2-competitive (see [14]). The algorithm is as follows.

**Algorithm TIMESTAMP:** Upon receiving a request for item \( x \), insert \( x \) in front of the first (from the front of the list) item \( y \) that precedes \( x \) on the list and was requested at most once since the last request for \( x \). Do nothing if there is no such item \( y \), or if \( x \) is requested for the first time.

In contrast with \( MTF \), it is not obvious how to use algorithm TIMESTAMP to derive an online strategy for the list accessing game. In what follows we discuss one possibility, in which each player applies the algorithm with respect to his requests only, while ignoring requests served by other players. We denote the resulting strategy by \( TS \). This approach is vital when the players do not know the other players’ requests (notice that this is not required to apply the \( MTF \) strategy).

**Theorem 2.9** For every \( i \) and for \( s_{-i} = (TS, \ldots, TS) \), \( R_{s_{-i}}(TS) \) is unbounded.

**Proof.** Assume that we are given a list which is initially ordered as follows: \( x_1, x_2, \ldots, x_k, z, y_1, \ldots, y_k \). Consider the following request sequence and the cost for the first player, assuming that the second player uses the \( TS \) strategy.

#### Initialization phase:

<table>
<thead>
<tr>
<th>( r_i )</th>
<th>( p(r_i) )</th>
<th>Cost</th>
<th>New configuration</th>
<th>Cost</th>
<th>New configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 )</td>
<td>1</td>
<td>1</td>
<td>( x_1, x_2, \ldots, x_k, z, y_1, \ldots, y_k )</td>
<td>1</td>
<td>( x_1, x_2, \ldots, x_k, z, y_1, \ldots, y_k )</td>
</tr>
<tr>
<td>( y_1 )</td>
<td>2</td>
<td>–</td>
<td>( x_1, x_2, \ldots, x_k, z, y_1, \ldots, y_k )</td>
<td>–</td>
<td>( x_1, x_2, \ldots, x_k, z, y_1, \ldots, y_k )</td>
</tr>
<tr>
<td>( x_1 )</td>
<td>1</td>
<td>1</td>
<td>( x_1, x_2, \ldots, x_k, z, y_1, \ldots, y_k )</td>
<td>1</td>
<td>( x_1, x_2, \ldots, x_k, z, y_1, \ldots, y_k )</td>
</tr>
<tr>
<td>( y_2 )</td>
<td>2</td>
<td>–</td>
<td>( x_1, x_2, \ldots, x_k, z, y_1, \ldots, y_k )</td>
<td>–</td>
<td>( x_1, x_2, \ldots, x_k, z, y_1, \ldots, y_k )</td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>( x_{1} )</td>
<td>1</td>
<td>1</td>
<td>( x_1, x_2, \ldots, x_k, z, y_1, \ldots, y_k )</td>
<td>1</td>
<td>( x_1, x_2, \ldots, x_k, z, y_1, \ldots, y_k )</td>
</tr>
<tr>
<td>( y_{k} )</td>
<td>2</td>
<td>–</td>
<td>( x_1, x_2, \ldots, x_k, z, y_1, \ldots, y_k )</td>
<td>–</td>
<td>( x_1, x_2, \ldots, x_k, z, y_1, \ldots, y_k )</td>
</tr>
<tr>
<td>( x_{k} )</td>
<td>1</td>
<td>1</td>
<td>( x_1, x_2, \ldots, x_k, z, y_1, \ldots, y_k )</td>
<td>1</td>
<td>( x_1, x_2, \ldots, x_k, z, y_1, \ldots, y_k )</td>
</tr>
<tr>
<td>( x_{k} )</td>
<td>2</td>
<td>–</td>
<td>( x_1, x_2, \ldots, x_k, z, y_1, \ldots, y_k )</td>
<td>–</td>
<td>( x_1, x_2, \ldots, x_k, z, y_1, \ldots, y_k )</td>
</tr>
</tbody>
</table>

The main phase:
Hence, we need the following change so that the $TS$ player will not insert $z$ before $x_1$:
2.4.4 Algorithm BIT

Algorithm BIT is a randomized online algorithm for list accessing problem. For each element \( x \) in the list, a bit \( b(x) \) is randomly initialized. When a request to access element \( x \) arrives, bit \( b(x) \) is complemented, and if \( b(x) = 1 \) then the element is moved to the front of the list. If every player maintains a separate set of random bits, and uses those bits to implement a strategy following the ideas of Algorithm BIT, the resulting strategy is self-competitive. The analysis of Algorithm BIT can be easily extended to prove that the BIT strategy is \( \frac{1}{2} \)-self competitive. Thus, we conclude that \((\text{BIT}, \ldots, \text{BIT})\) is a \( (\frac{1}{8} + \Theta(\frac{1}{k})) \)-Nash Equilibrium against an oblivious adversary.

2.5 Metrical Task Games

We consider the metrical task game in this section as an abstract model for analyzing online games. The metrical task game, generalizing metrical task systems, captures a wide range of cost minimization games.

2.5.1 The Model

In what follows we use terminology similar to that of [14, Chapter 9]. A metrical task game consists of a set of \( N \) players, a set of points (configurations/states) \( S \), and for each player \( i \), a distance function \( d_i : S \times S \rightarrow \mathbb{R}^+ \cup \{\infty\} \) that satisfies triangle inequality, and a set of allowable tasks \( \mathcal{R}_i \subseteq \mathbb{R}^{|S|} \). We note that for the sake of generalization we do not require symmetry, reflexivity, or positivity of the distance function (although this means that the name is a bit misleading). A legal request in a metrical task game is a task \( r \) such that \( r \in \mathcal{R}_{p(r)} \). Roughly speaking, a strategy of player \( i \) is a function that assigns a target

<table>
<thead>
<tr>
<th>( x )</th>
<th>( n )</th>
<th>( x ), . . . , ( y ), . . . , ( z )</th>
<th>( y )</th>
<th>. . .</th>
<th>( x ), . . . , ( y ), . . . , ( z )</th>
<th>. . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>1</td>
<td>1</td>
<td>. . .</td>
<td>. . .</td>
<td>( x ), . . . , ( y ), . . . , ( z )</td>
<td></td>
</tr>
<tr>
<td>( x )</td>
<td>2</td>
<td>( n )</td>
<td>. . .</td>
<td>. . .</td>
<td>. . .</td>
<td>. . .</td>
</tr>
<tr>
<td>( x )</td>
<td>1</td>
<td>1</td>
<td>. . .</td>
<td>. . .</td>
<td>( x ), . . . , ( y ), . . . , ( z )</td>
<td></td>
</tr>
<tr>
<td>( x )</td>
<td>2</td>
<td>( n )</td>
<td>. . .</td>
<td>. . .</td>
<td>. . .</td>
<td>. . .</td>
</tr>
<tr>
<td>( y )</td>
<td>1</td>
<td>( n )</td>
<td>. . .</td>
<td>. . .</td>
<td>. . .</td>
<td>. . .</td>
</tr>
<tr>
<td>( y )</td>
<td>2</td>
<td>1</td>
<td>. . .</td>
<td>. . .</td>
<td>. . .</td>
<td>. . .</td>
</tr>
<tr>
<td>( y )</td>
<td>2</td>
<td>( n )</td>
<td>. . .</td>
<td>. . .</td>
<td>. . .</td>
<td>. . .</td>
</tr>
<tr>
<td>( y )</td>
<td>2</td>
<td>( n )</td>
<td>. . .</td>
<td>. . .</td>
<td>. . .</td>
<td>. . .</td>
</tr>
</tbody>
</table>
configuration for every history, which is a sequence \( \{(q_{j-1}, r_j)\}_{j>0} \) of pairs of a configuration and a legal request.

Given an initial configuration and a sequence of legal requests, each request is served by the player it is designated to. While a request is being served, the configuration of the system can be changed by other players. If at the time that request \( r \) is invoked, the current configuration of the system is \( q \), then player \( p(r) \) can first change the configuration of the system to any desired configuration \( q' \) such that \( d_i(q, q') < \infty \) (and sometimes we also require that \( r(q') < \infty \)), incurring a transition cost of \( d_i(q, q') \). Then, the player serves the task in state \( q' \), incurring a processing cost of \( r(q') \).

### 2.5.2 The Non-malleability Principle

While looking for (approximate) equilibria in a metrical task game, we face two related difficulties. The first difficulty comes from the search for a competitive algorithm for each player. The second difficulty is guaranteeing that an equilibrium results from the interaction between the algorithms of the players. While solving the first difficulty is somewhat similar to designing competitive algorithms, the second one seems to be inherently different from any other familiar algorithmic problem. Thus, coping with both difficulties at once might be a hard task. A possible way to get around this hardness is by developing techniques that will enable us to cope with each part separately. Motivated by this idea, we examine the interaction between online strategies, and in particular the impact of this interaction on the performance of each strategy.

The decisions made by online algorithms can depend on the current configuration of the system. When online strategies interact, they might be sensitive to configuration changes caused by other players. Moreover, a player using an online algorithm might be manipulated by other players who change the configuration of the system wisely, and in particular by an adversary who can construct the request sequence in advance. Consequently, an algorithm which is competitive with respect to the online problem induced by \( S, d_i \) and \( R_i \), might yield a strategy which is not (self-) competitive due to the interaction between the players, and the fact that other players can be manipulated by the adversary. The FIFO strategy for the paging game is an example in which the adversary can construct a request sequence in which, by evicting pages not according to the FIFO policy, it causes other players to evict pages different from those it needs to serve, and hence avoids page faults. From the perspective of the players, the outcome is the poor performance of FIFO that is reflected in Theorem 2.3. In order to overcome the above phenomenon, and with the goal of deriving online strategies that are in equilibrium from competitive online algorithms in mind, we suggest the non-malleability principle. The idea behind the principle is to prevent players from manipulating the game.

**The non-malleability principle.** A strategy vector \( s \) complies with the non-malleability principle if for every \( i \), strategy \( s_i \) can be described as a function of the request sequence and the initial configuration only, and it is independent of the strategies of the other players.

The non-malleability principle can be said to nullify interaction between players, and thus the problem of finding equilibria can be reduced to the problem of finding for each player a “good” strategy. Specifically, each player actually faces an induced online problem - the metrical task system \( MTS_i \) induced by \( S, d_i \) and \( R_i \) - but with the exception that there might be some external configuration changes in between requests. These changes do not incur any cost to the player and they are independent of the algorithm the player uses.

---

\(^4\)The paging game seems to be an exception in which a known competitive algorithm leads to a simple solution for the problem.
Designing competitive algorithms for this kind of online problem can be done either from scratch or based on known competitive algorithms for metrical task system without the external configuration changes.

2.5.3 The Optimistic Method

We present here the optimistic method which implements the non-malleability principle, together with two techniques that are used in different cases and are based on this method.

The optimistic method constructs a strategy vector \( s \) in which all players make their decisions while assuming that the current configuration is some anticipated configuration, and ignoring the actual current configuration. Obviously, such a strategy vector complies with the non-malleability principle. Due to performance considerations, we would like the anticipated configuration to be the current configuration, and thus we set the anticipated configuration to be the configuration resulting from the players playing according to \( s \). In order to apply the optimistic method it is usually required that every request is publicly known once it is invoked, as every player must know all past requests of all the players in order to calculate the anticipated configuration and implement its strategy.

The difficulty in implementing the optimistic method originates from the need to make sure that the resulting strategies are feasible, since ignoring the current configuration might lead to illegal configuration changes by the players. We devise two techniques that are based on the optimistic method, where each technique can be applied to a different kind of online games. Both techniques reduce the problem of constructing an equilibrium to the problem of designing competitive algorithms for the induced online problems \( \{MTS_i\} \). These algorithms are then converted to strategies by operating according to the anticipated configuration instead of according to the current one. In the adjustment technique the anticipated configuration is calculated in a straightforward manner, and hence, due to feasibility considerations, the applicability of the technique is rather limited. On the other hand, the structure of the games on which the indifference technique is applied facilitates the use of a more relaxed approach that still guarantees feasibility.

The Adjustment Technique

In the adjustment technique, the players determine the configuration changes independently of the current configuration. As a result, after each player serves a request, the configuration depends only on the request sequence (and maybe the initial configuration), and thus it can be said to be adjusted to \( s \). This adjustment is crucial when one of the players deviates from \( s \), as this deviation would not have further effect on the system once an adjustment move is made.

Formally, given a strategy vector \( s \), a request sequence \( \sigma = \{r_j\}_j \) and an initial configuration \( q_0 \), we recursively define the history generated by \( s \) on the request sequence \( \sigma \) when initiated at \( q_0 \) as \( h(\sigma, s, q_0) = \{(q_{j-1}, r_j)\}_j \), where for every \( j > 0 \), \( q_j = s_{p(r_j)}(h(pref_j(\sigma), s, q_0)) \) is the configuration of the system after player \( p(r_j) \) serves the \( j \)th request given that all the players played according to \( s \). Now, let \( s_{i}^{adj} \) be a strategy for the \( i \)th player defined with respect to \( s \) as follows: \( s_{i}^{adj}(h(\sigma, s', q_0)) = s_i(h(\sigma, s, q_0)) \) for every \( \sigma, s' \) and \( q_0 \).

**The Adjustment Technique.** Given a strategy vector \( s \), construct a strategy vector \( s_i^{adj} \) by replacing each strategy \( s_i \) with the strategy \( s_{i}^{adj} \).
To ensure feasibility of $s^{adj}_i$, the state transitions made by $s^{adj}_i$, for every $i$, must be feasible regardless of the strategies played by the other players. Hence, the applicability of the adjustment technique depends on the metrical task game and the strategy vector $s$. A metrical task game on which the adjustment technique can be applied to every strategy vector $s$ is a game in which all transition costs are finite, i.e., $d_i(q, q') < \infty$ for every player $i$, and configurations $q$ and $q'$. We call such a metrical task game a total sharing game.

The adjustment technique yields equilibrium if it is applied to competitive algorithms for the induced online problems, $\{MTS_i\}$, with the possible external configuration changes. Such algorithms can be based on competitive algorithms for $\{MTS_i\}$ (without external configuration changes), if they are adapted to external configuration changes. A simple way to adapt an online algorithm $A$ to external configuration changes is by resetting the online algorithm after every external configuration change, i.e., following a request allocated to a different player in the game, the player forgets about any past requests and serves the next request as if it was the first request allocated to it. Formally, abusing notation, we denote by $A^{adj}_i$ the strategy for the metrical task game obtained when applying both this modification and the adjustment technique.

**Theorem 2.10** Let $G$ be a total sharing metrical task game, and let $\{A_i\}_{1 \leq i \leq N}$ be online algorithms such that for every $i$, $A_i$ is strictly $\alpha_i$-competitive online algorithm for $MTS_i$ for any initial configuration $q_0 \in S$. Then, for every $i$, $A^{adj}_i$ is strictly $\alpha_i$-competitive with respect to the strategy vector $\{A^{adj}_j\}_{-i}$ in $G$.

**Proof.** Fix $i$ and assume that all the players except the $i$th player are playing according to the strategy vector $\{ALG^{adj}_j\}_{-i}$. Fix a request sequence $\sigma$, and let $OPT(\sigma)$ denote an optimal strategy for the $i$th player for serving $\sigma$. Denote by $ALG(\sigma)$ the cost incurred by the strategy $ALG$ while serving $\sigma$ in $G$. Let $\sigma_j = r_{j-1}, r_j, r_{j+1}, \ldots, r_{j+|\sigma_j|-1}$ be the $j$th maximal subsequence of requests in $\sigma$ allocated to the $i$th player (i.e. $p(r_{j-1}) = i$ and $p(r_{j+|\sigma_j|}) = i$ for every $j$). By the definition of $j_s$ and the adjustment technique, $q_{j_s-1}$ - the configuration of the system when the $j_s$th request is invoked - is fully defined by $\sigma$ and the strategy vector $\{ALG^{adj}_j\}_{-i}$, regardless of the $i$th player strategy.

Denote by $ALG(\sigma, q)$ the cost incurred by the online algorithm $ALG$ while serving $\sigma$ in $MTS_i$ when the initial state is $q$. Finally, denote by $OPT(\sigma, q)$ the optimal solution for serving $\sigma$ in $MTS_i$ when the initial state is $q$.

Then, $A^{adj}_i(\sigma) = \sum_j A_i(\sigma_j, q_{j_s-1}) \leq \sum_j \alpha_i \cdot OPT(\sigma_j, q_{j_s-1}) \leq \alpha_i \cdot OPT(\sigma)$, where the equality follows from the definition of the strategy $A^{adj}_i$, the first inequality follows from the competitiveness of $A_i$ and the last inequality follows as the optimal strategy induces solutions for the online problems $\{(\sigma_j, q_{j_s-1})\}_j$.

By applying Theorem 2.10 on $FIFO$ we get a self-competitive strategy as the paging game is a total sharing metrical task game and $FIFO$ is a strictly competitive for the paging problem. Notice that Theorem 2.10 can be generalized for every metrical task game and strategy vector to which the adjustment technique can be applied. Also note that the adjustment technique results in strategies that maintain their competitiveness given any history, and hence it can be used to construct sub-game perfect (approximate)-equilibria.
The Indifference Technique

We define a product metrical task game to be a game satisfying the following two properties. First, its configuration set can be written as \( S = \times_{1 \leq i \leq N} S_i \), and hence every configuration \( q \) can be described by a vector of length \( N \), \( q = (q_1, \ldots, q_N) \), where \( q_i \in S_i \). Second, every distance function \( d_i \) satisfies for every \( q \) and \( q' \), such that there exists \( j \neq i \) with \( q_j \neq q'_j \), \( d_i(q, q') = \infty \).

The second property implies that a player can only change “its” coordinate in the configuration vector, and thus any strategy of the \( i \)th player can be described as a function that assigns \( q_i \in S_i \) for every history. Thus, using similar notations, \( h(\sigma, s, q_0) = \{ (q_{j-1}, r_j) \}_{\sigma, j} \), where for every \( j > 0 \), \( (q_j)p(r_j) = s_{p(r_j)}(h(\sigma_{p(r_j)}, s, q_0)) \) and \( (q_j)_k = (q_{j-1})_k \) for every \( k \neq p(r_j) \). Now, let \( s_{\text{ind}}^i \) be a strategy for the \( i \)th player defined with respect to \( s \) as follows: \( \forall \sigma, s', q_0, s_{\text{ind}}^i(h(\sigma, s', q_0)) \equiv s_i(h(\sigma, s, q_0)) \).

The indifference technique. Given a strategy vector \( s \), construct a strategy vector \( s_{\text{ind}}^i \) by replacing each strategy \( s_i \) with the strategy \( s_{\text{ind}}^i \).

As the indifference technique complies with the non-malleability principle, once it is applied, each player faces an induced online problem. The special structure of product metrical task games yields an interesting structure of the induced problems - each can be viewed as a metrical task system with configuration set \( S_i \) in which the distance function and allowable tasks can vary over time (but there are no external configuration changes). Formally, for player \( i \), each combination of the values of the \( N - 1 \) coordinates in a configuration corresponding to the players different from \( i \) induces a distance function and a set of allowable tasks for player \( i \). As the other players change some coordinates of the configuration, the distance function and the allowable tasks for the \( i \)th player are changed. This description of induced metrical task systems might appear a bit strange, but it turns out that certain competitive algorithms for known metrical task systems can be adapted to these distance function and allowable tasks changes. In such cases, the indifference technique yields (approximate) equilibria.

2.6 The Caching Game

In the caching game there are \( N \) players, each representing a server in a network given by an undirected edge-weighted graph \( G = (V, E) \) (\( |V| = N \)). Given is a set \( F \) of \( m \) different files that are assumed to have unit size. Server \( i \) has a cache of capacity \( k_i \) which is initially empty. When a server gets a request for a file it should serve the request by accessing the file. The server may cache the file in its own cache before serving the request. We assume that caching a file incurs some constant cost \( D \), while accessing a remote replica costs the accessing server exactly the distance between the server and the accessed replica. If a requested file is not cached by any server in the network, it must be cached by the server that serves the request.

Lemma 2.11 For every \( i \) such that \( k_i = \infty \), strategy vector \( s_{-i} \), and deterministic strategy \( A, R_{s_{-i}}(A) \geq 2 \).

Proof. Consider a sequence of repeated requests to the same file by player \( i \). Assume that initially the file is not cached in server \( i \), but is cached in server \( j \) whose distance from server \( i \) is exactly 1. Then, for the \( i \)th server the problem is reduced to the ski-rental problem with buying price of \( D \), and the theorem follows.
Lemma 2.12 For every $i$ such that $k_i < \infty$, strategy vector $s_{-i}$, and deterministic strategy $A$, $R_{s_{-i}}(A) \geq k_i$.

Proof. Consider a sequence of requests, all designated to server $i$, and all for files that are initially not cached by any server. Then, for the $i$th server, the problem is reduced to the paging problem with cache size of $k_i$ and the theorem follows from the lower bound on the competitiveness of deterministic online algorithms for the paging problem.

We now introduce a deterministic strategy which is $O(k_i)$-competitive for $k_i < \infty$ with respect to the constructed strategy vector and 2-competitive for $k_i = \infty$ with respect to any strategy vector. It is not hard to show that the caching game is a product game, allowing us to use the indifference technique. It suffices to provide a competitive algorithm for the induced online problem of every player, as follows. A server gets a sequence of requests for files. At each step, the requested file can be first cached by the server incurring a cost of $D$, subject to the cache size constraint. Then, the request is served by either accessing a locally cached copy (if such copy exists), incurring no cost, or accessing a remote replica, incurring a cost that is given as part of the request. Note that if the file is not cached by any server, the remote accessing cost can be set to $\infty$. Note that the same file may be requested with different remote accessing costs.

Observe that the above induced problem generalizes both the online paging problem and the ski-rental problem. Thus, we combine the marking principle borrowed from the online paging algorithms with the rent-or-buy principle to obtain the following algorithm, which we refer to as algorithm $\text{COUNT}'$.

In algorithm $\text{COUNT}'$, the server maintains a counter for each file in $F$. The counter is initialized to 0. We denote the value of the counter of file $f$ by $v(f)$. When a request for file $f$ arrives, the server increases $v(f)$ by the remote accessing cost indicated along with the request. If the requested file is in the cache, the server accesses it directly, incurring no cost. Otherwise, if $v(f) < D$, the file is accessed remotely, while if $v(f) \geq D$, $f$ is to be cached by the server. If the cache is full, then the page with the lowest counter value among the cached files is to be evicted. If the value of the counter of the evicted page is $\geq D$, then the counters of all the files in the cache (including both $f$ and the evicted file) are reset to 0.

Theorem 2.13 In the induced online problem, algorithm $\text{COUNT}'$ has a competitive ratio of at most $4k_i + 6$ for $k_i < \infty$, and at most 2 for $k_i = \infty$.

Proof. For $k_i = \infty$, there is no eviction of pages, and the problem reduces to the ski-rental problem. The theorem follows as algorithm $\text{COUNT}'$ reduces to the 2-competitive algorithm for the ski-rental problem.

Now assume that $k_i < \infty$. Fix a request sequence $\sigma$, and divide it into phases. The first phase begins at the first request. A phase ends right after a request that caused algorithm $\text{COUNT}'$ to reset the counters of the files in the cache. Let $\ell$ be the number of phases and denote by $F_j$ the set of files such that the value of their counters has exceeded $D$ during phase $j$ (notice that the counter of these files are reset before the beginning of the $(j+1)$st phase). Let $v_{\text{final}}(f)$ be the value of the counter of file $f$ after $\text{COUNT}'$ finished to serve $\sigma$. Then, by the definition of $\text{COUNT}'$, $\text{COUNT}'(\sigma) \leq \sum_{j=1}^{\ell} 2D|F_j| + \sum_{f \in F_\ell} v_{\text{final}}(f) = 2D(\ell - 1)(k_i + 1) + 2D|F_\ell| + \sum_{f \notin F_\ell} v_{\text{final}}(f)$. Since $OPT$ must pay at least $v_{\text{final}}(f)$ for serving the requests for every file $f \notin F_\ell$, and at least $D$ for every file $f \in F_\ell$, we get that $2D|F_\ell| + \sum_{f \notin F_\ell} v_{\text{final}}(f) \leq 2OPT(\sigma)$.
is a charging file, and the total remote accessing cost for these requests is at least 
\OPT, of which between the beginning of the

file

is empty. Upon arrival of request

\( r \)

construct

\( \sigma \)

file

that there exists some

\( i \)

respect to deterministic strategies. To achieve competitive strategies in the caching game for the case

for file

is not in \OPT’s cache at the beginning of the

phase of which

is a charging file. For every

\( f \)

is a charging file, and the total remote accessing cost for these requests is at least \( D \) while serving the requests for file

while serving the requests for file

is not in \OPT’s cache at the beginning of the

phase of which

is a charging file, and the total remote accessing cost for these requests is at least \( D \). It follows that

\OPT(\sigma) \geq \sum_{f \in F} D \cdot \frac{h(f)}{2} = \frac{D}{2} \cdot \sum_{f \in F} h(f) \geq \frac{D}{2} \cdot (\ell - 1). \]

If no server has a cache capacity constraint, then \((\text{COUNT}^1, \ldots, \text{COUNT}^\ell)\) is an equilibrium with respect to deterministic strategies. To achieve competitive strategies in the caching game for the case that there exists some

\( i \)

for which

\( k_i < \infty \), we further apply the indifference technique. Then, letting

\( s = (\text{COUNT}^{ind}_1, \ldots, \text{COUNT}^{ind}_\ell) \), we have that

\( R_{s,1}(\text{COUNT}^{ind}) \leq 4k_i + 6 \) for

\( k_i < \infty \) and

\( R_{s,1}(\text{COUNT}^{ind}) \leq 2 \) for

\( k_i = \infty \). Hence, \( s \) is a \( 10 \)-equilibrium with respect to deterministic strategies.

### 2.7 The Online Generalized Steiner Game

In the Online Generalized Steiner Game \( N \) players are given an undirected graph \( G = (V, E) \) with an associated edge cost function \( c : E \rightarrow \mathbb{R}^+ \). A request sequence \( \sigma \) arrives online. Each request in \( \sigma \) is of the form \( (a, b) \), where \( a, b \in V \), and it is allocated to one of the \( N \) players. During the game, the players construct \( F \), a subgraph of \( G \), whose edge set is referred to as the solution of the game. Initially, the solution is empty. Upon arrival of request \( r_i = (a, b) \), player \( p(r_i) \) serves the request by adding edges to the solution so that vertices

\( a \) and \( b \) are connected in \( F \). When a player adds edges to the solution, it pays their cost. Accordingly, the goal of each of the players is to minimize the cost it pays during the game.

As a first step towards constructing an equilibrium of this game, consider some known algorithms for the online problem. A greedy algorithm for the on-line generalized Steiner tree problem is \( O(\log^2 n) \)-competitive [9]. In [11], Berman and Coulston presented an \( O(\log n) \) competitive on-line algorithm (denote it by \( BC \)), matching the \( \Omega(\log n) \) lower bound of Imaze and Waxman [49].

The following results point out the inefficiency in using those algorithms to construct an equilibrium. Specifically, we construct examples in which the adversary can “manipulate” the other players, resulting poor performance by the players.

**Lemma 2.14** The greedy algorithm is not self-competitive.

**Proof.** Consider the following graph, \( G = (V, E) \):

\[
V = \{v_0, v_1, v_2\} \cup \{u_i|1 \leq i \leq k\} \cup \{w_i|1 \leq i \leq k\}
\]

\[
E = \{(v_0, v_1), (v_1, v_2)\} \cup \{(v_1, u_i)|1 \leq i \leq k\} \cup \{(u_i, w_i)|1 \leq i \leq k\} \cup \{(w_i, v_2)|1 \leq i \leq k\},
\]

29
where the weight of the edges \((u_i, w_i)\) is 5 and the weight of all the other edges are 2. Now, consider the following request sequence and the cost for the first player:

<table>
<thead>
<tr>
<th></th>
<th>Greedy</th>
<th>OPT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>edges added</td>
<td>Greedy’s cost</td>
</tr>
<tr>
<td>(v_0, v_1)</td>
<td>((v_0, v_1))</td>
<td>2</td>
</tr>
<tr>
<td>(u_1, w_1)</td>
<td>((u_1, w_1))</td>
<td>-</td>
</tr>
<tr>
<td>(v_1, u_1)</td>
<td>((v_1, u_1))</td>
<td>2</td>
</tr>
<tr>
<td>(u_2, w_2)</td>
<td>((u_2, w_2))</td>
<td>-</td>
</tr>
<tr>
<td>(v_1, u_2)</td>
<td>((v_1, u_2))</td>
<td>2</td>
</tr>
<tr>
<td>(\vdots)</td>
<td>(\vdots)</td>
<td>(\vdots)</td>
</tr>
<tr>
<td>(u_k, w_k)</td>
<td>((u_k, w_k))</td>
<td>-</td>
</tr>
<tr>
<td>(v_1, u_k)</td>
<td>((v_1, u_k))</td>
<td>2</td>
</tr>
</tbody>
</table>

The total cost for a greedy player is hence \(2(k + 1)\), while the total cost for the optimal player is 4, resulting a competitive ratio of at least \(\Omega(n)\).

**Lemma 2.15** Algorithm \(BC\) is not self-competitive.

**Proof.** The straight forward implementation of \(BC\) as a strategy implies that a player may buy many edges besides the edges that are needed to connect a given pair. Hence an example in which such implementation is inefficient can be easily constructed:

\[
V = \{v_0\} \cup \{u_i|1 \leq i \leq k\} \cup \{w_i|1 \leq i \leq k\}
\]

\[
E = \{(v_1, u_i)|1 \leq i \leq k\} \cup \{(u_i, w_i)|1 \leq i \leq k\},
\]

where the weight of the all the edges is 1 and the request sequence is:

<table>
<thead>
<tr>
<th></th>
<th>(r_i)</th>
<th>(p(r_i))</th>
<th>Edges added</th>
</tr>
</thead>
<tbody>
<tr>
<td>(u_1, w_1)</td>
<td>2</td>
<td>((u_1, w_1))</td>
<td></td>
</tr>
<tr>
<td>(u_2, w_2)</td>
<td>2</td>
<td>((u_2, w_2))</td>
<td></td>
</tr>
<tr>
<td>(\vdots)</td>
<td>(\vdots)</td>
<td>(\vdots)</td>
<td>(\vdots)</td>
</tr>
<tr>
<td>(u_{k-1}, w_{k-1})</td>
<td>2</td>
<td>((u_{k-1}, w_{k-1}))</td>
<td></td>
</tr>
<tr>
<td>(v_1, u_k)</td>
<td>1</td>
<td>((v_1, u_k)) and (BC) also adds the edges ((v_1, u_i)) for (1 \leq i &lt; k)</td>
<td></td>
</tr>
</tbody>
</table>

The cost for a \(BC\) player is hence \(k\), while the cost for the optimal player is 1, resulting a competitive ratio of at least \(\Omega(n)\).

Another implementation of \(BC\) might maintain a separate set of connected components for each player. Such a strategy is inefficient as can be seen by a simple modification to the example given in the previous lemma, in which instead of assigning all the \((u_i, w_i)\) requests to a single player (player 2), each request is designated to a different (new) player. The rest of the analysis is the same by the definition of \(BC\). ■

Despite the above examples, in what follows we show that \(BC’\), a slight modified version of \(BC\), is a competitive algorithm for the players’ induced problems in the game. It is not hard to show that this game
is a product game, and hence the indifference technique can be applied to BC' to obtain an \(O(\log n)\)-self-competitive strategy. We note that we do not know whether a similar result can be obtained using the greedy approach.

### 2.7.1 Algorithm BC'

Notice that the induced problem is identical to the online generalized Steiner problem with the exception that before each request arrives, some edge costs can get nullified. In what follows, we assume that each edge having zero cost is added to the solution immediately. We denote by \(G^i\) the graph \(G\) with the edge costs upon arrival of the \(i\)th request.

In what follows we describe the differences between BC' and BC as it appears in [11][Sect. 2]. The distance between two vertices in the graph is calculated as if the edges that were chosen to the solution have zero cost. We will denote the distance between vertices \(a\) and \(b\) just before serving the \(i\)th request by \(d^i(a, b)\). If algorithm BC' decides in step 3.4 (respectively, step 4.4) to add a path between \(u\) and \(v\) to the solution, then it adds to the solution all the edges that have not yet been added to the solution among those of some shortest path between \(u\) and \(v\). Algorithm BC' maintains a set of connected components denoted by \(C\), which is initially empty. During steps 3 and 4 of the algorithm, BC' examines only the connected components in \(C\). Given a request \(r_i = (a, b)\) it adds two new singletons, \(\{a\}\), \(\{b\}\), to \(C\). If, in step 3.4 (respectively, step 4.4), BC' adds to the solution a path \(P\), then it updates \(C\) by merging the connected components of all the vertices in \(P\) into one new connected component that also includes all the vertices of the path \(P\) that did not belong to some connected component in \(C\).

### 2.7.2 Analysis of Algorithm BC'

Fix a request sequence \(\sigma\). We denote by \(OPT\) the set of edges of an optimal solution. In what follows we derive a lower bound for the cost of \(OPT\), using similar ideas to those of [11]. Let \(B^i(v, r)\) denote the open ball in \(G^i\) with center at \(v\) and radius \(r\) (we abuse the notation and denote the set of vertices of the ball and the set of edges of the ball by the same notation).

**Lemma 2.16** Let \(r_i = (a, b)\) be a request such that \(d^i(a, b) \geq r\). Then, the cost of \(OPT \cap B^i(a, r)\) is at least \(r\).

**Proof.** In order to serve request \(r_i\), \(OPT\) must (at least) pay for a path between \(a\) and \(b\) in \(G^i\), and the lemma follows from the definitions of \(d^i(a, b)\) and \(G^i\). ■

A ball \(B^i(v, r)\) that satisfies the conditions of Lemma 2.16 will be referred to as a lower bound ball (this term was used in [11] to describe the analogue idea). It follows from Lemma 2.16 that a collection of pairwise disjoint lower bound balls \(\{B^i(v_k, r_k)\}_{k}\), comprises a lower bound for \(OPT\) (this lower bound is the analogue of the lower bound collection presented by [11]).

Similarly to [11], we will run a shadow algorithm that will find a set of lower bounds for \(OPT\). One of these lower bounds will be sufficient to prove the competitive ratio. For clarity, we use the notation of [11]. For example, \(C(k)\) will be a collection of lower bound balls with radius at most \(2^k\).
The shadow algorithm starts with all $C(k)$'s empty. Upon receiving a request $r_i = (a, b)$ it adds to every $C(j)$ with $j \leq \lfloor \log_2 d^i(a, b) \rfloor$ the largest ball $B^i(a, r)$ such that $r \leq 2^j$ and $B^i(a, r)$ does not intersect any of the previous balls of $C(j)$. The same is performed for $b$.

**Lemma 2.17** $\Sigma \geq BC'({\sigma})$.

**Proof.** We perform an amortized analysis. We create an account for every $(A, j)$ such that $A \in C$ and $j \leq \text{class}(A)$. The rest of the analysis is the same as in [11]. Particularly:

- In the first step $\Delta_{cost} = \Delta_{\Sigma}$;
- If while processing a component $X \in C$ in step 3 (resp. 4) it is the case that $d \geq 2^{m+1}$, then there is no pair of intersecting balls such that one is centered at $a$ (resp. $b$) and the other is centered at $v \in X$ (pay attention that for $k > i$ it holds that for every $a, b \in V$, $d^i(a, b) \geq d^k(a, b)$ and thus the same proof is valid);
- If while processing a component $X \in C$ in step 3 (resp. 4) it is the case that $d < 2^{m+1}$, then $\Delta_{cost} \leq d - 2^{m+1}$ (exactly same considerations) and $\Delta_{\Sigma} \geq d - 2^{m+1}$ (note again, that same considerations hold since $k > i$ it holds that for every $a, b \in V$, $d^i(a, b) \geq d^k(a, b)$).

**Lemma 2.18** $\max_k \Sigma(k) \geq \frac{1}{\log_2 n + 2} \Sigma$.

**Proof.** By following the same considerations of the proof in the analysis of [11], that is, removing some lower bound collections at the expense of liquidating the remaining accounts, leaving only $O(\log n)$ collections.

The above analysis implies that the individual efficiency of the strategy vector $((BC')^{ind_1}, \ldots, (BC')^{ind_n})$ is $O(1)$ away from the best possible. As for the overall efficiency, note that the lower bound on the price of an optimal solution for the induced online problem, that was used to prove the competitive ratio of $BC'$, is also a lower bound for the price of an optimal solution for the “social” online problem. Hence, the strategy vector $((BC')^{ind_1}, \ldots, (BC')^{ind_n})$ has a competitive ratio of $O(N \log n)$, which is $O(N)$ away from the known lower bound.

### 2.8 Conclusions and Future Research

In this chapter we have introduced the model of online games, with the aim of studying scenarios that combine online decision making with interaction between independent agents. As this is the first work in this line of research, we consider it a pioneering work that establishes the foundation for future research.

Our work has concentrated on equilibria-related issues. A natural question to be considered is what are the limitations of the techniques we have developed? These techniques seem to be ineffective for some profit maximization games, such as the game that generalizes the online maximum throughput problem: as soon as a player deviates from a certain strategy, the set of feasible actions for the other players strictly changes. Moreover, even for the problems we have studied above, it is not clear whether the techniques can
be extended and applied to randomized strategies. A negative answer to some of the above questions will strengthen the motivation for the development of other techniques for constructing equilibria. Of special interest is a general technique for the randomized case, as many online problems have randomized online algorithms that outperform the deterministic ones.

A different direction for future research is obtaining negative results of two kinds. The first kind is a lower bound on the competitiveness of strategies which does not follow from a lower bound on an online problem in a straightforward manner, but would rather exploit the nature of interaction between players. A different kind of negative results might deal with the existence of (approximate) equilibria. On a different note, Stackelberg equilibrium is a notion of equilibrium in asymmetric scenarios, in which some agent can be identified as the leader while another agent the follower. It would be interesting to generalize the model of online game to such scenarios and study the appropriate equilibrium. Another interesting research directions are the price of stability of online games (see Section 2.2), as well as applying other well-known concepts in algorithmic game theory to the model of online games.

We conclude with a question that we consider to be a by-product of our study. The techniques developed in this chapter suggest that an online game induces new online problems having a different flavor than classical online problems. Essentially, in the new problems, in between two requests to an agent, some modifications to the system may occur. Besides the fact that algorithms for these problems may define an equilibrium to the online game, such problems may be of independent interest as they capture certain phenomena that has not been studied so far.
Chapter 3

Convergence to Equilibria in Interdomain Backup Routing

In this chapter we study the convergence of best response dynamics in an interdomain routing game. The motivation for this routing game stems from the Border Gateway Protocol (BGP) [67] which establishes routes between the Autonomous Systems (ASes) that make up today’s Internet. The convergence of best response dynamics in the routing game that corresponds to BGP is referred to as BGP safety. BGP safety is of high importance, since in the resulted equilibrium the routing outcome is stable. We propose policy guidelines that allow backup routing and are naturally induced by the business relationships between ASes (generalizing the celebrated Gao-Rexford guidelines [32]). We prove that, if ASes conform to these simple and natural guidelines, BGP safety is guaranteed. The sharp contrast between our positive result and previous results is enabled by our novel proof techniques, that rely on the substitution of the traditional worst-case analysis of BGP dynamics with a more realistic probabilistic approach.

3.1 Introduction

3.1.1 Is Backup Interdomain Routing Safe?

The Border Gateway Protocol (BGP) [67] handles interdomain routing in today’s Internet. Interdomain routing is the task of establishing routes between the administrative domains that make up the Internet, called Autonomous Systems (ASes). BGP establishes routes between ASes hop by hop, as knowledge about the topology of the network and the actions of the different ASes propagates through the network. Informally, BGP requires ASes to continuously choose their most preferred routes (given their local preferences), out of the most recently announced routes learned from neighboring ASes. This route selection process enables ASes to express diverse local routing policies without global coordination. However, as observed in [75], this BGP feature may come at a costly price: the lack of global coordination between local routing policies may result in undesirable routing anomalies, in the form of persistent route oscillations. Hence, researchers seek constraints that guarantee BGP convergence to a “stable” routing outcome – a property known as “BGP safety” [40, 39].
In today’s commercial Internet, ASes are bound by long-term contracts that determine who provides connectivity to whom [46]. A pair of ASes can have one of two possible types of business relationships: customer-provider, in which one AS purchases connectivity from another, and peering, in which two ASes agree to carry transit traffic to and from each other’s customers, for free. In a landmark paper, Gao and Rexford [32] present a result that helps account for the (observed) stability of the Internet in its current form; Gao and Rexford propose three policy guidelines that are naturally induced by the business relationships between ASes. These guidelines imply constraints on the network topology and on ASes’ routing policies, that are shown by Gao and Rexford to guarantee BGP safety. Hence, BGP safety can be seen as following from the inherently economic nature of the Internet.

The result in [32] was an important step towards explaining the Internet’s relative resilience to persistent instability. However, in practice, interdomain routing is also affected by the existence of “backup links” [18, 37, 31]: it is common for ASes to place a relative preference on links between themselves and other ASes; some connections are regarded as preferred, or “primary”, while others are less preferred, or “backup” connections. The intent is typically that the backup connections will be used for carrying traffic only in case of failures in the primary connections. Backup links can exist between customers and their providers (some links to upstream providers may be designated as backups), or between peers [31]. Unfortunately, backup routing can lead to violation of guidelines in [32], and so BGP safety is no longer assured.

Gao et al. [31] explored the implications on BGP safety of the introduction of backup links to the Gao-Rexford commercial framework. It is shown in [31] that ASes’ will to increase the reliability of the network under link and router failures, via backup links, comes at the possible expense of BGP safety. Moreover, while this lack of safety can be rectified, this requires rather extreme means. In particular, if a route is marked as a backup route, then it must retain this marking as it traverses subsequent ASes, and ASes’ rankings of backup routes must be severely constrained. Implementation of the schemes in [31] will be difficult to realize (given, amongst other things, the current low-level nature of most router configuration languages, and the serious limitations imposed on routing policies). These bleak conclusions leave us with little hope of achieving safety in backup interdomain routing with BGP without resorting to drastic measures like significantly limiting the expressiveness of ASes, deploying costly machinery, or both. Is the common practice of backup interdomain routing truly unsafe?

3.1.2 Our contribution

Our contribution in this chapter is twofold:

- **Guidelines for safe backup interdomain routing.** Our first contribution is a surprising positive result: we propose simple policy guidelines, that are an extension of the Gao-Rexford conditions to the backup routing setting, and show that, if ASes adhere to these natural guidelines, then BGP is guaranteed to converge to a stable routing outcome. Our policy guidelines are consistent with the current business relationships between ASes. The guidelines can be locally implemented by individual ASes without unduly limiting the expressiveness of ASes’ routing policies, or requiring any changes to the protocol (neither in the data plane nor in the control plane). Hence, we show that it is possible for individual ASes, or pairs of ASes, to locally use backup routing, so as to increase the reliability of interdomain routing, without putting in peril the global stability of the Internet. This result is robust,
in the sense that it is independent of the network topology and holds even in the presence of topology changes (e.g., due to link and node malfunctions).

- **Allowing ASes greater flexibility without risking BGP oscillations.** The sharp contrast between our positive result for backup routing and the results in [31] is enabled by our novel proof techniques: we substitute the worst-case model, traditionally used to analyze BGP dynamics, with a more realistic probabilistic model (similar to that in [78]). As observed by Viswanathan et al., [78], this eliminates the possibility of pathological BGP oscillations, that cannot occur in practice (and so no machinery for handling such oscillations is necessary). We present methods for proving (probabilistic) BGP safety results in this model, and use these methods for obtaining (probabilistic) BGP safety results for several other classes of routing policies. We believe that our general approach to the analysis of BGP dynamics, and our specific proof techniques, are of independent interest. Indeed, we demonstrate, via counter examples, that many of the restrictions on routing policies required in previous works in order to guarantee BGP safety, are, in reality, a byproduct of the worst-case analysis used, and not inherent to BGP dynamics. Our (probabilistic) BGP safety results are strong evidence of this fact. The above implies that network operators can have much more freedom in defining local routing policies, as, in practice, more expressive routing policies than those considered in previous works are globally safe.

### 3.1.3 Policy Guidelines for Backup Routing

We propose the policy guidelines listed below. Our guidelines are a generalization of the celebrated Gao-Rexford guidelines to the case of backup routing. Specifically, we maintain the topology and export conditions presented in [32], and replace the preference condition in [32] by two natural conditions on ASes’ preferences.

- **Topology condition: no customer-provider cycles.** No AS is its own indirect provider; that is, the AS-level topology does not contain any cycle of provider-customer edges.

- **Export condition: provide transit services to customers only.** An AS should export routes through any neighbor to its customers, but should only export routes through its customers to its peers and providers.

- **Preference condition I: prefer primary routes to backup routes.** Backup connections are normally designated to be used only in the case of lack of reachability by other means. Hence, when selecting a route for a destination, an AS should prefer routes through neighbors to whom it is connected by a primary link (“primary routes”), over routes through neighbors to whom it is connected to by a backup link (“backup routes”).

  (We note that, in practice, it could be that an AS sometimes prefers backup routes through customer neighbors to primary routes through peer or provider neighbors. However, such violations of our preference condition are easy to handle, as we shall explain in Section 3.5, and so our results extend to such cases as well.)

- **Preference condition II: Prefer customer routes to peer/provider routes (subject to preference condition I).** When faced with a choice between two (or more) primary routes, an AS should prefer
a (revenue-generating) route through a customer over routes through a peer or provider. Similarly, an AS should prefer backup routes through customer neighbors over backup routes through peers or providers.

3.1.4 Related Work

Our work continues a long line of research on BGP convergence (see [25, 26, 31, 32, 38, 39, 40, 50, 68, 71] and references therein). Varadhan et al. [75] were the first to observe that BGP policy interaction may lead to persistent route oscillations. Griffin et al. [40, 39] put forth a formal model for the analysis of BGP dynamics, and provided sufficient conditions for BGP safety. Subsequently, Gao and Rexford [32] showed that, if each AS’s policy conforms to restrictions based on its business relationships with other ASes, and the global structure of business relationships consists of a customer-provider hierarchy with peering, then BGP safety is guaranteed.

Viswanathan et al. [78] were the first to propose a probabilistic model of BGP dynamics. Like [78], we take a probabilistic approach to analyzing BGP convergence. However, our work greatly differs from that of [78] in that we are interested in exploring the space of concrete routing policies for which (probabilistic) BGP safety is attainable. In contrast, the main focus of [78] is on abstract mathematical properties of their probabilistic convergence model. In particular, [78] does not consider backup interdomain routing, or any other concrete examples of routing policies (like the examples presented in Section 3.6).

Finally, Gao et al. [31] explored the implications of backup routing on the Gao-Rexford commercial framework. The analysis of BGP dynamics in [31] was carried out within the worst-case model and so their results were mainly of a negative flavor; basically, it is shown in [31] that to avoid persistent route oscillations one must resort to fairly drastic measures, and that, in particular, this must come at the expense of severely restricting the expressiveness of ASes’ routing policies. We show that when moving from the worst-case model to the probabilistic one, it is possible to bypass these difficulties and achieve BGP safety without paying such costly prices.

3.1.5 Organization of the Chapter

In Section 3.2 we motivate our approach and present our formal model for the probabilistic analysis of interdomain routing dynamics. In Section 3.3 we prove that if our policy guidelines hold then a stable routing outcome always exists in the network. We strengthen this result in Section 3.4, by showing that if our policy guidelines hold then BGP is guaranteed to converge to a stable routing outcome. We present some additional merits of our (probabilistic) safety result in Section 3.5. In Section 3.6 we discuss the applicability of the probabilistic approach to BGP dynamics and exhibit several examples (other than backup interdomain routing) of routing policies for which (probabilistic) BGP safety is guaranteed. We conclude in Section 3.7.

3.2 The Model

In this section we formally present (Subsection 3.2.2) our probabilistic model, based on the probabilistic model for analyzing BGP dynamics presented by Viswanathan et al. [78]. To motivate our approach, we first
present (Subsection 3.2.1) an informal example of the way in which backup routing can induce persistent route oscillations, and why one might hope that such routing anomalies will not occur in practice.

### 3.2.1 Illustrative Example and Some Background

**Backup routing can complicate things.** Consider a network in which the nodes represent ASes, and the edges represent BGP connections. Every pair of neighboring ASes (i.e., nodes connected by an edge) either have a customer-provider relationship, or are peers. The Gao-Rexford guidelines [32] can be stated as follows:

- **Topology condition: no customer-provider cycles.** Consider the digraph with the same set of nodes as our network and with a directed edge from every customer to its (neighboring) provider. We demand that there be no directed cycles in this graph.

- **Export condition: provide transit services to customers only.** A *customer route* is a route in which the next-hop AS is a customer. *Provider and peer routes* are defined similarly. Nodes should share only customer routes with their providers and peers but can share all of their routes with their customers.

- **Preference condition: prefer customers to peers and providers.** Nodes are required to always prefer customer routes over peer and provider routes.

![Figure 3.1: BGP oscillations with backup routing (based on DISAGREE in [40, 39]).](image)

It is known [32] that if all three conditions hold for a network of ASes, then BGP safety is guaranteed. However, backup routing complicates matters [31]: consider the simple network in Figure 3.1 (inspired by DISAGREE in [40, 39]). AS $d$ is a customer of ASes 1 and 2, and 1 is a customer of 2. ASes 1 and 2 wish to send traffic to $d$. AS 2 prefers the customer route $21d$ to the customer route $2d$. $(1, d)$ is a backup link, and

---

1As is common in such papers [31, 32, 39, 40], in our formulation nodes represent ASes rather than individual routers. Our results can be extended to the more general case of nodes representing routers (with the proper adjustments).
both 1 and d agree not to use this link unless forced to (due to the unavailability of other routes). Therefore, if both route 12d and 1d are available to AS 1 at some point in time, AS 1 should prefer 12d over 1d. Observe that this violates the Gao-Rexford preference condition (as 2 is 1’s provider and d is 1’s customer). Hence, the BGP safety result in [32] no longer holds. Are persistent BGP route oscillations possible with backup routing?

**Unreasonable BGP oscillations.** The Border Gateway Protocol (BGP) belongs to the family of path-vector protocols, the abstract properties of which were studied in [39]. In such protocols, communication between nodes takes place through update messages that announce chosen routes. The process is initialized when d announces itself to its neighbors by sending update messages. Then, each source node i iteratively establishes routes to d by: (1) importing (via update messages) routes to d chosen by its neighbors; (2) choosing the best *available* route from i to d (given its local preferences); and, (3) if there is a change to i’s chosen route, exporting the newly selected route to i’s neighbors via update messages.

Let us now revisit the example in Figure 3.1. The following simple scenario is known to induce a BGP oscillation for our example [40, 39]: suppose that d announces itself to 1 and 2 *simultaneously*. 1 and 2, for lack of other choices, will both choose their direct routes to d, and inform each other of these routes via update messages. Suppose both update messages arrive at their destinations at *precisely* the same time. In this case, 1 will learn of 12d (its most preferred route) and switch to it at the exact same time 2 switches to 21d. What happens if 1 and 2, once again, learn of each other’s routes at the *same* time? If this happens, they will both be forced to revert to their direct routes to d (as their previous routes are no longer available). Observe that if we repeat this timing of update messages *indefinitely* this will result in a persistent route oscillation.

The BGP oscillation described above cannot occur in practice as we cannot expect the same timing of update messages to repeat itself *ad infinitum*. In fact, it is easy to see [78] that any deviation (no matter how small) from this highly unlikely timing of update messages is guaranteed to lead to a “stable” routing outcome. This gives us hope that, while backup routing is susceptible to BGP oscillations, these might be pathological oscillations that cannot happen in practice. Unfortunately, in general, the violation of the Gao-Rexford preference condition might result in networks for which no “stable” routing state exists (like BAD-GADGET in [40, 39]). In such networks there is obviously no hope for BGP convergence as there is nothing for BGP to converge to. In the following sections we shall show that if ASes adhere to simple policy guidelines, then, not only does a “stable” routing outcome always exist, but BGP is always guaranteed to converge to such an outcome.

### 3.2.2 A Stochastic Model of Interdomain Routing

**The network and routing policies.** The network is represented by an *AS graph* [40, 39] $G$, in which the set of vertices (or nodes) $V$ represents ASes, and the the set of edges (or links) $E$ represents BGP connections. $V$ consists of $n$ *source nodes* $1, ..., n$ and a *unique destination node* $d$. Each source node $i$ has a ranking function $\preceq_i$ that specifies its preferences over all simple routes (*i.e.*, paths containing no loops) between itself and $d$.

---

2This formulation is reasonable as interdomain routing is handled by BGP independently for each destination AS.
The routing policy of each node $i$ consists of $\leq_i$ and of $i$'s import policy and export policy. $i$'s import policy dictates which set of routes $Im(i) \subseteq P_i$ $i$ is willing to send traffic along. Let $\emptyset$ denote the possibility of not being assigned a route. We assume that $\emptyset <_i P_i$ for any route $P_i \in Im(i)$ ($i$ prefers any route in $Im(i)$ to not getting a route at all) and that $P'_i <_i \emptyset$ for any route $P'_i \notin Im(i)$ ($i$ will not send traffic at all rather than send traffic along a route not in $Im(i)$). $i$'s export policy dictates which set of routes $Ex(i, j) \subseteq P_i$ $i$ is willing to announce to each neighbor $j$.

**Remark 3.1** The export and import policies of nodes can be embedded in their ranking functions (as explained above for the import policy). Hence, for ease of exposition, we shall disregard these in our formal definitions.

**A stochastic execution model.** In the model of Griffin et al. [40, 39], ASes can be activated in different timings, and update messages can be arbitrarily delayed. Each time a node $i$ is activated it is required to perform the following actions:

- Importing (via update messages) routes to $d$ chosen by $i$’s neighbors (based on $i$’s import policy).
- Choose the best available route from $i$ to $d$, given $i$’s ranking function ($i$ will only change current its route if a strictly better route is available).
- If there is a change to $i$’s chosen route, export the newly selected route to $i$’s neighbors via update messages.

In [39], BGP is required to converge for any possible timing of AS activations and update messages arrival (no matter how unlikely). This worst-case analysis is equivalent to assuming that the timing of update messages is chosen by some adversarial entity [59]. We consider a probabilistic version of the model in [40, 39]. In our model, there is an infinite sequence of discrete time-steps $t = 1, 2, \ldots$. For each node $i$ there is some fixed value $p_i \in (0, 1)$ such that node $i$ is activated in each time step with probability $p_i$. For every pair of neighboring nodes, $i$ and $j$, there is some fixed value $p_{ij} \in (0, 1)$ such that an update message sent from $i$ to $j$ arrives at $j$ with probability $p_{ij}$ in every time step following its transmission time (that is, the probability that it reached $j$ immediately is $p_{ij}$. If it does not arrive at $j$ immediately, the probability that it arrives in the next time-step is $p_{ij}$, and so on).

**Remark 3.2** We note that our results continue to hold for many similar alternate formulations of the stochastic execution model. Informally, the only property of the model needed for our proofs is the prevention of pathological timings of ASes’ activations (timings that happen with probability zero).

**Stable states.** BGP is required to converge to a stable routing outcome. Informally, a stable state is a global configuration that, once reached, remains unchanged. To define the notion of a stable state (or stable outcome) we require the following two definitions.

**Definition 3.1 (path assignment)** A path assignment is an $n$-tuple of routes in $G$, $P_1, \ldots, P_n$, such that each $P_i$ is a simple path from $i$ to $d$, or the empty path $\emptyset$. 40
Informally, a path assignment is an allocation of routes to the different source nodes. A path assignment is said to be consistent if the routes of nodes that lie on the path of node $i$ are suffixes of node $i$’s path:

**Definition 3.2 (consistent path assignment)** A path assignment $P_1, ..., P_n$ is said to be consistent if for every two nodes $i \neq j$ it holds that if $j$ is on $P_i$, then it must hold that $P_j \subset P_i$ (that is, $P_j$ is a suffix of $P_i$).

Now we are ready to define stable states. A state is stable if it is a consistent path assignment, in which every node gets its most preferred available route:

**Definition 3.3 (stable state)** A path assignment $P_1, ..., P_n$ is said to be a stable state if it is consistent and if, for every link $(i, j) \in E$, if $P_i \neq (i, j)P_j$, and $(i, j)P_j$ does not contain a loop, then $(i, j)P_j <_1 P_i$.

It is known [40, 39] that a stable state is always in the form of a tree rooted in $d$.

**Definition 3.4 (probabilistic BGP safety)** [78] BGP is said to be (probabilistically) safe on a given routing instance, if the probability of BGP reaching a stable state (from any initial path assignment) converges to 1.

We refer the reader to [78] for a thorough explanation of probabilistic formulations of routing dynamics.

### 3.3 Existence of Stable States in Backup Interdomain Routing

In this section we take the first step towards proving that backup interdomain routing via BGP is safe – we prove that if our policy guidelines (presented in the Introduction) hold, then there always exists a stable state to which BGP can potentially converge (in Section 3.4 we show that BGP is indeed guaranteed to converge to a stable state). Our proof is constructive, in the sense that it shows how a stable state can be found, by constructing it edge by edge. The high-level idea of our proof is showing that, from the empty path assignment, in which each node is assigned the empty route, there is an activation sequence of ASes that leads to a stable state. That is, it is possible to activate ASes, one by one, in a carefully chosen order, and thus reach a stable state. The key idea is to gradually add backup edges to the routing state, and stabilize the network after each such addition. We prove that the stabilization process is such that it ensures the proper convergence to a stable state.

**Theorem 3.3** For any network topology and ASes’ routing policies that uphold the policy guidelines, it holds that a stable state (possibly more than one) exists.

**Proof.** Let us begin our proof by presenting some terminology that will be of great use to us:

**Coloring nodes.** Let $s$ be some consistent path assignment. We distinguish between three kinds of nodes:

- **Black nodes.** A node that does not have a path to $d$ in $s$ is referred to as a black node.
- **White nodes.** We say that a node with a path to $d$ in $s$ is white if the first edge on that path is a primary edge (i.e., it has a primary route).

• **Gray nodes.** We say that a node with a path to $d$ in $s$ is gray if the first edge on that path is a backup edge (i.e., it has a backup route).

**Observation 3.4 (possible color changes)** When a source node $i$ is activated then the only possible color changes for $i$ and the other nodes in the network are:

- **Node $i$ can only turn from gray to white** (if it switches from a backup route to a primary route), or from black to either gray or white (if it changes status from not having a route to $d$ to having a route to $d$).

- **Other nodes can only turn from gray or white to black** (if $i$ chooses a primary provider or peer edge instead of a customer backup edge, thus making its new route non-exportable to upstream providers or to peers).

**Stabilizing networks.** The basic building block of the activation sequence of ASes which we construct is the stabilization phase. We say that a node wishes to move, or wishes to switch paths, if the path it currently has is not its most preferred available route. In a stabilization phase, we start with a consistent path assignment and iteratively activate, one by one and in any order, any node that wishes to switch to its best route from its current position, as long as it does not change its color from black to gray (i.e., as long as it is not switching from not having a route to $d$ to having a backup route). We make knowledge of nodes newly selected route immediately available to their neighbors (i.e., update messages arrive at their destinations immediately).

**Observation 3.5 (non-gray nodes remain non-gray in the stabilization phase)** If a node is not gray at some point in the stabilization phase, it cannot turn gray from that moment until the end of the stabilization phase.

This is implied by Observation 3.4, and the definition of a stabilization phase.

**Definition 3.5 (stabilization phase convergence)** We say that a stabilization phase starting from some path assignment (state) $s$ converges, if it reaches a state $s'$ in which each node is either assigned its most preferred available route, or can only be assigned its most preferred available route by changing colors from black to gray.

That is, a stabilization phase converges if it reaches a state that is “almost” stable, in the sense that nodes either do not wish to switch routes, or can only do so by moving from no route to $d$ to being assigned a backup route to $d$. We are now ready to prove that the stabilization phase always converges:

**Claim 3.6 (the stabilization phase converges)** A stabilization phase from any initial state $s$ always converges.

**Proof.** By Observation 3.4, and the definition of a stabilization phase, no node can oscillate between being white, black and gray. Hence, starting from some moment in time every node either uses only primary edges or only backup edges. From that moment in time we can think of the network as the network one gets
by removing the unused backup/primary edges of each node. For this network the Gao-Rexford preference condition holds (because we require this from ASes within their set of primary routes and within their set of backup routes). Hence, all Gao-Rexford conditions hold for this network. By the BGP safety result in [32] we now have that any activation of ASes is bound to eventually reach a stable state. This concludes the proof of the claim.

An iterative stabilization process. We start with the empty path assignment (in which each source node is assigned the empty path $\emptyset$). We repeatedly apply the following procedure:

In each step $k = 1, \ldots$

- Stabilize the network (using the stabilization phase procedure explained above). We denote the routing state that the stabilization phase converges to by $T^k$.
- Once the network stabilizes, pick a black node $i$ that has an available backup route to $d$ in $T^k$ (which it could not use before because we did not allow transitions from black to gray). If many such black nodes exist we choose a black node as follows: let $Q_C$ be the set of black nodes that have an available customer backup route to $d$ (in $T^k$). We pick $i_k \in Q_C$ to be a minimal such node, in the sense that there is no other node $j$ in $Q_C$ such that $j$ is an indirect customer of $i$ (i.e., $j$ appears on a customer route from $i$ to $d$). If $Q_C$ is empty (no black node has an available backup customer route), we arbitrarily pick a black node that wants to move.
- Allow $i_k$ to switch to its most preferred available route (thus making it gray). We denote the edge through which $i_k$ connects to $T^k$ upon activation by $e_k = (i_k, j_k)$.

Non-black nodes remain non-black. We shall now prove a crucial property of the stabilization phase: if a black node is connected to $d$ via some route at the beginning of one of the iterations of our stabilization process, it shall never become black again after that moment in time. Moreover, no node that ceases to be black at some point in time will ever revert to being black again. We start with the following two propositions that help us characterizing some possible changes in the routing tree.

**Proposition 3.7** If node $j$ did not have a customer route at the end of the $k$'th stabilization phase, it will not have a customer route that does not pass through $i_k$ during or at the end of the $(k + 1)$'th stabilization phase.

**Proof.** Assume to the contrary that there are some nodes that do not have a customer route at the end of the $k$'th stabilization phase but have a customer route that does not pass through $i_k$ during or at the end of the $(k + 1)$th stabilization phase. Let $j$ with be the closest such node to $d$, and let $P$ be its (new) customer route to $d$. By definition of $j$, all nodes on $P$ had a customer route at the end of the $k$'s stabilization phase. Therefore, $j$ could have used $P$ at the end of the $k$'th stabilization phase – a contradiction. ■

**Proposition 3.8** Let $j$ be a node with a customer route that passes through $i_k$ during or at the end of the $(k + 1)$'th stabilization phase. Then, $j$ was black at the end of the $k$th stabilization phase.
Proof. Assume to the contrary that the proposition is false. Consider a customer route to $i_k$ and let $j$ be the closest node to $i_k$ on that route such that $j$ was not black at the end of the $k$th stabilization phase. Let $e = (j, \ell)$ be the customer edge that $j$ is using on its new customer route. By the choice of $j$, $\ell$ is black at the end of the $k$th stabilization phase. Notice that if $e$ is a backup edge then $j$ was gray at the end of the $k$th stabilization phase, which contradicts the order in which we choose nodes to turn gray ($i_k$ is a (possibly indirect) customer of $j$). So $e$ is a primary edge and hence $\ell$ could have used it to connect via the route $j$ had at the end of the $k$th stabilization phase – a contradiction. ■

Corollary 3.9 Let $j$ be a node with a customer route during or at the end of the $(k + 1)$th stabilization phase. Then, if $j$ was not black at the end of the $k$th stabilization phase, it had a customer route already at the end of the $k$th stabilization phase.

Recall that we wish to prove that if a node is not black (i.e., connected to $d$ via some route) at some point in time, then it shall remain connected to $d$ thereafter. The next two propositions will greatly assist us in proving this:

Proposition 3.10 (customer routes remain fixed) If $i$ is a node that is connected to $d$ via a customer route (backup or primary) at the end of the $k$th stabilization phase, then, $i$’s route to $d$ will remain fixed from that moment forth (i.e., throughout all the subsequent stabilization phases).

Proof. Assume, for point of contradiction, that the statement is false. Let $i$ be the first node that violates it, i.e., changes its customer route. Consider the activation of $i$ in which this violation takes place, and let $(i, j)$ be the new edge picked by $i$ at that time. Let us first handle the case that $j$ had some route to $d$ at the end of the previous stabilization phase. We examine the following two subcases:

• **$i$ is a customer of $j$.** Observe that if $i$ switches from the customer route which it previously had to a provider route this implies (by the guidelines) that $i$ must be moving from a backup connection to a primary connection. Also observe that it must be that $j$ has a customer route to $d$ (because $j$ had a possible customer route via $i$). $i$ was chosen as the first node to change its customer route. Hence, $j$’s customer route remained constant since it was first assigned this route. Therefore, $i$ could have switched to its new route by the end of the previous stabilization phase – a contradiction.

• **$i$ is a peer or provider of $j$.** In this case, $j$ must have a customer route to $d$ (by the export condition). According to our assumption that $j$ was not black at the end of the previous stabilization phase and by Corollary 3.9 it must be that $j$ had a customer route at the end of the previous stabilization phase as well. Hence, $i$ could have connected to it before, and we reach the same contradiction as in the previous subcase.

If $j$ had no route to $d$ at the end of the previous stabilization phase, it follows that $(i, j)$ is a backup edge (because otherwise $j$ could have a route to $d$ that goes through $i$), and so $i$ is gray and was chosen as the $k$th black node for some $k$ (i.e., $i$ was allowed to change from black to gray in the $k$th step). Moreover, $j$’s current customer route to $d$ includes at least one backup edge (otherwise $j$ would have had a route right after the initial stabilization phase). Let $e_{k'} = (i_{k'}, j_{k'})$ be the backup edge on $j$’s path to $d$ that is closest to $d$. Since $i$ has a customer route to $i_{k'}$, and by the choice of $i$, it follows that $k' < k$ (because of our method for
choosing black nodes in case that more than one black node with a backup customer route exists). Hence, \( ik' \) should have been selected and connected to \( d \) via a customer route prior to \( i \). This is true for all backup edges on \( j \)'s route, and so \( j \) should have been connected to \( d \) before \( i \) – a contradiction.

**Proposition 3.11** AS \( ik \) remains connected to \( d \) until the end of the subsequent \((k+1)'th\) stabilization phase.

**Proof.** The only way for \( ik \) to turn black is if a node on its path to \( d \) moves from a backup customer edge to a primary peer/provider edge during the \((k+1)'th\) stabilization phase. Let us consider the different possibilities: According to Proposition 3.7, nodes that did not have a customer route at the end of the \( k'\)th stabilization phase will not have a customer route that does not pass through \( ik \) at the end of the \((k+1)\)th stabilization phase. Hence, such nodes do not provide an alternative route for \( ik \) and so \( ik \) does not move to a path that include such nodes. According to Proposition 3.10, nodes that had customer routes at the end of the \( k'\)th stabilization phase do not change their route afterwards. It follows that no node on \( ik \)'s path to \( d \) moves from a backup customer edge to a primary peer/provider edge and so \( ik \) will remain connected.

The combination of the previous propositions enables us to finally show what we were aiming to prove:

**Proposition 3.12 (connected nodes remain connected)** If \( i \) is a node that is connected to \( d \) at some point in time, then, \( i \) will not turn black from that moment forth (i.e., \( i \) will remain connected to \( d \) via some exportable route throughout the execution of the iterative stabilization procedure).

**Proof.** The only way for a node \( i \) that was connected to \( d \) to become disconnected is a node \( j \) on its route that switches from a backup customer route to a peer/provider route, thus making its new route non-exportable to its peers and providers, and so effectively disconnecting them from \( d \). By Proposition 3.11, we know that node \( ik \) will not cause such a problem during the \((k+1)'th\) stabilization phase. By Proposition 3.10, if a node is assigned a customer route at the end of a stabilization phase that route remains fixed. So, if a gray node \( j \) switches from a customer route to a primary peer/provider route during the \((k+1)'th\) stabilization phase it must be the case that \( j \) did not have a customer route at the end of the \( k'\)th stabilization phase. It follows from Proposition 3.7 that \( j \)'s customer route to \( d \) goes through \( ik \). This, in turn, means that \( ik \) is a (possibly indirect) customer of \( j \). We also know that \( j \) is a gray node that turned gray before stabilization phase \( k \) (i.e., it was already gray by the end of the \( k'\)th stabilization phase). However, this leads to a contradiction to the way we pick black nodes to connect to \( d \) via backup routes.

**Concluding the proof.** In the beginning of each step of the iterative stabilization process we convert some node from black to gray. We know, by Proposition 3.12, that such a node will never turn black again, and that no non-black nodes turn black. Hence, the number of steps in our stabilization process is finite. Consider the last stabilization phase in our stabilization process. At that point in time, no nodes are black, other than the nodes which cannot connect to \( d \) via any route, and so the stabilization phase is no longer constrained by the requirement that nodes should not turn from black to gray. We can therefore think of this last stabilization phase simply as an execution of BGP in which nodes are activated one by one. By Observation 3.5, all white nodes remain white throughout the stabilization phase, and all gray nodes either turn white or remain gray throughout the stabilization phase. This implies that, from some point in time onwards, each node (that can potentially connect to \( d \)), is either only using primary routes, or only using backup routes. Therefore, the
network is effectively equivalent to a network in which for each node we remove all backup edges, or all primary edges (depending on which edges it never uses). For this network, all three Gao-Rexford conditions hold (as the preference condition holds within the set of primary routes, and within the set of backup routes, for each node). Hence, the results in [32] imply convergence to a stable state.

3.4 Backup Interdomain Routing is Safe

In Section 3.3 we have shown that if ASes adhere to our policy guidelines then a stable state always exists in the network. We are now ready to prove our main result: if our policy guidelines hold then BGP is guaranteed to converge to a stable state. The proof’s structure, and the terminology used, are similar in essence to those of Section 3.3. However, the analysis now is much more delicate and subtle. In particular, the proof necessitates new ideas, and techniques different than those used to prove Theorem 3.3.

Theorem 3.13 For any network topology and ASes’ routing policies that uphold the policy guidelines it holds that (probabilistic) BGP safety is guaranteed.

Proof. We begin our proof by presenting a sufficient condition for (probabilistic) BGP safety. Specifically, we show that in order to guarantee BGP safety in the probabilistic model, it is sufficient to consider very simple forms of BGP dynamics.

Reduction to simple BGP dynamics. Consider a simple special case of BGP dynamics, which we refer to as the “sequential dynamics”: start with some initial consistent path assignment. Activate ASes one by one. Every time some AS is activated, let it change its route to its most preferred available route, and make knowledge about its newly chosen route immediately available to its neighbors (that is, update messages from it to its neighbors arrive immediately). We say that a path assignment is “safe” if there is some activation sequence of ASes for which a sequential dynamics starting at reaches a stable state (observe that we do not require that all possible activation sequences in the form of sequential dynamics reach a stable state, just that one such activation sequence does).

Claim 3.14 (a sufficient condition for probabilistic BGP safety) If, for a given network and routing policies, every possible path assignment is safe, then (probabilistic) BGP safety is guaranteed.

Proof. Recall that probabilistic BGP safety implies that the probability that BGP reaches a stable state (from any initial path assignment) converges to 1. Now, observe that, by our assumption that all path assignments are safe, it follows that from any initial path assignment there is some probability, perhaps extremely small, of reaching a stable state. This is because, with some positive probability, the activation sequence of ASes exactly fits the sequential dynamics that reaches a stable state. BGP will fail to converge to a stable path assignment only if the activation of ASes is such that it repeatedly fails to “land” on a “safe” sequential dynamics (for all path assignments it reaches). However, the probability of this happening converges to 0 with time. Hence, the probability that BGP reaches to a stable routing outcome (state) converges to 1 with time.

To be precise, we first have to reach a consistent path assignment, which also happens with some positive probability.
Using this new terminology, we can describe our proof for the existence of a stable state for backup interdomain routing in Section 3.3 as showing that the empty path assignment (in which each node is assigned $\emptyset$) is safe. To prove our BGP safety result this will not be sufficient – we must prove that all path assignments are safe.

**All path assignments are safe.** We now turn to prove that convergence to a stable state is possible from any initial configuration (path assignment) $s$, thus showing that all path assignments are safe. Intuitively speaking, the proof follows a line of argument similar to that of the the proof for the existence of a stable state (Theorem 3.3). However, the analysis here needs to be more subtle, as we no longer have control over the initial state of the network (we are no longer starting from the empty path assignment). Specifically, our proof relies on a refinement of the iterative stabilization process presented in Section 3.3. We characterize “problematic structures” or routing states that may exist in the initial configuration, and show that the (refined) iterative stabilization process eliminates such problematic structures while not adding new ones, thus inevitably reaching a stable state eventually.

**The (refined) iterative stabilization process.** We refine the stabilization phase presented in Section 3.3 as follows: as before, we activate nodes one by one and let nodes that want to move select new routes, while not allowing black nodes to turn gray. However, we now give precedence to certain nodes as follows:

- First, let nodes that want to move to a primary customer route select new routes.
- Only if no node wishes to move to a primary customer route, allow a node that wants to move to a primary peer/provider route select a new route.
- Only allow a gray node that wants to move to (yet another) backup route to move if no nodes that belong to the previous categories wish to select new routes.

The (refined) iterative stabilization process, is a sequence of (refined) stabilization phases, such that after each phase we allow a black node to turn gray, as explained in Section 3.3. Informally, the purpose of this refinement of the iterative stabilization process is to make sure that the routes created be such that they contain long subroutes that only consist of primary edges. This will be helpful in eliminating “problematic structures” that may exist in the network.

**What are problematic structures?** A stabilization phase will reach a stable state if at the end of the phase, no more black nodes wish to move. Recall that any stabilization phase is guaranteed to converge (according to Claim 3.6). We would like to avoid cases in which nodes turn black during the stabilization phase. Hence, we explore the reasons that may cause a non-black node to become disconnected from $d$ (thus turning black). According to Observation 3.4, the answer to this question necessitates the study of the circumstances that can cause a gray node with a customer route to move to a primary peer/provider route (and thus turn white). We refer to such a move as a **bad move**. Our goal is to try to prevent the repetition of bad moves. This shall be done by studying problematic structures that may appear in the traffic pattern, and eliminating them, one by one. We shall require the following definitions:
Definition 3.6 (primary components) We say that two source-nodes $x$ and $y$ are in the same primary component if there is a path in $G$ that leads from $x$ to $y$ that does not violate the export constraint (i.e., in which no node goes through a customer neighbor’s peer/provider route), such that all edges on the path are primary edges.

Remark 3.15 (symmetry) Observe that if there is an exportable path that leads from $x$ to $y$ in $G$ then the same path reversed is also exportable and leads from $y$ to $x$. Hence, the order of $x$ and $y$ is insignificant.

Definition 3.7 (problematic structures) Let $s$ be some consistent path assignment (state). We refer to the two following events as “problematic structures” (in $s$):

- **Type I**: Let $x$ and $y$ be two nodes in the same primary component. Then, $x$ and $y$ are said to form a problematic structure of type I if both $x$ and $y$ have a backup customer route to $d$ in $s$.

- **Type II**: Let $x$ be a node in the same primary component as $d$. Then, $x$ is said to form a problematic structure of type II if $x$ has a backup customer route to $d$ in $s$.

No new problematic structures are formed. The following claims will show that the (refined) iterative stabilization process is such that no new problematic structures are formed during its execution.

Claim 3.16 (problematic structures of type I are not formed) Let $x$ and $w$ be two nodes in the same primary component. If at some point in the (refined) stabilization phase $x$ and $w$ do not form a problematic structure of type I, they will not form such a problematic structure later in the (refined) stabilization phase.

Proof. Assume to the contrary that $x$ and $w$ form a problematic structure of type I, and consider the possible events that may lead to this event. Without loss of generality, assume that $x$ was connected to its backup customer route when $w$ moved to a backup customer route. Let us consider the moment in time just before $w$ switches to the backup customer route, thus creating a problematic structure of type I. As we know that $x$ and $w$ are in the same primary component, there must be an exportable path $P = v_1, v_2, \ldots, v_k$ between $v_1 = x$ and $v_k = w$ that consists of primary edges only. Let $1 \leq t \leq k$ be such that $v_{t+1}$ is a customer of $v_t$ but $v_{t-1}$ is not a provider of $v_t$. Observe that if $x$ is a (possibly indirect) provider of all the nodes along $P$, then $t = 1$. Similarly, if $w$ is a (possibly indirect) provider of all nodes along $P$ then $t = k$.

Notice that $x$ has a backup customer route that is exportable to $v_2$. Hence, by the definition of the (refined) stabilization phase, $v_2$ had a primary route that is at least as good as the route via $x$ when $w$ moved to a backup customer route (otherwise we would have let $v_2$ move prior to $w$). We continue, by induction on $i < t$, to show that $v_i$ had a primary customer route that is at least as good as the route via $v_{i-1}$ when $w$ moved to a backup customer route. For $i < t - 1$, this route does not go through $v_{i+1}$, and is exportable to $v_{i+1}$. Thus, $v_{i+1}$ has a primary route that is at least as good as the route via $v_i$ when $w$ moved to a backup customer route. As for $i = t - 1$, if $v_{t-1}$ is a customer of $v_t$, the same argument follows; if $v_{t-1}$ is a peer of $v_t$, then either the route used by $v_{t-1}$ is exportable to $v_t$ or it is a customer route that goes via $v_1$ - in both cases $v_{t}$ has a primary route that is at least as good as the route via $v_{t-1}$, when $w$ moved to a backup customer route. If $t = k$ we are done, as $w$ has a primary route, which is a contradiction to the fact that it decides to move to a customer backup route.

48
Otherwise, we have, again by induction, that for \( i \geq t \), \( v_i \) has a route that is either a route that goes through \( v_{i+1} \), or a route that is exportable to \( v_{i+1} \) (as it is its customer). So, for \( i = k - 1 \) we get that either \( w \) can use a primary route via \( v_{k-1} \) (and in this case \( w \) would be white, and will not choose a backup customer route), or \( v_{k-1} \) has a route that goes through \( w \) (in which case \( w \) already has a primary customer route, as we are assuming that \( w \) did not yet switch to a backup customer route). For both cases, we get a contradiction, as \( w \) would not switch to a backup customer route from a primary customer route.

Claim 3.17 (customer routes within \( d \)'s primary component are safe) Let \( x \) be a node that has a customer path to \( d \) that consists of primary edges only. Then, \( x \) will never turn gray after the first (refined) stabilization phase. Moreover, \( x \) is guaranteed to have a primary customer route to \( d \) before any activation of a node that chooses a backup route.

Proof. By induction on the \( x \)'s distance from \( d \) along paths of primary edges only. For \( x \) that is a direct provider of \( d \) the proof is trivial. Now, let \( x \) be a node with a customer path to \( d \) that consists of primary edges only, and let \( y \) be the next-hop node on that path. By the induction hypothesis, \( y \) has a primary customer route before any activation of a node that chooses a backup route. Since \( y \) is a customer of \( x \), \( y \)'s customer route cannot pass through \( x \) (as there are no customer-provider cycles). So, if \( x \) does not already have a primary customer route, it can always choose a route through \( y \) and obtain one (recall that by our definition of a (refined) stabilization phase, \( x \) will be allowed to choose its route before any node that wishes to select a backup path).

Claim 3.18 (problematic structures of type II are not formed) Let \( x \) be a node that is in the same primary component as \( d \). If, at some point after the first refines stabilization phase, \( x \) does not form a problematic structure of type II, it will not form such a structure at a later stage in the (refined) iterative stabilization process.

Proof. We consider all types of (exportable) \( r \) routes between \( x \) and \( d \) within the primary component:

- **x has a customer path.** If \( x \) has a customer path to \( d \) that consists of primary edges only, by Claim 3.17, \( x \) will never form a problematic structure.

- **x has a peer path.** If \( x \) has a peer path to \( d \) that consists of primary edges only, its neighbor \( y \) on that path has a customer path to \( d \) that consists of primary edges only. Assume, by way of contradiction, that \( x \) forms a problematic structure of type II, and consider the possible events that can lead to that point. By Claim 3.17, \( y \) will have a primary customer route before any activation of a node that chooses a new backup route. Since \( x \) does not have a backup customer route at this point, \( y \) cannot be using a route that goes via \( x \) unless \( x \) has a primary customer route, which contradicts the assumption that \( x \) turns gray. So \( y \) has primary customer route and \( x \) can use the peer edge \((x, y)\) and have a primary peer route, which again contradicts the assumption that \( x \) turns gray.

- **x has a provider path.** We are left with the case that \( x \) has only provider routes to \( d \) within the primary component. Assume, for point of contradiction, that \( x \) forms a problematic structure of type II, and consider the possible events that may lead to that point. Just before choosing the backup customer edge \( x \) was either black or gray with a peer/provider route. So, its providers could not use \( x \) as their
next hop node. In what follows we show that at least one of $x$’s primary providers is not black and hence $x$ can turn white, which is a contradiction to the fact that $x$ turns gray.

Let $P = v_1, v_2, \ldots, v_k$ be a path between $v_1 = x$ and $v_k = d$ that consists of primary edges only (we know that such a path exists as $x$ is in $d$’s primary component). Let $1 \leq t \leq k$ be such that $v_{t+1}$ is a customer of $v_t$ but $v_{t-1}$ is a not provider of $v_t$ ($x$ is a customer of $v_2$; and if $x$ is an indirect customer of $d$ it cannot possibly have a customer backup route to $d$). According to Claim 3.17, $v_t$ has a primary customer route before any activation of a node that chooses a new backup route. Let $m \leq t$ be the minimum index such that $v_m$ has a customer route (there must be such an index since we know that $v_1$ has such a route). Notice that $m > 1$ as $x$ has no customer route at this point, and that $v_m$’s route doesn’t go through $v_t$ for $i < m$. By backwards induction, it follows that for $i < m$ all nodes have a primary peer/provider route (by the choice of $m$, the claim is trivial for $v_{m-1}$, and is easy to prove for $v_{t-1}$, given that it is true for $v_t$). So, $v_1 = x$ also has a primary peer/provider route – a contradiction.

Concluding the proof. Recall, that we have defined “bad moves” as cases in which nodes switch from backup customer routes to primary peer/provider routes, thus possibly disconnecting nodes that send traffic through them (by making their new routes non-exportable). In what follows, we prove that bad moves are linked to the existence of problematic structures. Specifically, a first bad move in a (refined) stabilization phase can only occur if a problematic structure exists, and after every such bad move a problematic structure ceases to exist. So, by iteratively applying the (refined) stabilization phase, we remove problematic structures, one by one, and add no new such structures along the way. Eventually, no problematic structures will remain. Once this happens, we are assured that no bad moves are possible, and so no non-black (connected) node will ever turn black (disconnected) again. This will imply convergence to a stable state via the exact same arguments as in Section 3.3. Recall, that our intent was to show that from any initial path assignment there is some activation of ASes (sequential dynamics) that leads to a stable state. Our proof shows that the (refined) iterative stabilization process constructs such an activation sequence. Hence, by Claim 3.14, (probabilistic) BGP safety is guaranteed.

Proposition 3.19 (problematic structures and bad moves) Consider the first bad move in the $(k + 1)$’th stabilization phase, where a gray node $x$ moves from a backup customer route to a primary peer/provider route. Then, $x$ is part of a problematic structure which is eliminated by the bad move.

Proof. If $x$ is in the same primary component as $d$, it forms a problematic structure of type II which stops existing once $x$ turns white.

If $x$ is not in the same primary component as $d$, its newly selected route to $d$ must go through a backup edge $e$ (by definition of primary components). Observe that if $e = (u, v)$ where $v$ is $u$’s customer, then $u$ must form (along with $x$) a problematic structure of type I, that existed prior to the bad move. This structure is eliminated by the bad move as $x$ no longer uses this backup edge. We shall now show that it must be the case that the backup edge $e$ is indeed a customer edge (that is, from a provider to a customer), thus concluding the proof.
Let $y$ be $x$’s peer/provider through which $x$ decides to send traffic in the bad move. Let $z$ be $y$’s next hop on $x$’s route through $y$. If $z$ is $y$’s customer, since $(x, y)$ is a primary edge, the backup edge has to be a customer edge (and so we are done). Consider the case that $z$ is not $y$’s customer. It follows that $y$ is $x$’s provider and that $y$ prefers $x$ over $z$. However, this means that $y$ wants to move from $z$ to $x$ at the same time that $x$ wants to move from its backup customer route to $y$. Hence, by the definition of the (refined) stabilization phase we must give precedence to $y$, and so the bad move would not have even taken place in that case—a contradiction.

### 3.5 Other Aspects of Our BGP Safety Result

In this section we present several merits of our (probabilistic) BGP safety result. We also discuss other known types of routing anomalies (other than persistent route oscillations) associated with the common practice of backup routing, and the connections between these and our safety result.

#### 3.5.1 Four Merits

We now present four strengths of our (probabilistic) BGP safety result for backup interdomain routing (presented in Section 3.4) – local implementability, topology-independence, resilience to topology changes, and flexibility.

**Local Implementability.** Our policy guidelines (as in [32]) can easily be implemented by individual ASes, without requiring any coordination with other ASes, or any changes to the protocol (in neither the data plane nor the control plane). Specifically, the Gao-Rexford export condition and our two preference conditions can easily be implemented by individual ASes, simply by configuring their local BGP attributes [67] appropriately. (As in [32], we make the natural assumption that the topology condition holds for the network, which simply means that we assume that no ISP is an indirect provider of itself.)

**Topology-Independence.** Our results in Sections 3.3 and 3.4 are independent of the topology of the network. I.e., these results hold regardless of the specific structure of the AS graph, and of which edges in this graph are designated as backup edges, and which edges represent primary BGP connections. The above implies that the global stability of the Internet is maintained for any network of ASes that upholds our reasonable policy guidelines. We note that designating an edge as a backup connection can be done by individual ASes locally via the BGP community attribute [16, 18] (and need not necessarily be done via coordination between pairs of ASes). Hence, our results show that these local decisions made by ASes never endanger the global stability of the Internet.

**Resilience to Topology Changes.** Observe that if the topology of a network and the routing policies of ASes are such that our policy guidelines hold, then, our guidelines continue to hold even after the removal of nodes, or links, from the AS graph (representing, for example, link and node failures). Also observe, that the addition of new edges and links (representing, e.g., the signing of new business contracts, the creation
of new BGP connections, or the addition of new ASes to the network) does not jeopardize global stability as long as the topology condition still holds, and new nodes (ASes) uphold our export and preference policy guidelines. Hence, our results are resilient to changes to the topology of the network, and, in particular, hold even in the presence of network malfunctions.

**Flexibility.** Recall that, by our first preference condition, ASes are required to prefer primary routes over backup routes. In practice, however, some ASes might prefer backup routes through customer neighbors to primary routes through peer or provider neighbors. We stress that our results in Sections 3.3 and 3.4 apply to such cases as well; we can always treat backup customer routes, that are preferred over primary peer/provider routes, simply as primary customer routes. By making sure that these backup customer routes are less preferred than all actual primary customer routes, we can ensure that they will not be used as long as some primary customer route is available. Hence, our first preference condition (which is the main change to the Gao-Rexford framework needed to guarantee safe backup routing) is flexible, and encompasses many seemingly violating cases.

### 3.5.2 Routing Anomalies

Other than persistent route oscillations, backup routing is known to also potentially cause the following types of routing anomalies:

**Convergence to “bad” stable states.** Consider the routing instance depicted in Figure 3.1. Observe that there are two stable solutions (either 1 sends traffic through 2, or 2’s route goes through 1), and BGP can potentially converge to both (depending on the timing of ASes activations). The designation of the edge (1, d) as a backup edge implies it should not be used if 1 has an alternate primary route to d, as is indeed the case. However, as explained above, this might not happen, as we may converge to a state in which both 1 and 2 are sending traffic along the backup edge. This “nondeterminism” of backup routing (the fact that BGP can converge to multiple states) was first observed in [37]. [37] also discusses related phenomena, known as “BGP Wedgies”, in which the network fails to converge to “good” stable states after the restoration of failed primary edges. In the example in Figure 3.1, even if 1 sends traffic through 2 (the “good” stable state), if the primary link (2, d) fails, then the network will converge to the stable state in which 2 sends traffic through 1 (as it should). The problem is that even once (1, d) is restored, the routing state will remain unchanged (this is because 2 will not revert to 2d, and so it will not inform 1 that a primary route to d is available). We view these routing anomalies as the price one pays for allowing ASes’ routing policies to be expressive enough for more than one stable state to exist in the network (all previous BGP safety results were obtained for routing policies that induce a unique stable state). It would be interesting to devise mechanisms that ensure BGP convergence to “good” states.

**Routing anomalies due to lack of incentive-compatibility.** BGP is known to not be incentive-compatible even in the Gao-Rexford framework [59], and so, naturally, this impossibility result extends to our backup routing framework (which generalizes the Gao-Rexford framework). The lack of incentive-compatibility can be a problem as it might lead to various routing anomalies, and, in particular, cause inconsistencies between the data plane and the control plane [35]. We argue that these undesirable phenomena do not affect

52
our (probabilistic) BGP safety result as even if nodes do not follow the protocol, in an attempt to improve their routing outcomes, it is unlikely that they will deviate from the natural guidelines considered in this chapter. This is due to the fact that the guidelines accurately reflect the economic nature of relationships between ASes, and are therefore induced by ASes’ economic interests. The important problem of making backup routing incentive-compatible remains open.

3.6 Discussion: Enabling More Expressive Routing Policies

Using the probabilistic model, we were able to prove (Section 3.4) that interdomain routing with BGP is safe. This was made possible by the fact that pathological persistent route oscillations, that cannot occur in practice (see Section 3.2), were removed from consideration. Our BGP safety result is a strong indication that the probabilistic approach is more suitable for the analysis of BGP dynamics. In fact, we argue that many of the impossibility results in the literature, no longer hold once we discard the standard model of [40, 39] for the more realistic probabilistic model.

For instance, it was recently shown by Sami et al. [68] that BGP safety necessitates the existence of a unique stable state. [68] shows that if more than one stable state exists in a network then there is bound to be a timing of update messages for which BGP will oscillate indefinitely. This result was proven within the worst-case framework for analyzing BGP dynamics put forth in [40, 39], and, in that model, implies that BGP is unsafe. In backup interdomain routing, while a stable state is guaranteed to exist (as we have shown in Section 3.3), this state need not be unique. Indeed, in the example presented in Section 3.2.1 two stable states exist (and, indeed, there is timing of update messages that induces a BGP oscillation, also presented in that section). Hence, our results bypass the devastating impossibility result in [68] by showing that even though there is a timing of update messages for which BGP will oscillate indefinitely, the probability that this will happen is practically non-existent (converges to zero).

Below we discuss four classes of routing policies – generalized next-hop policies, policy-consistent routing policies [29] (that contain “next-hop policies” as a special case), set-avoiding policies (a special case of forbidden-set routing policies [28]), and shortest-path routing with backup connections. For each of these classes, as in our backup interdomain routing setting, it can easily be shown that worst-case persistent route oscillations are possible. In contrast, we show that, in the probabilistic framework, BGP is guaranteed to converge to a stable state in all cases. We regard the aforementioned classes of routing policies as “toy examples”, meant to illustrate (along with our result for backup routing) the usefulness of the probabilistic model in doing away with unreasonable oscillations. This suggests that ASes can be allowed much more expressiveness in their choice of local routing policies without harming global stability, and so network operators can have much more freedom in configuring local routing policies than implied by previous works on BGP convergence.

3.6.1 Generalized Next Hop

A node has a next-hop policy of its ranking of routes is only affected by the identity of the node it directly forwards traffic to (the next-hop node).

**Definition 3.8 (next-hop policies)** A node $i$ is said to have a next-hop policy if for every two paths $P_1, P_2 \in$
it holds that if the next-hop node on both paths is the same node then \( P_1 =_i P_2 \).

Alternatively, in a next hop preference function for each AS \( i \), there exists a preference function \( g_i \) over the set of \( i \)'s neighbors, and it holds that:

1. \( p_1 \geq_1 p_2 \) if and only if \( g_i(j) \geq g_i(k) \) where \((i, j)\) is the first edge of \( p_1 \) and \((i, k)\) is the first edge of \( p_2 \) and \( j \neq k \).

2. \( p_1 =_i p_2 \) if both paths starts with the same edge.

A generalized next hop preference function is defined by omitting Condition 2 from the above definition.

It is known that there are routing instances (like DISAGREE from \([40, 39]\)) in which all nodes have next-hop policies, and ASes can export all routes, yet persistent route oscillations are possible. By using similar techniques to those presented in Section 3.4 we are able to prove the following theorem:

**Theorem 3.20** If all the ASes have generalized next hop preference functions, and ASes can export all routes, then (probabilistic) BGP safety is guaranteed. Moreover, one can efficiently compute a path of length \( \mathcal{O}(n^2) \) from any consistent state to a stable state.

**Proof.** According to Claim 3.14, we may consider only simple dynamics. Specifically, we may assume that the path assignment forms a routing tree. Now, consider the potential function \( \phi(s) = \sum_{i \in V} g_i(n_i(s)) \) where \( n_i(s) \) is the next hop of \( i \) in the state \( s \). When AS \( i \) is activated, \( g_i(n_i(s)) \) can only increase by the definitions of \( BR_i \) and \( g_i \). Also, since for any other AS \( j \neq i \), \( g_j \) is unchanged, \( \phi(s) \) can only increase as well. Now, since \( \phi(s) \) is bounded above and is increased by at least 1 every round till we reach a stable state, we are guaranteed to converge.

In order to compute a convergence path we activate the ASes in a round-robin fashion. We claim that after \( n \) such rounds we are guaranteed to converge. To prove such a result, we use the fact that convergence is guaranteed for such an execution. Let \( s \) be the resulted stable state, and \( T \) the routing tree defined by \( s \). Let \( s_\ell \) be the resulted state after the \( \ell \) round of activation of each of the ASes.

**Claim 3.21** For every AS \( i \in V \) with depth \( \ell \) from \( d \) in \( T \), \( s_\ell(i) = s(i) \) and \( i \) does not change his selected path from that point on.

**Proof.** By induction on \( \ell \). For \( \ell = 1 \) we have that an AS \( i \) has an edge to \( d \). Since it can choose this edge regardless of the other edges chosen by the other ASes, the path it selects must be at least as good as this path. But, since \( g_i \) only increases and eventually we get to the stable state \( s \), it must be the case that \( i \) chooses the edge to \( d \) right away and does not update its selection any more.

Now, assume that the claim is true for \( \ell - 1 \). Specifically, the subtree of \( T \) that includes all the ASes with distance at most \( \ell - 1 \) is already formed at the end of the \( (\ell - 1) \)th round and will not be changed. Consider an AS \( i \) with depth \( \ell \) from \( d \) in \( T \). Its next hop in \( T \), \( j \), is already connected to \( d \) through the path used on \( T \), and this path will not be changed. Hence, at the \( \ell \)th round, \( i \) can choose the edge to \( j \) regardless of the other edges chosen by the other ASes a this point. Moreover, that means that the path \( i \) selects must be at least as good as the path through \( j \). And again, since \( g_i \) only increases and eventually we get to the stable state \( s \), it must be the case that \( i \) chooses the edge to \( j \) right away and does not update its selection any more.

It follows that after at most \( n \) rounds \( s_n = s \).
3.6.2 Policy-Consistency

Informally, policy consistency holds for a network if no two neighboring ASes disagree over which of two routes is better. A well-studied special case of policy consistency are next-hop routing policies.

**Definition 3.9 (policy-consistency)** [29] We say that policy consistency holds for a network if for every two neighboring source nodes, \( i, j \in V \), and for every two paths in \( P^i \) on which \( i \) does not appear, \( P_1 \) and \( P_2 \), it holds that \( P_1 \preceq_j P_2 \) implies that \((i,j)P_1 \preceq_i (i,j)P_2\).

Notice that policy consistency has an implicit transitive property: Let \( p \) be a simple path that starts at \( i \), goes to \( k \) via subpath \( p_1 \) and from there, via a subpath \( p_2 \), ends at \( d \). Then, if \( p_2' \succeq_k p_2 \) where \( p_2' \) and \( p_1 \) are disjoint, then \( p_2 \preceq_i p \), where \( p' \) is the concatenation of \( p_1 \) and \( p_2' \).

In the proof of the following theorem we only note the changes needed from the proof of Theorem 3.20.

**Theorem 3.22** If policy consistency holds for a network, and ASes can export all routes, then (probabilistic) BGP safety is guaranteed. Moreover, one can efficiently compute a path of length \( O(n^2) \) from any consistent state to a stable state.

**Proof.** Let \( f_i \) be \( i \)'s preference function. Consider the potential function \( \phi(s) = \sum_{i \in V} f_i(p_i(s)) \) where \( p_i(s) \) is the path from \( i \) to \( d \) in the state \( s \). When AS \( i \) is activated, \( f_i(p_i(s)) \) can only increase by the definitions of a best response move and \( f_i \). By the transitive property of policy consistency we get that \( \phi(s) \) can only increase as well: for every AS \( j \) with a path \( p_j \) that includes \( i \), \( f_j(p_j(s)) \) can only increase and for every other AS \( k \), \( f_k(p_k(s)) \) is unchanged as \( p_k(s) \) is unchanged. Now, since \( \phi(s) \) is bounded above and is increased by at least 1 every round till we reach a stable state, we are guaranteed to converge.

The rest of the proof follows the lines of the proof of Theorem 3.20. We only need to replace the reference to \( g_i \) in Claim 3.21 with \( f_i \). ■

3.6.3 Set-Avoiding Policies

A set-avoiding routing policy is such that there is a set of nodes \( S \) such that all routes that do not traverse nodes in \( S \) are preferred to all sets in which some node in \( S \) is traversed. We also require that all paths avoiding \( S \) be assigned the same ranking, and also that all paths that do not avoid \( S \) be assigned the same ranking.

**Definition 3.10 (set-avoiding policies)** We say that a source node \( i \) has a set-avoiding policy if there exists some subset of nodes (not containing \( i \) itself) \( S \subseteq V \setminus \{i\} \), such that

- \( P_1 \preceq_i P_2 \) for every \( P_1 \) and \( P_2 \) in \( P^i \) such that \( P_2 \) does not traverse any node in \( S \), and \( P_1 \) traverses some (one or more) node is \( S \).
- \( P_1 =_i P_2 \) if both \( P_1 \) and \( P_2 \) do not traverse any node in \( S \), or if both \( P_1 \) and \( P_2 \) traverse at least one node in \( S \).

55
• For any route $P \in P^i$, $P \geq \emptyset$ (i.e., being assigned some route is better than not being assigned a route at all).

Similarly to the case of policy-consistency, there are routing instances (based on DISAGREE from [40, 39]) for which all source-nodes have set-avoiding policies, and ASes can export all routes, yet persistent route oscillations are possible. Also, similarly to the case of policy consistency, the existence of such oscillations does not prevent BGP safety in the probabilistic model:

**Theorem 3.23** If all ASes in a network have set-avoiding policies (possibly for many different sets), and ASes can export all routes, then (probabilistic) BGP safety is guaranteed.

**Proof.** As we consider only simple dynamics, the path assignments form a routing tree (with some possible nodes that have no route to $d$ at all). Each node $i$ that have a route to $d$ that does not go through nodes in $S_i$, its avoided set, doesn’t want to move. Thus, if node $i$ moves it means that it moved from a route with nodes from $i$ to a route that avoids $S_i$. Now assume to the contrary that there exists a simple dynamic that oscillates. Let $T_d$ be the set of nodes that their route does not change during the oscillation (specifically, $d \in T_d$). Notice that at least one of the nodes that move during the oscillation has to choose at some point during the oscillation a node from $T_d$ as its next hop. Let $i$ be such a node. But that means that $i$ moves to a route that avoids $S_i$ and that route does not change anymore. So $i$ has no further reason to move, which contradicts the assumption that $i$ takes part in the oscillation. 

### 3.6.4 Shortest-Path Routing with Backup Connections

In the shortest-path routing with backup connections setting, as in our backup interdomain routing setting, some connections (links) are marked as backups, while others are “primary”. We require that a node always prefer a route that goes through a neighbor to which it is connected via a primary edge, over a route that goes through a neighbor to which it is connected via a backup edge. When faced with choices between two (or more) primary routes, or two (or more) backup routes always, a node must always prefer shorter routes to longer ones.

As before, it is easy to show that persistent route oscillations are possible in this setting, yet a (probabilistic) BGP safety result holds:

**Theorem 3.24** For any shortest-path routing with backup connections instance, if AS export all routes, then (probabilistic) BGP safety is guaranteed.

**Proof.** Consider a simple dynamic, starting from some routing tree. Notice that once a node $i$ uses a primary route, it never uses a backup route anymore. Hence, we can divide the nodes into two groups: nodes that at some point have primary route and nodes that only have backup routes. We can throw away the backup edges for the first group of nodes and throw away the primary edges for the second group. We are left with an instance of shortest path routing which converges to a BFS tree as can be seen by a simple induction on the distance from $d$: For nodes with distance 1 to $d$, they will obviously pick an edge to $d$ as their route and will not change that route. Assume that the BFS tree is built for all the nodes with distance at most $\ell$, and consider a node with distance $\ell + 1$. All its possible routes to $d$ of length $\ell + 1$ are available and fixed up to the edge it has to choose, so once it picks an edge it will keep it as it next hop.
3.7 Conclusions and Future Research

In this chapter we argued that, contrary to common belief, backup routing need not endanger the global stability of the Internet. We proposed policy guidelines that are naturally induced by the business relationships between ASes and shows that if ASes adhere to these guidelines then (probabilistic) BGP safety is guaranteed. This surprising positive result, unlike previous ones [31], is easily to implement locally without unduly limiting the expressiveness of ASes’ routing policies. Our results have two facets: on the one hand, they can be regarded as prescriptive, and as offering tools for network operators to make local backup routing decisions without harming the global stability of the Internet. On the other hand, we believe that our results are also somewhat descriptive of today’s situation, and might be helpful in explaining the observed behavior of the Internet.

Our results were made possible thanks to a probabilistic analysis of interdomain routing dynamics. We believe that this is a more realistic approach to the important problem of analyzing BGP routing anomalies than the standard worst-case analysis, and see our results for backup routing, and for other classes of routing policies, as strong evidence of the correctness of this belief. We expect our general approach, and our specific proof techniques, to be useful for the analysis of similar problems in the context of interdomain routing, and in other contexts (like intradomain routing).

Many important questions remain open. A generalization of this work might characterize the set of BGP instances for which probabilistic safety is guaranteed. A different important aspect is BGP’s convergence rate in settings like the ones studied in this chapter, as it has very practical implications. Incentive compatibility in the stochastic model is a game-theoretic aspect that can reflect on the stability of BGP. The question whether an AS can gain by not following the rules set by BGP might require a different approach in the light of the stochastic approach. All of the above suggest interesting and promising directions for future research, along with a comprehensive experimental analysis.
Chapter 4

Cut Problems in Graphs with a Budget Constraint

In this chapter we study budgeted variants of classical cut problems: the Multiway Cut problem, the Multicut problem, and the $k$-Cut problem, and provide approximation algorithms for these problems. Specifically, for the budgeted multiway cut and the $k$-cut problems we provide constant factor approximation algorithms. We show that the budgeted multicut problem is at least as hard to approximate as the sparsest cut problem, and we provide a bi-criteria approximation algorithm for it. 

4.1 Introduction

Given an undirected graph $G = (V, E)$ with a positive cost function on the edges $c : E \to \mathbb{Q}^+$, and a subset of vertices $S \subseteq V$, called terminals, the well-known multiway cut problem is to find a minimum cost subset of edges whose removal disconnects the terminals from each other. The study of the multiway cut problem was initiated by Dahlhaus, Johnson, Papadimitriou, Seymour and Yannakakis [21], who proved that it is MAX-SNP-hard even when restricted to instances with 3 terminals and unit edge cost. They also gave a $(2 - \frac{2}{k})$-approximation algorithm for the problem, where $|S| = k$. Their algorithm finds, for each terminal $s_i$, a minimum cost cut separating $s_i$ from the remaining terminals, and outputs the union of the $k - 1$ cheapest of the $k$ cuts.

In [15], Călinescu, Karloff and Rabani introduced a $(1.5 - \frac{1}{k})$-approximation algorithm, where $|S| = k$. They considered a linear programming relaxation for the multiway cut problem which embeds the given graph into the $(k - 1)$-dimensional simplex. The algorithm of [15] rounds an optimal solution to the linear programming relaxation; its bound was later improved to $\sim 1.3438$ by [52].

In this chapter we study two budgeted variants of the multiway cut problem that differ in their objective function. In the budgeted variants, given an instance of the multiway cut problem together with an additional positive integer $B$, the budget, the problem is to find a subset of edges whose cost is within the given budget and whose removal maximizes the value of the given objective function.

We say that a pair of terminals $(s_i, s_j)$ is separated if there is no path between $s_i$ and $s_j$, and that

\footnote{The results of this chapter appeared in a journal paper in [23].}
a terminal $s_i$ is isolated if there is no path between $s_i$ and any other terminal. The number of isolated terminals is the objective function of the first budgeted variant of the multiway cut problem, referred to as the budgeted isolating multiway cut (BIMC) problem. In the second budgeted variant, referred to as the budgeted separating multiway cut (BSMC) problem, the objective function is the number of separated pairs of terminals. We also consider the weighted versions of both BSMC and BIMC.

An application of the weighted BSMC problem is network design against denial-of-service attacks in networks. In [8], Aura, Bishop and Sniegowski suggest a formal framework for the study of the single-server inhibition attack, which is a common scenario for modelling a denial of service attack. One of the problems they consider is finding the best attack whose cost is within a given budget constraint. In this problem, every client has a non-zero weight denoting its importance. The cost of an attack is the total cost of the disconnected links in the network, and the value of the attack is the total weight of the clients separated from the given server. This problem can be considered as a weighted BSMC by setting the weight of every (server, client) pair to be the client’s weight.

A well known generalization of the multiway cut problem is the multicut problem, which is the problem of finding a minimum cost cut separating a given set of source-sink pairs of vertices. Indeed, the multiway cut problem is a special case of the multicut problem in which the set of source-sink pairs consists of all the pairs of a given set of terminals. Consider the following budgeted variant of the multicut problem. Given is a set of source-sink pairs of vertices together with a budget. Let the source-sink pairs be associated with a non-negative weight. The goal is to find a cut whose cost is within the budget that separates a maximum weight set of source-sink pairs. Thus, this budgeted multicut problem is precisely the weighted version of the BSMC problem.

Finally, given an undirected graph, we consider the problem of finding a set of edges whose cost is within a given budget and whose removal partitions the graph into a maximum number of connected components. This problem, referred to as the budgeted graph disconnection (BGD) problem, can be thought of as the budgeted version of the $k$-cut problem. In the $k$-cut problem, an integer $k$ is given and the goal is to find a minimum cost edge set whose removal partitions the graph into at least $k$ connected components. We note that the cardinality version of BGD (in which all the edges have a unit cost) was introduced by Frederickson and Solis-Oba [30], where it was referred to as the Maximum Components problem.

**4.1.1 Our Results**

The hardness of the multiway cut problem implies that both BIMC and BSMC cannot be efficiently solved unless $P = NP$. Although the problem definitions of BIMC and BSMC are closely related, they capture different aspects of the theory of cuts, and therefore differ in their level of hardness. Thus, we study each of the problems independently.

**BIMC and weighted BIMC**: We give constant factor approximation algorithms that match some of the lower bounds we prove. Our algorithms basically use a greedy approach. In the weighted case we improve on the greedy approach by using an FPTAS for the knapsack problem.

**Weighted BSMC/Budgeted Multicut**: We show that weighted BSMC is at least as hard to approximate as the Sparsest Cut problem is (up to a constant). For the sake of comparison, we note the recent series of results regarding the sparsest cut problem initiated by Arora, Rao and Vazirani [5] improving
on previous \( O(\log k) \)-approximations [7, 62]. In [5], a new structural theorem about metric spaces of negative type is proved, and an \( O(\sqrt{\log n}) \) approximation is presented for the uniform case. Chawla, Gupta and Räcke [17] gave an \( O(\sqrt{\frac{\log k}{n}}) \)-approximation for the general sparsest cut problem, while the current best known result is an \( O(\sqrt{\log k \log \log k}) \)-approximation due to Arora, Lee and Naor [4].

We notice that the weighted BSMC on trees is a special case of the maximum coverage problem, and hence it can be approximated using the algorithm of Khuller, Moss and Naor [54]. We provide an analysis of their algorithm’s performance with respect to the optimal fractional solution of a natural linear programming relaxation. We then consider the weighted BSMC problem on general graphs. We show that the same relaxation has an unbounded integrality gap, and achieve a bi-criteria approximation of \( (\frac{e}{e-1}, O(\log^2 n \log \log n)) \) using a recent hierarchical decomposition of graphs by Räcke (see [66] and [43]).

Interestingly, we show that BSMC is related to the budgeted variant of the Sparsest Cut problem. Specifically, we prove that for certain weight functions, an approximation algorithm for BSMC can be used to derive an approximation algorithm for the budgeted sparsest cut problem, and vice versa.

**BGD:** We give a constant factor approximation algorithm for BGD which is a generalization of Frederickson and Solis-Oba’s algorithm [30] for the cardinality version of BGD. The analysis we present for our algorithm is based on the Gomory-Hu tree (see [36]) and relies upon the approximation algorithm of Saran and Vazirani [69] for the \( k \)-cut problem.

### 4.1.2 Related Work

To the best of our knowledge, except for the cardinality version of BGD, all of the above mentioned budgeted cut problems are studied for the first time here. Nevertheless, there is a vast literature on budgeted optimization problems and we mention the following relevant works.

The \( k \)-median problem is a fundamental problem in which one has to minimize the connection cost of cities to opened facilities, while only \( k \) facilities can be opened. The constraint on the number of opened facilities is the budget constraint. In the Lagrangian relaxation of the \( k \)-median problem the budget constraint is relaxed by moving it into the objective function, i.e., the constraint on the number of opened facilities is replaced by a cost for opening a facility. This is a special case of the facility location problem. Some approximation algorithms for the \( k \)-median problem (for example, see [51]) exploit known approximation algorithms for the facility location problem using Lagrangian relaxation.

In [64], Naor, Shachnai and Tamir introduce a general approximation technique via Lagrangian relaxation for a class of subset selection problems, which is a class of budget problems. They apply their technique to problems of real-time scheduling with budget. They also show that, for some of these problems, the greedy approach yields a constant factor approximation algorithms.

Vohra and Hall [79] considered a budgeted variant for the classical set cover problem, while Khuller, Moss and Naor [54] studied its weighted variant. Khuller et al. gave a constant factor approximation algorithm for the problem that is based on the greedy approach, and showed that their result is tight under a (weak) assumption on the hardness of \( NP \). Their result points out the possible gap between the hardness of a problem and the hardness of its budgeted variant, as the set cover problem cannot be approximated within
a factor of \((1 - \epsilon)\ln n\) for any \(\epsilon > 0\) under the same assumption on the hardness of \(NP\). By improving a former work by Wolsey [80], Sviridenko [72] generalized the result of Khuller et al. for the problem of maximizing any submodular function subject to a budget constraint. We note that this framework does not capture most of the problems we deal with in this chapter, but it does capture the weighted BSMC on trees.

Recently, two variants of a budgeted cut problem were studied. In the minimum size bounded capacity cut (respectively, maximum size bounded capacity cut), given is a graph \(G = (V, E)\) with edge costs and vertex weights, a source \(s \in V\), a sink \(t \in V\), and a budget \(B\). The purpose is to find an \(s - t\)-cut \((S, T)\) whose cost is at most \(B\) such that the total weight of the vertices in \(S\) is minimized (respectively, maximized). Hayrapetyan, Kempe, Pál and Svitkina [45] presented an efficient \((1 - \frac{1}{\lambda}, \frac{1}{\lambda})\)-bi-criteria approximation algorithm for any \(0 < \lambda < 1\) for the minimization variant, while Svitkina and Tardos [73] studied the maximization variant to which they introduced an \((1, \log^2 n)\)-bi-criteria approximation.

4.1.3 Organization

The rest of this chapter is organized as follows. In Section 4.2 we formally define the problems considered in this chapter. The BIMC problem is studied in Section 4.3, while BSMC is studied in Section 4.4. Section 4.5 deals with the BGD problem. We conclude in Section 4.6 with further discussion.

4.2 Preliminaries

In this section we formally define the problems considered in this chapter. In all of these problems, we are given an undirected graph \(G = (V, E)\) with a positive cost function on the edges \(c : E \rightarrow \mathbb{R}^+\), and a positive budget \(B\).

**Problem 4.1 (Budgeted Graph Disconnection (BGD))** Find a subset of edges \(C \subseteq E\) of cost at most \(B\) whose removal partitions the graph into the maximum number of connected components.

Other problem definitions are based on the following terms.

**Definition 1 (Separation)** Given a subset of edges \(C \subseteq E\), we say that vertices \(s\) and \(s'\) \((s' \neq s)\) are separated by \(C\), or, equivalently, that \(C\) is a separating cut of \((s, s')\), if every path between \(s\) and \(s'\) contains at least one edge from \(C\).

**Definition 2 (Isolation)** Let \(S \subseteq V\) be a given subset of vertices. Given a subset of edges \(C \subseteq E\), we say that a vertex \(s \in S\) is isolated by \(C\), or equivalently, that \(C\) is an isolating cut of \(s\), if for every \(s' \in S\), \(s' \neq s\), \(s\) and \(s'\) are separated by \(C\).

In the following problems, we are additionally given a subset of vertices \(S \subseteq V\) (let \(k = |S|\)), called terminals. In the weighted BIMC problem we are also given a weight function on the terminals, \(w : S \rightarrow \mathbb{Z}^+\), used in the next definition.

**Definition 3** Given a subset of edges \(C \subseteq E\), its isolation weight, denoted by \(w(C)\), is the sum of the weights of the terminals isolated by \(C\).
Problem 4.2 (Weighted Budgeted Isolating Multiway Cut (weighted BIMC)) Find a subset of edges $C \subseteq E$ of cost at most $B$ whose isolation weight is maximized.

Without loss of generality we assume that there exists $s \in S$ such that the cost of the minimum cost isolating cut of $s$ is at most $B$. We denote by $BIMC$ the special case of weighted BIMC where $w(s) = 1$ for every $s \in S$.

In the weighted BSMC problem we are additionally given a weight function on the pairs of terminals, $w : S \times S \rightarrow \mathbb{Z}^+$, used in the next definition.

**Definition 4** Given a subset of edges $C \subseteq E$, its separation weight, denoted by $w(C)$, is the sum of the weights of the pairs of terminals separated by $C$.

Problem 4.3 (Weighted Budgeted Separating Multiway Cut (weighted BSMC)) Find a subset of edges $C \subseteq E$ of cost at most $B$ whose separation weight is maximized.

Without loss of generality we assume that for every pair $s, s' \in S$, the cost of the minimum cost separating cut of $s$ and $s'$ is at most $B$. We denote by $BSMC$ the special case of weighted BSMC where $w(s, s') = 1$ for every $s, s' \in S$.

With respect to the same input, we define the Sparsest Cut problem.

**Definition 5** Given a non-empty subset of vertices $U \subset V$, the cut associated with $U$, denoted by $(U, \overline{U})$, is $\{e = (u, v) \in E : u \in U, v \not\in U\}$.

**Definition 6** The Sparsity of the cut $(U, \overline{U})$ is given by $\frac{w(U, \overline{U})}{w(U)}$, where $w(\cdot)$ is the separation weight.

Problem 4.4 (Sparsest Cut) Find a non-empty subset of vertices $U \subset V$ such that the sparsity of its associated cut is minimized.

**Definition 7 (Bi-criteria approximation for a budget problem)** An algorithm $ALG$ is a bi-criteria approximation with parameters $(\alpha, \beta)$ for a given maximization budget problem $\Pi$, or simply an $(\alpha, \beta)$-approximation for $\Pi$, if for every instance of $\Pi$ with budget $B$, $ALG$ outputs a solution whose value is at least $\frac{|OPT|}{\alpha}$ and whose cost is at most $\beta B$, where $|OPT|$ is the value of the optimal solution with respect to the given budget $B$.

4.3 The Budgeted Isolating Multiway Cut Problem

In this section, we study the BIMC and weighted BIMC problems. First we introduce some hardness results, including integrality gaps of two possible linear relaxations. These integrality gaps suggest that an approximation algorithm which is based on these linear relaxations cannot outperform the constant factor approximation algorithm we give for BIMC. Lastly, we also give two approximation algorithms for weighted BIMC, the second of which matches one of the lower bounds we introduce.
4.3.1 Hardness Results

**Proposition 4.1** Unless $P = NP$, there is no $\alpha$-approximation for the BIMC problem for all $\alpha > 1/3$.

**Proof.** Assume to the contrary that there exists an $\alpha$-approximation algorithm for the BIMC problem, $\alpha > 1/3$, and denote it by $ALG$. We show how to solve the multiway cut problem with $k = 3$, which is MAX-SNP-hard. Given an instance of the multiway cut problem, let $C$ be the cost of a minimum multiway cut. Notice that for every budget $B \geq C$, $ALG$ will return a solution that isolates at least $\alpha \cdot 3 > 1$ terminals. Since every cut that isolates at least two terminals isolates all three terminals, it follows that if $B \geq C$, $ALG$ will isolate all the terminals, and otherwise it will isolate at most one terminal. Thus, by using $ALG$, it is possible to binary search the range $[0, \sum_{e \in E} c(e)]$ for the value $C$, and furthermore, one can find a minimum multiway cut of the given instance. ■

Proposition 4.1 can be easily generalized as follows (proofs are omitted).

**Proposition 4.2** Unless $P = NP$ there is no $\alpha$-approximation for the BIMC problem with $k$ (fixed) terminals, for every $\alpha > 1 - 2/k$.

**Proposition 4.3** Unless $P = NP$ there is no $\alpha$-approximation for the BIMC problem for every $\alpha > 1 - 2/OPT$, where $OPT > 2$ is the number of isolated terminals in an optimal solution.

**Integrality Gap of Linear Programming Relaxations**

We consider two natural linear programming relaxations for the BIMC problem. In these relaxations we assume that for every $s \in S$, the cost of the minimum cost isolating cut of $s$ is at most $B$ (if not, a slight modification can be made in the relaxations and the relevant claims still hold). The first one is a straightforward formulation of the problem. We assign an indicator variable $y_e$ for every edge $e \in E$, which will be set to 1 iff the edge $e$ is picked to the solution. The budget constraint can be stated accordingly. We also assign a variable $x_s$ for every terminal $s \in S$, which indicates whether terminal $s$ is isolated by the given solution. In order to enforce that a terminal $s$ is isolated if $x_s = 1$, we state the constraint $x_s \leq \sum_{e \in P_{s,s'}} y_e$ for each path between $s$ and any other terminal $s'$.

\[
\begin{align*}
\text{max} & \quad \sum_{s \in S} x_s \\
\text{s.t.} & \quad x_s - \sum_{e \in P_{s,s'}} y_e \leq 0 \quad \text{for every } s, s' \in S \ (s \neq s') \\
& \quad \text{and path } P_{s,s'} \text{ from } s \text{ to } s' \\
& \quad \sum_{e \in E} c(e) \cdot y_e \leq B \\
& \quad 0 \leq x_s \leq 1 \quad \text{for every } s \in S \\
& \quad 0 \leq y_e \quad \text{for every } e \in E
\end{align*}
\]

(N-ISO-LP)

**Proposition 4.4** The integrality gap of $N$-ISO-LP is at least 2.

**Proof.** Consider a star with $N$ leaves that are all terminals, set $B = N/2$, and $c(e) = 1$ for every edge $e$. An optimal integral solution picks $N/2$ edges and has a value of $N/2$, while an optimal fractional solution is: $y_e = \frac{1}{2}$ for every $e \in E$ and $x_s = 1$ for every $s \in S$. This solution has value $N$. ■
The second linear programming formulation we consider is derived from the linear programming relaxation of the multiway cut problem presented in [15]. We assume that \( S = \{s_1, \ldots, s_k\} \), and embed the given graph into the \( k \)-dimensional simplex. We reserve the 0-coordinate for the connected component that contains all the terminals not isolated by the solution, and the \( i \)-th coordinate for the connected component that contains terminal \( s_i \), if terminal \( s_i \) is isolated by the solution. Thus, we allow terminal \( s_i \) to be mapped to either the 0th component, or the \( i \)-th component.

\[
\begin{align*}
\max_{s_i \in S} & \quad \sum_{s_i \in S} x^i_{s_i} \\
\text{s.t.} & \quad x^i_{s_i} + x^0_{s_i} = 1 \quad \text{for } 1 \leq i \leq k \\
& \quad \sum_{i=0}^{k} x^i_v = 1 \quad \text{for every } v \in V \setminus S \\
& \quad x^i_v \geq 0 \quad \text{for every } v \in V \text{ and } 0 \leq i \leq k \\
& \quad y_e = \frac{1}{2} \sum_{i=0}^{k} |x^i_u - x^i_v| \quad \text{for every } e = (u, v) \in E \\
& \quad \sum_{e \in E} c(e) \cdot y_e \leq B
\end{align*}
\]

Proposition 4.5 The integrality gap of CKR-ISO-LP is at least 2.

Proof. Consider a clique of \( N \) terminals, let \( B = N - 1 \), and \( c(e) = 1 \) for every edge. An optimal solution can isolate only one terminal, while an optimal fractional solution is: \( x^i_{s_i} = \frac{2}{N} \) for every \( 1 \leq i \leq k \) (due to feasibility, the rest of the solution is uniquely defined), which has a value of 2. 

The integrality gaps shown above suggest that using “natural” linear relaxations, one cannot improve on the approximation factor achieved by the following approximation algorithm for BIMC.

4.3.2 A Greedy Approximation Algorithm for BIMC

The following greedy algorithm for BIMC is a variant of the algorithm presented in [21] for the multiway cut problem. Note that a minimum cost isolating cut for \( s_i \in S \) can be computed efficiently by merging the terminals in \( S \setminus \{s_i\} \) into a single node \( r \) and computing a minimum cut separating \( r \) from \( s_i \).

Algorithm 1 A greedy algorithm for BIMC

\[
\begin{align*}
\text{for each } s \in S \text{ do} \\
& \quad \text{Find a minimum cost isolating cut for } s, \text{ and denote it by } C_s. \\
\text{end for} \\
\text{Sort the cuts in a non-decreasing order of their cost.} \\
\text{Choose the maximal sequence of cuts, starting from the cheapest, whose total cost is at most } B.
\end{align*}
\]

Lemma 4.6 Let \( \ell \) denote the number of isolated terminals in an optimal solution. Algorithm 1 achieves an approximation factor of \( \frac{1}{2} \) if \( \ell \) is even, and \( \frac{1}{2} - \frac{1}{2\ell} \) if \( \ell \) is odd.\(^2\)

Proof. Let \( OPT \) be an optimal solution, and let \( I \) denote the set of terminals isolated by \( OPT \). We assume without loss of generality that there is no edge in \( OPT \) that can be removed without changing the

\(^2\)For the trivial case in which \( \ell = 1 \) the algorithm finds an optimal solution.
set of isolated terminals. Let $G' = (V, E \setminus OPT)$. For $s \in I$, let $OPT_s$ be the edges in $OPT$ that have an endpoint in the connected component of $s$ in $G'$.

Consider the following charging scheme for the terminals in $I$. Charge the cost of every edge $e \in OPT$ as follows: if there exist two distinct terminals $s \in I$ and $s' \in I$, such that $e \in OPT_s$ and $e \in OPT_{s'}$, then charge each of the two terminals with $c(e)/2$; otherwise, charge the terminal $s \in I$, such that $e \in OPT_s$, with $c(e)$. Denote by $c(s)$ the total cost charged to terminal $s$. Obviously, since every edge in $OPT$ is clearly paid for by the charging scheme,

$$\sum_{s \in I} c(s) = c(OPT) \leq B.$$ 

Since $OPT_s$ is an isolating cut for each $s \in I$,

$$c(C_s) \leq c(OPT_s) \leq 2c(s).$$

Let $A_\ell$ be the set of the first $\ell$ terminals sorted in Step 4 of the algorithm. Notice that

$$\sum_{s \in A_\ell} c(C_s) \leq \sum_{s \in I} c(C_s) \leq 2 \sum_{s \in I} c(s) \leq 2B.$$ 

Thus, the cost of the first $\lfloor \ell/2 \rfloor$ terminals is at most $B$, and the lemma follows immediately. ■

The above analysis is tight as the following example shows.

**Example 4.7** Let $N$ be an odd integer, let $B = N(N - 1)/2$, and consider a graph with the following two connected components:

- A clique of $N$ terminals with $c(e) = 1$ for every edge in the clique;
- A star with $N$ leaves, all of which are terminals, and each leaf is connected to the root by an edge of cost $c(e) = N - 1 - \epsilon$.

Choosing all the clique edges results in a solution whose value is $N$, while Algorithm 1 chooses only edges from the star, and achieves a value of at most $B/(N - 1 - \epsilon) = \lfloor N/2 \rfloor = (N - 1)/2$.

### 4.3.3 Approximation Algorithms for the Weighted BIMC Problem

We present two algorithms for the weighted BIMC problem.

**A Greedy Algorithm**

The following is a generalization of Algorithm 1.

**Lemma 4.8** Algorithm 2 achieves an approximation factor of $\frac{1}{4}$.

**Proof.** We follow the proof of Lemma 4.6 and only specify the changes needed in the analysis. Let $\ell$ be the isolation weight of $OPT$, i.e., the value of the optimal solution. By applying the charging scheme, and since the sequence $\{C_i\}_{i=1}^\ell$ is sorted with respect to the ratio of cost to weight, any prefix of the sequence
Algorithm 2 A greedy algorithm for weighted BIMC

for each \( s \in S \) do
    Find a minimum cost isolating cut for \( s \), and denote it by \( C_s \).
end for

Sort the cuts satisfying \( c(C_s) \leq B \), \( s \in S \), in non-decreasing order of the ratio between their cost and the weight of their terminal \((c(C_s)/w(s))\). Let \( \{C_i\}_{i=1}^k \) be the resulting sequence of cuts.

Let \( \{C_i\}_{i=1}^{m+1} \) be the maximal prefix of \( \{C_i\}_{i=1}^k \) with a total cost of at most \( B \). Choose the heavier cut (with respect to isolation weight) between \( \bigcup_{i=1}^m C_i \) and \( C_{m+1} \) (if \( m = k \) then \( \bigcup_{i=1}^m C_i \) is an optimal solution).

with isolation weight at most \( \ell \) costs at most \( 2B \), and similarly, any prefix with isolation weight \( \ell/2 \) costs at most \( B \). Thus, the cut \( \bigcup_{i=1}^{\ell+1} C_i \), which costs more than \( B \), must have an isolation weight of at least \( \ell/2 \), implying that the heavier cut between \( \bigcup_{i=1}^m C_i \) and \( C_{m+1} \) has weight at least \( \ell/4 \). ■

A \( (\frac{1}{3} - \epsilon) \)-Approximation

It can be readily seen from the analysis of Algorithm 2 that improving the approximation factor requires an efficient use of the given budget. To this end, we use as a procedure the FPTAS for the knapsack problem presented in [48], denoted by \( A(\pi, \epsilon) \), where \( \pi \) is the knapsack instance.

Algorithm 3 A \( (\frac{1}{3} - \epsilon) \)-approximation for weighted BSMC

for each \( s \in S \) do
    Find a minimum cost isolating cut for \( s \), and denote it by \( C_s \).
end for

Construct an instance of the knapsack problem, \( \pi \), as follows: treat each terminal \( s \in S \) such that \( c(C_s) \leq B \) as an item whose profit is \( w(s) \) and whose size is \( c(C_s) \), and let \( B \) be the capacity of the knapsack.

Run \( A(\pi, \epsilon) \) and denote by \( P \) the resulting subset of terminals. Output the cut \( \bigcup_{s \in P} C_s \).

Let \( OPT \) be an optimal solution for the weighted BIMC instance. Since every terminal \( s \), for which \( c(C_s) > B \), cannot be isolated by either \( OPT \) or Algorithm 3, we ignore such terminals in what follows. Let \( I \) denote the set of the terminals isolated by \( OPT \) and let \( \ell \) be the isolation weight of \( OPT \), i.e., the value of the optimal solution. Denote by \( |OPT(\pi)| \) the value of the optimal solution for the knapsack instance \( \pi \).

Lemma 4.9 \( |OPT(\pi)| \geq \frac{1}{3} \ell \).

Proof. Let \( U = \{X \subseteq I \mid \sum_{s \in X} w(s) \geq \frac{1}{3} \ell \} \), i.e., \( U \) is the collection of subsets of \( I \) having profit at least \( \frac{1}{3} \ell \). Let \( Y \in U \) be a subset of minimum size in \( \pi \) (notice that there must exist such a subset). Assume to the contrary that \( |OPT(\pi)| < \frac{1}{4} \ell \), and in particular that \( \sum_{s \in Y} c(C_s) > B \). As every single item is a feasible solution by itself, it follows that for every item \( s \), \( w(s) < \frac{1}{4} \ell \). Thus, there are at least two terminals in \( Y \), and moreover, \( \sum_{s \in Y} w(s) < \frac{3}{4} \ell \) (otherwise, by taking off a terminal from \( Y \) we get a contradiction to the minimality of \( Y \) in \( U \) with respect to size). By arguments similar to those used in the proof of Lemma 4.6,
we get that \( \sum_{s \in I} c(C_s) \leq 2B \). Thus,
\[
\sum_{s \in I \setminus Y} c(C_s) < B,
\]
and
\[
\sum_{s \in I \setminus Y} w(s) > \frac{1}{3} \ell.
\]
Thus, \( I \setminus Y \) is a feasible solution to \( \pi \) with the desired value.

**Lemma 4.10** Algorithm 3 achieves an approximation factor of \( \frac{1}{3} - \epsilon \).

**Proof.** Follows from Lemma 4.9 and the FPTAS for the knapsack problem.

We note that it can be shown that Lemma 4.9 is tight for arbitrarily large values of \( k \) by constructing appropriate examples.

In what follows we show that given a lower bound on the optimal solution’s value (for example, the value of the solution returned by Algorithm 3), the analysis performed in Lemma 4.9 can be improved for some instances. First, we generalize the definition of the sets \( U \) and \( Y \) as follows: for \( i > 1 \), let
\[
U_i = \left\{ X \subseteq S \mid \sum_{s \in X} w(s) \geq \left( \frac{1}{2} - \frac{1}{4i - 2} \right) \ell \right\}
\]
and let \( Y_i \) be a set in \( U_i \) of minimum size in \( \pi \). Notice that \( U = U_2 \) and \( Y = Y_2 \). The following lemma, generalizes Lemma 4.9.

**Lemma 4.11** If, for every \( X \in U_i \), \( |X| \geq i \), then \( |OPT(\pi)| \geq (\frac{1}{2} - \frac{1}{4i - 2}) \ell \).

**Proof.** Assume by contradiction that \( |OPT(\pi)| < (\frac{1}{2} - \frac{1}{4i - 2}) \ell \) and in particular that \( \sum_{s \in Y_i} c(C_s) > B \). Recall that \( |Y_i| \geq i \) and that any subset of \( Y_i \) of size \( i - 1 \) must have a total weight less than \( (\frac{1}{2} - \frac{1}{4i - 2}) \ell \). Thus, there must exist a terminal in \( Y_i \) that has a weight of at most
\[
\left( \frac{1}{2} - \frac{1}{4i - 2} \right) \cdot \frac{\ell}{i - 1} \leq \frac{\ell}{2i - 1},
\]
and thus, from the minimality of \( Y_i \) in \( U \) with respect to size,
\[
\sum_{s \in Y_i} w(s) < \left( \frac{1}{2} + \frac{1}{4i - 2} \right) \ell.
\]
By similar arguments to those used in the proof of Lemma 4.9, we get that \( \sum_{s \in I \setminus Y_i} c(C_s) < B \) and \( \sum_{s \in I \setminus Y_i} w(s) > (\frac{1}{2} - \frac{1}{4i - 2}) \ell \) and thus \( I \setminus Y_i \) is a feasible solution to \( \pi \) with the desired value.

Now, given an instance of weighted BIMC, let \( \ell' \) be a lower bound on \( l \). Define
\[
U'_i = \left\{ X \subseteq S \mid \sum_{s \in X} w(s) \geq \left( \frac{1}{2} - \frac{1}{4i - 2} \right) \ell' \right\}.
\]
Since \( U_i \subseteq U'_i \), if for every \( X \in U'_i \) it holds that \( |X| \geq i \), it follows from Lemma 4.11 that the solution returned by Algorithm 3 is within \( \frac{1}{2} - \frac{1}{4i-2} - \epsilon \) of the optimal solution. Notice that finding such maximal \( i \) can be done efficiently.

### 4.4 The Weighted Budgeted Separating Multiway Cut Problem

In this section, we study weighted the BSMC problem which is equivalent to the weighted budgeted variant of the multicut problem. We show that approximating it is at least as hard as the sparsest cut problem. We present a natural linear programming relaxation for the problem and show that it has an unbounded integrality gap for general graphs. We notice that the weighted BSMC on trees is a special case of the maximum coverage problem, and hence it can be approximated using the algorithm of Khuller, Moss and Naor [54].

Moreover, we prove that their algorithm’s output is within a constant factor from the optimal fractional solution of the aforementioned linear relaxation, implying a constant integrality gap for tree instances. Lastly, we use a recent hierarchical decomposition of graphs by Räcke (see [66] and [43]) to obtain a bi-criteria approximation of \( (\frac{e^r}{e^r-1}, O(\log^2 n \log \log n)) \) for arbitrary graphs.

#### 4.4.1 Hardness Results

**Hardness with respect to the Sparsest Cut Problem**

We first prove the following lemma.

**Lemma 4.12** Given a non-empty cut \( C \subseteq E \) that partitions \( G \) into \( r > 2 \) connected components, there is an algorithm that finds a cut \( C' \subset C \) such that \( c(C')/w(C') \leq c(C)/w(C) \) and \( C' \) partitions \( G \) into \( r - 1 \) connected components.

**Proof.** Given the \( r \) connected components into which \( G \) is partitioned by \( C \), denote by \( V_i \) the vertex set of the \( i \)th connected component. Define:

\[
C_{ij} = \{ e = (u, v) \in C : u \in V_i \land v \in V_j \} .
\]

Obviously, \( C = \bigcup_{1 \leq i < j \leq r} C_{ij} \) and \( C_{ij} \cap C_{im} = \emptyset \) for every \( (i, j) \neq (\ell, m) \). Define the separation weight of \( C_{ij} \), denoted by \( w(C_{ij}) \), as the sum of the weights of the pairs of terminals \( (s_g, s_h) \) such that \( s_g \in S \cap V_i \) and \( s_h \in S \cap V_j \). Then:

\[
w(C) \geq \sum_{1 \leq i < j \leq r} w(C_{ij}) \quad \text{and} \quad c(C) = \sum_{1 \leq i < j \leq r} c(C_{ij}). \tag{4.1}
\]

**Proposition 4.13** There exists a \( C_{ij} \) (\( 1 \leq i < j \leq r \)) such that \( c(C_{ij})/w(C_{ij}) \geq c(C)/w(C) \).

**Proof.** Assume to the contrary that for all \( 1 \leq i < j \leq r \), \( c(C_{ij})/w(C_{ij}) < c(C)/w(C) \). Thus, \( c(C_{ij}) \cdot w(C) < c(C) \cdot w(C_{ij}) \), and by summing up over all \( 1 \leq i < j \leq r \) and from Equality (4.1), we get that

\[
c(C) \cdot w(C) = \sum_{1 \leq i < j \leq r} c(C_{ij}) \cdot w(C) < c(C) \cdot \sum_{1 \leq i < j \leq r} w(C_{ij})
\]

68
and thus \( w(C) < \sum_{1 \leq i < j \leq r} w(C_{ij}) \) contradicting Inequality (4.1).

Note that a \( C_{ij} \) (1 \( \leq i < j \leq r \)) satisfying the above proposition can be found easily. Now, define \( C' = C \setminus C_{ij} \), and observe that \( C' \) partitions \( G \) into \( r - 1 \) connected components. (The previous connected components of \( V_i \) and \( V_j \) are merged into one, and the rest are not changed.) Now, since \( c(C_{ij})/w(C_{ij}) \geq c(C)/w(C) \), we get:

\[
c(C_{ij}) \cdot w(C) \geq c(C) \cdot w(C_{ij})
\]

and

\[
(c(C) - c(C_{ij})) \cdot w(C) \leq c(C) \cdot (w(C) - w(C_{ij})).
\]

Thus,

\[
c(C')/w(C') = (c(C) - c(C_{ij}))/(w(C) - w(C_{ij})) \leq c(C)/w(C)
\]

and the lemma follows.

**Corollary 4.14** Given a non-empty cut \( C \subseteq E \), there is an algorithm that finds a non-empty subset of vertices \( U \subseteq V \) such that the sparsity of the cut associated with \( U \) is at most \( c(C)/w(C) \).

**Proof.** Assume that cut \( C \) partitions \( G \) into \( r > 1 \) connected components, and denote by \( V_i \) the vertex set of the \( i \)th connected component. If \( r = 2 \), then \( V_1 \) can be returned. Otherwise, apply recursively the algorithm from Lemma 4.12 until a cut \( C' \) that partitions \( G \) into two connected components is obtained. Let \( V'_1 \) be the vertices of one of these connected components. Then, \( V'_1 \) can be returned.

The following theorem shows that the weighted BSMC problem is at least as hard to approximate as the sparsest cut problem is (up to a constant).

**Theorem 4.15** Let ALG be an \((\alpha, \beta)\)-approximation for weighted BSMC. Then, there exists a \((1 + \epsilon)\alpha\beta\)-approximation for Sparsest Cut, for every \( \epsilon > 0 \).

**Proof.** Assume we are given an instance of the sparsest cut problem, denote it by \( \pi \) and let \( OPT_\pi \) denote its optimal solution. Denote the sparsity of the optimal solution by

\[
|OPT_\pi| = \frac{c(OPT_\pi, OPT_\pi)}{w(OPT_\pi, OPT_\pi)}.
\]

Let \((\pi, B)\) denote the input for the weighted BSMC problem that consists of the instance \( \pi \) and the budget \( B \), and let \( OPT_{\pi, B} \) be a corresponding optimal solution. Then, since \((OPT_\pi, OPT_\pi)\) is a feasible solution for the weighted BSMC problem on \((\pi, B)\) for every \( B \geq c(OPT_\pi, OPT_\pi) \), then \( w(OPT_\pi, OPT_\pi) \leq w(OPT_{\pi, B}) \) for every \( B \geq c(OPT_\pi, OPT_\pi) \).

For \( \lfloor \log_{\alpha+\epsilon} c\left(C_{min}\right) \rfloor \leq i \leq \lfloor \log_{\beta+\epsilon} c\left(E\right) \rfloor \), where \( C_{min} \) is the minimum cost cut in \( G \), let \( C_{B_i} \) be the cut returned by \( ALG(\pi, B_i = (1 + \epsilon)^i) \). Then, by applying Corollary 4.14 on each \( C_{B_i} \), we can obtain a non-empty subset of vertices \( U_i \subseteq V \) such that the sparsity of the cut associated with \( U_i \) is at most

\[
\frac{c(C_{B_i})}{w(C_{B_i})} \leq \frac{\beta B_i}{w(OPT_{\pi, B_i})/\alpha} = \frac{\alpha \beta}{w(OPT_{\pi, B_i})}.
\]
Let \( j = \lceil \log_{1+\epsilon} c(OPT_\pi, OPT_\pi) \rceil \). Then,

\[
\frac{B_j}{w(OPT_{\pi,B_j})} \leq (1 + \epsilon) \frac{c(OPT_\pi, OPT_\pi)}{w(OPT_\pi, OPT_\pi)} = (1 + \epsilon)|OPT_\pi|.
\]

We conclude that the sparsity of \((U_j, \overline{U_j})\) is at most \((1 + \epsilon)\alpha\beta|OPT_\pi|\), and the theorem follows by outputting the sparsest cut among the computed cuts

\[
\{(U_i, \overline{U_i})\}_{\lceil \log_{1+\epsilon} c(C_{\min}) \rceil \leq i \leq \lceil \log_{1+\epsilon} c(E) \rceil}.
\]

### Integrality Gap of a Linear Programming Relaxation

In this subsection we give a natural linear programming relaxation for the weighted BSMC problem. We assign an indicator variable \( y_e \) for every edge \( e \in E \) (we assume without loss of generality that \( c(e) \leq B \) for every \( e \in E \)). The budget constraint can be stated accordingly. We assume that \( S = \{s_1, \ldots, s_k\} \) and assign an indicator variable \( x_{ij} \) for every pair of terminals \( s_i, s_j \in S \) indicating whether the pair \( (s_i, s_j) \) is separated by the given solution. The separation constraint of a pair of terminals \( (s_i, s_j) \) is \( x_{ij} \leq \sum_{e \in P_{i,j}} y_e \), for each path \( P_{i,j} \) between \( s_i \) and \( s_j \).

\[
\max_{\text{s.t.}} \sum_{s_i, s_j \in S} w(s_i, s_j) \cdot x_{ij} \quad \text{(SEP-LP)}
\]

\[
x_{ij} - \sum_{e \in P_{i,j}} y_e \leq 0 \quad \text{for every } s_i, s_j \in S \text{ and path } P_{i,j} \text{ from } s_i \text{ to } s_j
\]

\[
\sum_{e \in E} c(e) \cdot y_e \leq B \quad \text{for every } s_i, s_j \in S
\]

\[
0 \leq x_{ij} \leq 1 \quad \text{for every } s_i, s_j \in S
\]

\[
0 \leq y_e \quad \text{for every } e \in E
\]

**Proposition 4.16** The integrality gap of SEP-LP is \( \Omega(n) \).

**Proof.** Consider the following graph that consists of 3 parts:

- A square of non-terminals \( \{v_1, v_2, v_3, v_4\} \) and edges with cost of \( c(e) = 1 + \epsilon \).
- \( N/2 \) terminals are connected to \( v_1 \) by edges with \( c(e) = 2 \).
- \( N/2 \) terminals are connected to \( v_3 \) by edges with \( c(e) = 2 \).

Let \( B = 2 \), and \( w(s_i, s_j) = 1 \) for every \( s_i, s_j \in S \). An optimal solution picks any of the non-square edges and has a value of \( N - 1 \), while a feasible optimal fractional solution is: \( y_e = \frac{1}{(1+\epsilon)} \) for the square edges \( (v_2, v_3) \) and \( (v_3, v_4) \), and \( x_{ij} = 1/(1 + \epsilon) \) for every pair of terminals \( (s_i, s_j) \) such that \( (s_i, v_1) \in E \) and \( (s_j, v_3) \in E \). This solution has a value of \( N^2 \cdot \frac{1}{(1+\epsilon)} \).

This proposition implies that an algorithm based on the above linear relaxation would have poor performance. Nevertheless, in what follows we show an approximation algorithm for the special case of trees based on this linear relaxation.
4.4.2 Approximation Algorithms for Weighted BSMC in Trees

As the multicut problem on trees is NP-hard, so is the BSMC problem on trees. At the same time, like there are better approximation algorithms for the multicut problem on trees than for multicut on general instances, BSMC is apparently easier when restricted to trees, as it is a special case of the maximum coverage problem. The elements are the pairs of terminals, and the edges are the sets. The set that corresponds to an edge \( e \) contains a pair of terminals \((s_i, s_j)\) if and only if \( e \) belongs to the unique path between \( s_i \) and \( s_j \). The weight of an element is the weight of the corresponding pair, while the cost of a set is the cost of the corresponding edge.

In [54], Khuller, Moss and Naor presented an \( \frac{\epsilon-1}{\epsilon} \)-approximation algorithm for the maximum coverage problem, and accordingly, the same algorithm can be used to approximate BSMC on trees. We note that the analysis of [54] implies that the algorithm’s output is within a factor of \( \frac{\epsilon-1}{\epsilon} \) from the optimal integral solution, but presents no guarantee on the ratio between the output and the optimal fractional solution.

In what follows we provide a dual-fitting analysis for the same algorithm, which proves that the algorithm’s output is within a factor of \( \frac{1}{3} \) from the optimal fractional solution (we note that our analysis holds for a general instance of the maximum coverage problem). This type of guarantee might be important for some applications (for example, see [61]).

The weighted BSMC problem on trees can be cast as a linear integer program, whose fractional relaxation is SEP-LP. Notice that there is a unique path in the given tree between \( s_i \) and \( s_j \), denoted by \( P_{ij} \). The dual LP of SEP-LP is:

\[
\begin{align*}
\min & \quad B \cdot \gamma + \sum_{s_i, s_j \in S} \beta_{ij} \\
\text{s.t.} & \quad c(e) \cdot \gamma - \sum_{i,j \in P_{ij}} \alpha_{ij} \geq 0 \quad \text{for every } e \in E \\
& \quad \alpha_{ij} + \beta_{ij} \geq w(s_i, s_j) \quad \text{for every } s_i, s_j \in S \\
& \quad \alpha_{ij} \geq 0 \quad \text{for every } s_i, s_j \in S \\
& \quad \beta_{ij} \geq 0 \quad \text{for every } s_i, s_j \in S \\
& \quad \gamma \geq 0
\end{align*}
\]

We define the \textit{worthiness of an edge} \( e \) with respect to \( C \), a feasible solution, as

\[
\Gamma_C(e) = \frac{\sum_{i,j \in P_{ij}} w(s_i, s_j) \cdot (1 - x_{ij})}{c(e)},
\]

where \( x \) is the corresponding solution of SEP-LP. Algorithm 4 greedily adds edges to the solution as long as the budget constraint is not violated. At the same time it maintains the corresponding solution of SEP-LP. Note that we may assume without loss of generality that for all \( e \in E, c(e) \leq B \).

**Observation 4.17** If \( C \neq E \), then \( \sum_{h=0}^{\lfloor |C|/2 \rfloor} c(e_h) + c(e_{|C|}) > B \).

The following proposition follows immediately from the definition of the worthiness of an edge and the fact that edges are only added to the solution during the algorithm.

**Proposition 4.18** For every \( e \in E \) and \( 0 < h \leq |C| \), \( \Gamma_{C_h}(e) \leq \Gamma_{C_{h-1}}(e) \), i.e. the worthiness of an edge can only decrease during the algorithm.
Algorithm 4 A greedy algorithm for weighted BSMC on Trees

Initialize: $h = 0$, $C_0 = \emptyset$, $x_{ij} = 0$ for every $s_i, s_j \in S$ and $y_e = 0$ for every $e \in E$.

while $\exists e \in E \setminus C_h$

Let $e_h \in E \setminus C_h$ be an edge with the lowest cost among the edges with the maximum value of $\Gamma_{C_h}$.

If $c(e_h) > B - c(C_h)$, output the better solution between $\{e_h\}$ and $C = C_h$.

$C_{h+1} \leftarrow C_h \cup \{e_h\}$.

Set $y_{e_h} = 1$ and $x_{ij} = 1$ for all the pairs of terminals $(s_i, s_j)$ separated by $e_h$.

$h \leftarrow h + 1$

end while

Output $C = C_h$.

Corollary 4.19 If $C \neq E$, then

$$c(e_{|C|}) \cdot \Gamma_C(e_{|C|}) \leq c(e_{|C|}) \cdot \Gamma_{C_0}(e_{|C|}) = w(\{e_{|C|}\}),$$

i.e., adding the edge $e_{|C|}$ to $C$ increases its separation weight by at most the separation weight of $\{e_{|C|}\}$.

Corollary 4.20 For every $0 < h \leq |C|$, $\Gamma_{C_h}(e_h) \leq \Gamma_{C_{h-1}}(e_{h-1})$.

Proof. From Proposition 4.18, $\Gamma_{C_h}(e_h) \leq \Gamma_{C_{h-1}}(e_{h-1})$, and the corollary follows from the greediness of Algorithm 4. ■

Observation 4.21 $w(C) = \sum_{h=0}^{\left|C\right|-1} c(e_h) \cdot \Gamma_{C_h}(e_h)$.

Theorem 4.22 Algorithm 4 returns a solution which is within a factor of $\frac{1}{3}$ from the optimal fractional solution of SEP-LP.

Proof. If the algorithm reached Step 9, then $C = E$, and the solution is optimal. Otherwise, denote by $w(ALG)$ the value of the solution output by Algorithm 4. Consider the following dual solution:

$$\beta_{ij} = w(s_i, s_j) \cdot x_{ij}, \alpha_{ij} = w(s_i, s_j) \cdot (1 - x_{ij}), \gamma = \Gamma_C(e_{|C|}).$$
Since \( \Gamma_C(e|C|) \geq \Gamma_C(e) \) for every \( e \notin C \), this is a feasible dual solution. Let \( z \) denote its value. Then,

\[
z = B \cdot \gamma + \sum_{s_i, s_j \in S} \beta_{ij} \tag{4.2}
\]

\[
= B \cdot \Gamma_C(e|C|) + \sum_{s_i, s_j \in S} w(s_i, s_j) \cdot x_{ij} \tag{4.3}
\]

\[
< \left( \sum_{h=0}^{\lvert C \rvert - 1} c(e_h) + c(e|C|) \right) \cdot \Gamma_C(e|C|) + w(C) \tag{4.4}
\]

\[
\leq \sum_{h=0}^{\lvert C \rvert - 1} c(e_h) \cdot \Gamma_C(e|C|) + w(\{e|C|\}) + w(C) \tag{4.5}
\]

\[
\leq \sum_{h=0}^{\lvert C \rvert - 1} c(e_h) \cdot \Gamma_C(e|C|) + w(\{e|C|\}) + w(C) \tag{4.6}
\]

\[
= w(C) + w(\{e|C|\}) + w(C) \tag{4.7}
\]

\[
\leq 3w(ALG), \tag{4.8}
\]

where: Inequality (4.4) follows from the definition of the Algorithm 4 and Observation 4.17, (4.5) follows from Corollary 4.19, (4.6) follows from Corollary 4.20 and (4.7) follows from Observation 4.21. Thus, the theorem follows by weak duality.

\[\blacksquare\]

### 4.4.3 An Approximation Algorithm for Weighted BSMC on General Graphs

In this subsection we present an \( (\frac{e}{e-1}, O(\log^2 n \log \log n)) \)-approximation algorithm for weighted BSMC.

In [66], Räcke describes a hierarchical decomposition of any undirected graph \( G = (V, E) \) into a tree \( T_G \), where there is a \( 1 - \frac{1}{e} \) correspondence between \( V \) and the leaves of \( T_G \). \( T_G \) has the property that any feasible multi-commodity flow function in \( T_G \) can be routed in \( G \) causing a congestion bounded by a function of \( G \)'s parameters, denoted by \( \beta \). By min-cut-max-flow theorems this implies a corresponding bounded ratio between the cost of cuts in \( G \) and the cost of cuts in \( T_G \). In [43], Harrelson, Hidrum and Rao give a polynomial-time construction of \( T_G \) with \( \beta = O(\log^2 n \log \log n) \), which we use in the following algorithm.

**Algorithm 5** A bi-criteria approximation algorithm for weighted BSMC

1. Let \( B' = 2\beta B \).
2. Construct a decomposition tree, \( T_G \), of \( G \).
3. for \( \forall e = (u, v) \in T_G \) with a cost \( > B' \) do
   4. Merge the vertices \( u \) and \( v \).
5. end for
6. Let \( T'_G \) be the resulted tree.
7. Run Algorithm 4 on \( T'_G \) with budget \( B' \), and output the associated cut in \( G \).

**Theorem 4.23** Algorithm 5 is a \( (\frac{e}{e-1}, O(\log^2 n \log \log n)) \)-approximation for the weighted BSMC problem.
Proof. Let $OPT$ be an optimal solution, and let $I$ denote the set of pairs of terminals separated by $OPT$. Let $OPT_{T_G}$ be a minimum cost cut separating $I$ in $T_G$. By [33], $c(OPT_{T_G}) \leq 2MCF_I(T_G)$, where $MCF_I(T_G)$ is the value of the maximum multi-commodity flow in $T_G$ between the pairs in $I$. By the construction of $T_G$ and its property, $MCF_I(T_G) \leq \beta MCF_I(G)$. Since $MCF_I(G)$ lower bounds the cost of any cut separating $I$ in $G$, $MCF_I(G) \leq c(OPT)$, and thus we get

$$c(OPT_{T_G}) \leq 2\beta c(OPT) \leq 2\beta B = B'. $$

In particular, $OPT_{T_G}$ does not contain any edge with cost more than $B'$, and thus $OPT_{T_G}$ is a feasible solution for the weighted BSMC problem on $T'G$ with budget $B'$, with value $w(OPT_{T_G}) \geq w(OPT)$. From [54], running Algorithm 4 will return a solution $C$ whose cost is at most $B'$ and whose value is at least $\frac{e-1}{e}w(OPT_{T_G})$. By the properties of the decomposition tree, the associated cut in $G$ has a cost of at most $B'$ and a separation weight of at least $w(C)$ and the theorem follows.

Since the weighted budgeted variant of Multicut is equivalent to weighted BSMC, we conclude that a bicriteria $(\frac{e}{e-1}, O(\log^2 n \log \log n))$-approximation exists for this problem as well.

### 4.5 The Budgeted Graph Disconnection Problem

The following algorithm for BGD is a variant of the algorithm presented in [69] for the $k$-cut problem. In what follows, we refer to the algorithm for the $k$-cut problem and its proof as they appear in [77][pp. 40-44].

#### Algorithm 6

A greedy algorithm for BGD

- Compute a Gomory-Hu tree $T$ for $G$.
- Sort the edges of $T$ in a non-decreasing order of their cost.
- Choose the maximal sequence of edges starting from the cheapest, whose cost is at most $B$, and output the union of the cuts associated with these edges in $G$, denoted by $C$.

#### Lemma 4.24

Let $\ell$ denote the value of an optimal solution. Algorithm 6 achieves an approximation factor of $\frac{1}{2} + \frac{1}{\ell}$ if $\ell$ is even, and $\frac{1}{2} + \frac{1}{2\ell}$ if $\ell$ is odd.

Proof. The algorithm for the $k$-cut problem [77] outputs the union of the lightest $k-1$ cuts of the cuts associated with edges of $T$ in $G$ and achieves an approximation factor of $2 - 2/k$. In particular, the cost of the lightest $k-1$ edges of $T$ is at most $(2 - 2/k)|OPT_k|$, where $|OPT_k|$ is the cost of an optimal $k$-cut. Assume without loss of generality that $OPT$ is an optimal $\ell$-cut. Then, the cost of the lightest $\ell - 1$ edges of $T$ is at most $(2 - 2/\ell)B$, and thus the cost of the lightest $\lfloor (\ell - 1)/(2 - 2/\ell) \rfloor = \lfloor \ell/2 \rfloor$ is at most $B$. Thus, $C$ is associated with at least $\lfloor \ell/2 \rfloor$ edges, and partitions $G$ to at least $\lfloor \ell/2 \rfloor + 1$ connected components. Noticing that the feasibility of $C$ follows from the properties of Gomory-Hu trees completes the proof.

We note that Example 4.9 in [77] shows that the above analysis is tight.
4.6 Further Discussion

Among the problems that were studied in this chapter, the weighted BSMC problem seems the hardest. It remains an open question whether it is possible to improve upon the bi-criteria approximation that we presented or even achieve a uni-criteria approximation. In this section we review some related ideas and point out some possible directions towards solving the problem.

4.6.1 The Budgeted Sparsest Cut Problem

Consider the following budget problem, whose input is the same as the input for the weighted BSMC problem.

**Problem 4.5 (Budgeted Sparsest Cut)** Find a non-empty subset of vertices \( U \subset V \) such that \( c(U, \overline{U}) \leq B \) and the sparsity of \((U, \overline{U})\) is minimized.

In order to understand the relation between weighted BSMC and Budgeted Sparsest Cut, we look for results similar to those of Subsection 4.4.1. Notice that the algorithm of Corollary 4.14 actually finds a cut whose cost is at most \( c(C) \). Hence, Theorem 4.15 can be easily generalized to obtain the following.

**Theorem 4.25** Let \( ALG \) be an \((\alpha, \beta)\)-approximation for weighted BSMC. Then, there exists a \(((1+\epsilon)\alpha\beta, \beta)\)-approximation for the Budgeted Sparsest Cut problem for every \( \epsilon > 0 \).

Specifically, notice that a uni-criteria approximation for weighted BSMC implies an appropriate uni-criteria approximation for the Budgeted Sparsest Cut problem.

4.6.2 Linear Programming Formulation Revisited

In Subsection 4.4.1 we introduced SEP-LP, a linear programming relaxation for weighted BSMC. As its integrality gap is \( \Omega(n) \), using SEP-LP to obtain a good approximation seems unlikely. Nevertheless, one might look for a different linear programming formulation with a smaller integrality gap. To this end, reconsider the example that was used to prove the integrality gap of SEP-LP in Proposition 4.16. Notice that the optimal fractional solution fractionally buys a cut whose cost is more than the given budget. Accordingly, adding a constraint to forbid solutions that buy expensive cuts might improve the linear programming formulation.

We consider the following modified version of SEP-LP, in which the variables \( x_{ij} \) and \( y_e \) have the same meaning, and we only replace the separation constraints \((x_{ij} \leq \sum_{e \in P_{i,j}} y_e)\). Let \( L = \{U \subseteq V \mid c(U, \overline{U}) \leq B\} \), i.e., \( U \in L \) if the cost of its associated cut is at most \( B \). We assign a variable \( z_U \) for every cut \( U \in L \). In what follows we show that the set of integer feasible solutions to SEP-LP2 corresponds to the set of the
feasible cuts for the BSMC problem.

\[
\begin{align*}
\max & \quad \sum_{s_i, s_j \in S} w(s_i, s_j) \cdot x_{ij} \\
\text{s.t.} & \quad x_{ij} - \frac{1}{2} \sum_{U \in L: |U \cap \{s_i, s_j\}| = 1} z_U \leq 0 \quad \text{for every } s_i, s_j \in S \\
& \quad \frac{1}{2} \sum_{U \in L: |U \cap e| = 1} z_U - y_e \leq 0 \quad \text{for every } e \in E \\
& \quad \sum_{e \in E} c(e) \cdot y_e \leq B \\
& \quad 0 \leq x_{ij} \leq 1 \\
& \quad 0 \leq y_e \\
& \quad 0 \leq z_U \\
& \quad \forall e \in L
\end{align*}
\]  
(SEP-LP2)

Let \( C \) be a feasible solution to a given BSMC instance. Denote by \( U_1, \ldots, U_k \) the set of connected components into which the graph is partitioned by \( C' \). Note that by feasibility of \( C, U_i \in L \) for \( 1 \leq i \leq k \). Set: \( y_e = 1 \) for every \( e \in C \) and \( y_e = 0 \) otherwise; \( x_{ij} = 1 \) for every \( i, j \) such that \( C \) separates \( s_i \) and \( s_j \) and \( x_{ij} = 0 \) otherwise; \( z_U = 1 \) for every \( U \in \{ U_i \}_{i=1}^k \) and \( z_U = 0 \) otherwise. Consider a pair of terminals separated by \( C, s_i \) and \( s_j \), and assume that \( s_i \in U_i \) and \( s_j \in U_j \). Then, since \( U_i \neq U_j \) we have that

\[
x_{ij} = 1 = \frac{1}{2} \cdot 2 = \frac{1}{2} \cdot (z_{U_i} + z_{U_j}) \leq \frac{1}{2} \sum_{U \in L: |U \cap \{s_i, s_j\}| = 1} z_U.
\]

Next, consider an edge \( e \in E \). If \( e \subseteq U_i \) for some \( 1 \leq i \leq k \), we have that \( \sum_{U \in L: |U \cap e| = 1} z_U = 0 \). Otherwise, \( e \) has one endpoint in \( U_i \) and one endpoint in \( U_j \), and specifically, \( e \in C \). Hence, we have that

\[
\frac{1}{2} \sum_{U \in L: |U \cap e| = 1} z_U = \frac{1}{2} \cdot (z_{U_i} + z_{U_j}) = 1 = y_e.
\]

We conclude that the above solution is a feasible integer solution for SEP-LP2 with the same value as \( C \).

Next, given \( \{x_{ij}\}_{s_i, s_j \in S}, \{y_e\}_{e \in E}, \{z_U\}_{U \in L} \), an integer feasible solution to SEP-LP2, consider the cut

\[
C = \bigcup_{z_U \geq 1} (U_i, \overline{U_i}).
\]

Let \( e \in C \). It follows by the definition of \( C \) that there exists \( U_i \) such that \( z_{U_i} \geq 1 \) and \( |U_i \cap e| = 1 \). Hence, by feasibility we have:

\[
\frac{1}{2} \leq \frac{1}{2} \cdot z_{U_i} \leq \frac{1}{2} \sum_{U \in L: |U \cap e| = 1} z_U \leq y_e,
\]

and thus, by integrality, \( y_e \geq 1 \). Accordingly, by feasibility we have that:

\[
\sum_{e \in C} c(e) \leq \sum_{e \in C} c(e) \cdot y_e \leq B,
\]

which means that \( C \) is a feasible solution. Consider a pair of terminal, \( s_i \) and \( s_j \), such that \( x_{ij} = 1 \). By feasibility and integrality, there exists \( U_i \) such that \( z_{U_i} \geq 1 \) and \((U_i, \overline{U_i})\) separates \( s_i \) and \( s_j \). It follows by
the definition of $C$ that $s_i$ and $s_j$ are also separated by $C$. We conclude that $C$ is a feasible solution whose value is at least the value of the given solution for SEP-LP2.

As the size of SEP-LP2 is exponential, the dual program should be considered.

$$\begin{align*}
\min & \quad B \cdot \gamma + \sum_{s_i, s_j \in S} \delta_{ij} \\
\text{s.t.} & \quad \alpha_{ij} + \delta_{ij} \geq w(s_i, s_j) \quad \text{for every } s_i, s_j \in S \\
& \quad c(e) \cdot \gamma - \beta_e \geq 0 \quad \text{for every } e \in E \\
& \quad \sum_{U \cap e = 1} \beta_e - \sum_{U \cap \{s_i, s_j\} = 1} \alpha_{ij} \geq 0 \quad \text{for every } U \in L \\
& \quad \alpha_{ij} \geq 0 \quad \text{for every } s_i, s_j \in S \\
& \quad \beta_e \geq 0 \quad \text{for every } e \in E \\
& \quad \gamma \geq 0 \\
& \quad \delta_{ij} \geq 0 \quad \text{for every } s_i, s_j \in S
\end{align*}$$

(SEP-DLP2)

In order to solve SEP-DLP2 a separation oracle is needed, as the number of constraints might be exponential. However, observe that such an oracle should find a cut whose sparsity (with respect to $\alpha_{ij}$ as pair weights and $\beta_e$ as edge costs) is less than $1$, among cuts belonging to $L$. This problem generalizes the budgeted sparsest cut problem, as it deals with finding a sparsest cut among a given set of cuts (here, the interesting cuts are not necessarily the cheap cuts with respect to the edge costs). It is reasonable to look for algorithms for weighted BSMC that use an approximation algorithm for this problem to obtain an approximate solution to SEP-DLP2 (and an appropriate solution to SEP-LP2). However, we did not succeed to prove a general theorem, analogous to Theorem 4.25, but only a result for a special case, which is presented in the following subsection.

4.6.3 Node-Weighted BSMC

The node-weighted BSMC problem is a special case of the weighted BSMC problem in which $w(s_i, s_j) = \varphi(s_i) \cdot \varphi(s_j)$, where $\varphi : V \rightarrow \mathbb{R}^+$ is a given node weight function. In what follows we assume without loss of generality that $\varphi$ is normalized so that the minimal positive weight of a node is exactly $1$.

**Theorem 4.26** Let $\text{ALG}$ be an $(\alpha, \beta)$-approximation for the budgeted sparsest cut problem. Then, there exists an $(O(\alpha), \beta + 1)$-approximation for the node-weighted BSMC problem.

**Proof.** Let $\Phi = \sum_{s_i \in S} \varphi(s_i)$, and notice that $\sum_{s_i \neq s_j} w(s_i, s_j) \leq \Phi^2/2$, hence $\Phi^2/2$ is an upper bound on the value of the optimal solution. We assume without loss of generality that for every pair $(s_i, s_j)$, there exists a separating cut whose cost is at most $B$. If that is not the case, such a pair of terminals can be merged into a new terminal $s'$ with $\varphi(s') = \varphi(s_i) + \varphi(s_j)$, as no feasible solution can separate such pair.

Consider the following iterative algorithm. In each iteration we run $\text{ALG}$ to find a “good” cut in some connected component and add its edges to the solution. If the solution’s cost exceeds the budget, we terminate with the current solution. In the first iteration, the input for $\text{ALG}$ is the given graph $G$, and the given budget $B$. The output of $\text{ALG}$ in the $i$th iteration is a cut $C_i$ that separates its input to two connected components. The connected component with the bigger node weight is the input for the next iteration, along with the budget $B$. Notice that the algorithm may also terminate if the current connected component contains only one vertex with a positive weight. Let $C$ be the output of the algorithm.
Obviously, the above algorithm runs in polynomial time, as the number of iteration is bounded by the number of edges in the graph. Furthermore, by the termination criteria and the input budget for ALG, \( c(C) \leq (\beta + 1) \cdot B \). Let \( OPT \) denote the optimal solution, and denote its separation weight by \( w(OPT) = z \cdot \Phi^2 \). Notice that as aforesaid \( z \leq 1/2 \). We also have that

\[
\forall s_i \in S, w(OPT) \leq \Phi \cdot (\Phi - \varphi(s_i)),
\]

as the optimal solution cannot do better than separating all the terminals. In what follows we prove that \( w(C) = \frac{w(OPT)}{O(\alpha)} \).

Let \( h \) denote the total number of iterations, and \( \gamma_i \cdot \Phi \) be the node weight of the connected component with the smaller node weight among the components that were separated during the \( i \)th iteration. Similarly, let \( \gamma_{h+1} \cdot \Phi \) be the node weight of the connected component with the bigger node weight among the components that were separated during the last iteration. Note that for \( i \leq h \), \( \sum_{j=i+1}^{h+1} \gamma_j \geq \gamma_i \) as among the two components that were separated during the \( i \)th iteration, \( \sum_{j=i+1}^{h+1} \gamma_j \) is the node weight of the bigger component while \( \gamma_i \) is the node weight of the smaller one.

If for some \( i \leq h \), \( \gamma_i \geq \frac{1}{4} \), then since \( \sum_{j=i+1}^{h+1} \gamma_j \geq \gamma_i \), we have that \( w(C) \geq \frac{\Phi^2}{16} \geq \frac{w(OPT)}{8} \) and the theorem follows. Otherwise, if \( \sum_{i=1}^{h} \gamma_i > \frac{z}{2} \), then

\[
w(C) = \frac{\Phi^2}{2} \cdot \sum_{i=1}^{h+1} \gamma_i \cdot (1 - \gamma_i) \geq \frac{\Phi^2}{2} \cdot \sum_{i=1}^{h} \gamma_i \cdot \frac{3}{4} \geq \frac{3\Phi^2 \cdot z}{16} = \frac{w(OPT)}{16/3},
\]

and the theorem follows.

Now, assume that \( \sum_{i=1}^{h} \gamma_i \leq \frac{z}{2} \) (so \( \gamma_{h+1} \geq 1 - \frac{z}{2} \geq \frac{3}{4} \)), and specifically that for every \( i \leq h \) it holds that \( \gamma_i < 1/4 \). The cut that is induced by \( OPT \) on the input graph of the \( i \)th iteration has a separation weight of at least \( w(OPT) - \Phi^2 \cdot \sum_{j=1}^{i-1} \gamma_j \), and its cost is at most \( B \). Hence, by the correctness of ALG,

\[
\frac{c(C_i)}{\gamma_i \cdot (1 - \sum_{j=1}^{i} \gamma_j)} \cdot \Phi^2 \leq \alpha \cdot \frac{B}{\Phi^2 \cdot (z - \sum_{j=1}^{i-1} \gamma_j)},
\]

which means that

\[
\gamma_i \cdot \left(1 - \sum_{j=1}^{i} \gamma_j\right) \geq \frac{c(C_i) \cdot (z - \sum_{j=1}^{i-1} \gamma_j)}{\alpha \cdot B}.
\]
By summing over all the iterations we have:

\[
\begin{align*}
    w(C) &= \sum_{i=1}^{h} \gamma_i \cdot \left(1 - \sum_{j=1}^{i} \gamma_j\right) \cdot \Phi^2 \\
    &\geq \frac{\Phi^2}{\alpha \cdot B} \cdot \sum_{i=1}^{h} c(C_i) \cdot \left(1 - \sum_{j=1}^{i-1} \gamma_j\right) \\
    &= \frac{\Phi^2}{\alpha \cdot B} \cdot \left(c(C) \cdot z - \sum_{i=1}^{h} c(C_i) \cdot \sum_{j=1}^{i-1} \gamma_j\right) \\
    &\geq \frac{\Phi^2 \cdot c(C)}{\alpha \cdot B} \cdot \left(z - \sum_{i=1}^{h-1} \gamma_i\right) \geq \frac{z \Phi^2 \cdot c(C)}{2\alpha \cdot B}.
\end{align*}
\]

We distinguish between the different terminations conditions. If it is the case that the solution cost exceeds the budget \(B\), then \(w(C) \geq \frac{w(OPT)}{2\alpha}\) and the theorem follows. Lastly, consider the case that the last connected component contains only one vertex with a positive weight. Since it follows from Inequality 4.9 that \(z \leq 1 - \gamma_{h+1}\), we have that

\[
w(C) \geq \Phi^2 \cdot \gamma_{h+1} \cdot (1 - \gamma_{h+1}) \geq \Phi^2 \cdot \gamma_{h+1} \cdot z \geq z \cdot \Phi^2 \cdot \frac{3}{4} = \frac{w(OPT)}{4/3}
\]

and the theorem follows. \[\blacksquare\]
Chapter 5

Improved Approximation Algorithms for the Spanning Star Forest Problem

In this chapter we study the spanning star forest problem. We present a 0.71-approximation algorithm for this problem, improving upon the approximation factor of 0.6 of Nguyen et al. [65]. We also present a 0.64-approximation algorithm for the problem on node-weighted graphs. Finally, we present improved hardness of approximation results for the weighted versions of the problem.\footnote{The results of this chapter were presented in a conference paper in [19].}

5.1 Introduction

A star graph is a tree of diameter at most two. Equivalently, a star graph consists of a vertex designated center along with a set of leaves adjacent to it. In particular, a singleton vertex is a star as well. Given an undirected graph, a spanning star forest consists of a set of node-disjoint stars that cover all the nodes in the graph. In the spanning star forest problem, the objective is to maximize the number of edges (or equivalently, leaves) present in the forest.

A dominating set of a graph is a subset of the vertices such that every other vertex is adjacent to a vertex in the dominating set. Observe that in a spanning star forest solution, each vertex is either a center or adjacent to a center. Hence the set of centers form a dominating set of the graph. Therefore, the size of the maximum spanning star forest is the number of vertices minus the size of the minimum dominating set. Computing the maximum spanning star forest of a graph is NP-hard because computing the minimum dominating set is NP-hard.

The spanning star forest problem has found applications in computational biology. Nguyen et al. [65] use the spanning star forest problem to give an algorithm for the problem of aligning multiple genomic sequences, which is a basic bioinformatics task in comparative genomics. The spanning star forest problem and its directed version have found applications in the comparison of phylogenetic trees [12] and the diversity problem in the automobile industry [1].

Surprisingly, even though the maximum spanning star forest is a natural NP-hard problem, there is not much literature on approximation algorithms for this problem. In fact, the first approximation algorithms
for this problem appeared recently in the work of Nguyen et al. [65]. They gave a number of approximation algorithms: the most general one being a 0.6-approximation algorithm on an unweighted graph. This should be contrasted with the complementary problem of minimizing the size of the dominating set of the graph which is known to be hard to approximate within a factor of $(1 - \epsilon) \ln n$ for any $\epsilon > 0$ unless NP is in $DTIME(n^{\log \log n})$ [27, 63]. This disparity in approximability of complementary problems is fairly commonplace (for example the maximum independent set is not approximable to within any polynomial factor while its complement problem of minimum vertex cover can be approximated to within a factor of 2). Nguyen et al. [65] also showed that the spanning star forest problem is hard to approximate to within a factor of $\frac{545}{546} + \epsilon$ unless P=NP. The paper also gave algorithms with better approximation factors for special graphs such as planar graphs and trees (in fact, for trees the optimal spanning star forest can be computed in linear time).

There are some natural weighted generalizations of the spanning star forest problem. The first generalization is when edges have weights and the objective is to maximize the weights of the edges in the spanning star forest solution. There is a simple 0.5-approximation algorithm for this case [65]. Note that the edge-weighted version is no longer the complement of the (weighted) dominating set problem. Another generalization is the case when nodes have weights. The objective now is to maximize the weights of nodes that are leaves in the spanning star forest solution. This problem is the natural complement of the weighted minimum dominating set problem. To the best of our knowledge, the approximability of the node-weighted spanning star forest problem has not been considered before.

5.1.1 Our Results and Techniques

We prove the following results in this chapter. First, we improve the result of [65] by giving a 0.71-approximation algorithm for the unweighted spanning star forest problem. Second, we give a 0.64-approximation algorithm for the node-weighted spanning star forest problem. Finally, we prove better hardness of approximation results for the weighted versions of the problem. In particular, we show that the node and edge-weighted spanning star forest problem cannot be approximated to within a factor of $\frac{31}{32} + \epsilon$ and $\frac{19}{20} + \epsilon$, respectively, for any $\epsilon > 0$ unless P=NP.

Our algorithms are based on an LP relaxation of the spanning star forest problem and randomized rounding. For each vertex we have a variable $x_i$ which is 1 if $x_i$ is a leaf. However, the natural rounding scheme of making vertex $i$ a leaf with probability $x_i$ does not give a good approximation ratio. Instead, we make vertex $i$ a leaf with probability $f(t, x_i) = e^{-t(1-x_i)}$, where the value of $t$ is carefully chosen. In fact the value of $t$ depends on the value of the optimal solution of the LP relaxation. Our rounding approach tries to extract as much information as possible from the LP relaxation, in particular, from the value of the optimal LP solution. Usually, the value of LP solution is used as an upper/lower bound to analyze the performance of the algorithm. We, in addition use it as a parameter of randomized rounding as well. Note that for fixed $t$, the function $f(t, x_i)$ is non-linear in $x_i$. Non-linear rounding schemes used in ([34, 56]) round with probability $x_i^c$, where $c$ is a fixed constant or is a value that depends on the input$^2$. An interesting point about the rounding is that the function $f(t, x_i)$ is nonzero even for $x_i = 0$, so with some low probability, the rounding can round a variable $x_i = 0$ to 1.

$^2$For the problem of maximum $k$-densest subgraph, a randomized rounding using $c = 0.5$ appears to be a folklore result that is attributed to Goemans [34].
The nonlinear rounding algorithm, obtains an approximation factor of $\ln \frac{n}{\text{OPT}} + O(1)$ for the dominating set problem, where $n$ is the number of vertices in the graph and $\text{OPT}$ is the value of the optimal (fractional) dominating set. This almost matches the best known approximation factor due to Slavík (for the more general set cover problem) [70].

However, the LP rounding only provides a 0.5 approximation, when the dominating set is large (say $0.5n$). To get the claimed factor of 0.71 for unweighted graphs, we use the LP algorithm in conjunction with another algorithm. The idea is to divide the input graph $G$ into the union of a subgraph $G'$ and some trees, where in $G'$ the minimum degree is at least 2. Given a spanning star forest solution for $G'$, we can “lift” back the solution to the original graph $G$. Then we use as a black box the algorithm from [65] that produces a spanning star forest of size at least $\frac{3}{5}n$ on an $n$-vertex graph of minimum degree 2.

We now turn to the node-weighted spanning star forest problem. Our LP rounding algorithm can be easily generalized to the node-weighted case. As in the unweighted case, the LP rounding algorithm by itself does not give us the stated factor of 0.64. To get the claimed approximation factor, we combine our rounding algorithm with the following trivial factor 0.5 algorithm: Compute any spanning tree, designate an arbitrary vertex as root. Divide the tree in to levels based on distance from the root. Make nodes at alternate levels as centers. It is easy to check that one of the two solutions will have weight at least $\frac{1}{2}$ times the sum of the weights of all nodes.

Finally, we turn to our hardness of approximation results. The hardness results are obtained by gadget reductions from the result of Håstad [44] that states that MAX3SAT is NP-hard to approximate to within a factor of $\frac{7}{8} + \epsilon$, for any $\epsilon > 0$, unless P=NP.

### 5.2 Preliminaries

In this chapter, we will consider undirected simple graphs that can be unweighted, node-weighted (where weights are on the nodes) or edge-weighted (where the weights are on the edges). Without loss of generality, assume that $G$ is connected, otherwise we can consider each connected components separately. We say a graph is a star if there is one vertex (called the center) incident to all edges in the graph (all other vertices are called leaves). The size of a star is the number of edges in the star (for weighted case, it is the sum of weights of the edges or the sum of the weights of the leaves in the star, for edge-weighted and node-weighted stars respectively). In particular, a singleton vertex is a star of size 0.

A spanning star forest of a graph $G$ is a collection of node disjoint stars that covers all vertices of $G$. The problem we are interested in is to find a spanning star forest that maximizes the sum of the sizes of its constituent stars. The unweighted, node-weighted and edge-weighted versions of the problem are denoted by Unweighted Spanning Star Forest, Node-Weighted Spanning Star Forest and Edge-Weighted Spanning Star Forest, respectively.

We will now fix some notation. Unless mentioned otherwise, a graph $G = (V, E)$ will be an unweighted graph. For a node-weighted graph, for any vertex $v_i \in V$, its weight will be denoted by $w_i \geq 0$. For an edge-weighted graph, for any edge $e \in E$, its weight will be denoted by $w_e \geq 0$. Further, for a vertex $v_i \in V$, $N(i)$ will denote the neighbor set of $v_i$ in $G$, that is, $N(i) = \{v_j \mid (v_i, v_j) \in E\}$. We will usually denote $|V|$ by $n$. By abuse of notation, we will use $OPT(G)$ to denote the optimal spanning star forest for $G$ as well as its the total size.
5.3 An LP-Based Algorithm

In this section we will present a linear programming based algorithm for the \textsc{Node-Weighted Spanning Star Forest} problem. Towards this, we define the following linear programming relaxation. For every vertex \(v_i\), the variable \(x_i\) has the following meaning: \(x_i = 1\) if \(v_i\) is a leaf in the spanning star forest and is 0 otherwise. For a vertex \(v_i\), it is not possible to have all vertices in \(N(i) \cup \{v_i\}\) as leaves. These constraints have been included in the linear program.

\[
\begin{align*}
\text{max} & \quad \sum_{v_i \in V} w_i \cdot x_i \\
\text{s.t.} & \quad x_i + \sum_{v_j \in N(i)} x_j \leq |N(i)|, \forall v_i \in V \\
& \quad 0 \leq x_i \leq 1, \forall v_i \in V
\end{align*}
\]

Let \(LP_{OPT}(G)\) be the value of the optimal solution of the LP. For the rest of the section, fix an optimal solution \(\{x_i\}_{i \in V}\). Let \(W = \sum_{i=1}^{n} w_i\) be the sum of the weights of all the nodes in \(G\). Define

\[
a = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i} = \frac{\sum_{i=1}^{n} w_i x_i}{W}.
\] (5.1)

Notice that this implies that the optimal objective value is \(aW\). Note that setting all \(x_i = 1/2\) gives a feasible solution with value \(W/2\). Thus, \(a \geq 1/2\). We will round the given optimal LP solution using the following rounding algorithm.

\begin{center}
\textbf{ROUNDING-ALG.}
\end{center}

1. Make vertex \(v_i\) a leaf with probability \(e^{-t(a)(1-x_i)}\), where \(t(a) = \frac{a}{2} \ln \left(\frac{1}{1-a}\right)\). (Note that as 1/2 \(\leq a < 1\), \(t(a) \geq 0\).)
2. Let \(L_1\) denote the set of vertices declared leaves in the first step.
3. Let \(L_2 = \{v_i \in V | v_i \cup N(i) \subseteq L_1\}\). Declare all vertices in \(L_1 \setminus L_2\) as leaves.
4. Assign every leaf vertex to one of its neighbors that is not declared a leaf. Ties are broken arbitrarily.

We have the following approximation guarantee for the above rounding algorithm.

**Lemma 5.1** Given an LP solution \(\{x_i\}_{i \in V}\), \textbf{ROUNDING-ALG} outputs a spanning star forest with expected size at least \(aW(1-a)^{\frac{1}{2}-1}\). That is, it is a \((1-a)^{\frac{1}{2}-1}\) factor approximation algorithm for the spanning star forest problem.
Proof. It is easy to verify that ROUNDED-ALG does indeed generate a valid spanning star forest. For notational convenience let $t = t(a)$ where $a$ is as defined in (5.1). Now the expected total weight of all leaves after step 2 of ROUNDED-ALG is

$$E(\ell_1) = \sum_{i=1}^{n} w_i e^{-t(1-x_i)} = e^{-t} W \left( \frac{\sum_{i=1}^{n} w_i e^{tx_i}}{W} \right)$$

$$\geq e^{-t} W (e^{a W}) e^{-t(1-a)} = W e^{-t(1-a)}$$

The inequality above is obtained by the fact that the arithmetic mean is larger than the geometric mean, and then using $\sum_{i=1}^{n} w_i x_i = a W$. Now after step 3, a vertex $v_i$ can cease to be a leaf with probability exactly

$$e^{-t(1-x_i)} \prod_{j \in N(i)} e^{-t(1-x_j)}.$$ 

Thus, if $\ell_2$ is the total weight of vertices that were leaves after step 2 but ceased to be leaves after step 3, then its expectation is given by

$$E(\ell_2) = \sum_{i=1}^{n} w_i \left( e^{-t(1-x_i)} \prod_{j \in N(i)} e^{-t(1-x_j)} \right)$$

$$= \sum_{i=1}^{n} w_i e^{-t} \left( e^{-t(|N(i)|-\sum_{j \in N(i)} x_j-x_i)} \right)$$

$$\leq \sum_{i=1}^{n} w_i e^{-t} = W e^{-t}$$

The inequality follows from the fact that the $x_i$'s form a feasible solution. Now the expected value of the solution produced by ROUNDED-ALG is the expected total weight of leaves at the end of step 3. In other words, the expected value is given by

$$E(\ell_1) - E(\ell_2) \geq W \left( \frac{e^{at} - 1}{e^t} \right)$$

Now substituting the value $t = \frac{1}{a} \ln \left( \frac{1}{1-a} \right)$ completes the proof. 

We have the following remarks concerning ROUNDED-ALG.

- The integrality gap of the LP is at most $3/4$: consider a 4-cycle. Note that setting all $x_i = 2/3$ is a valid solution, giving an LP optimal value of $8/3$. However, the integral optimum value is 2.

- The randomized rounding algorithm can easily be derandomized using the method of conditional expectations [3]. In fact, exact formulas for $E(\ell_1)$ and $E(\ell_2)$ are presented in the proof and the conditional expectations are easy to compute from these formulas.

- In the worst case where $a = 1/2$, the approximation ratio of ROUNDED-ALG for spanning star forest is rather bad (equal to 0.5). However, as we will see in the next two sections, we will take advantage
of Rounding-Alg to get good approximation algorithms.

### 5.3.1 Application of Rounding-Alg to Dominating Set

Observe that the approximation ratio in Lemma 5.1 improves as the value of \( a \) increases. In particular, the approximation ratio tends to 1 as \( a \) approaches 1. This suggests that the above rounding scheme yields an approximation algorithm for the complementary objective of minimizing the dominating set. In fact, by analyzing the behavior of the function as \( a \) approaches 1, we obtain the following result.

**Theorem 5.2** The Rounding-Alg computes a \( \left( \ln \frac{W}{OPT_f} + 1 + \frac{OPT_f}{W} \ln \frac{W}{OPT_f} \right) \) approximation ratio solution for the weighted dominating set problem, where \( OPT_f \) is the total weight of the optimal fractional dominating set solution.

**Proof.** Let the optimal LP value for the spanning star forest be given by \( aW \), where \( W \) is the sum of all the node weights. This implies that the optimal (fractional) dominating set has size \( OPT_f = (1 - a)W \).

Now, the dominating set returned by Rounding-Alg has size

\[
W - aW (1 - a)^{1/2 - 1} = OPT_f \cdot \frac{1 - (1 - a)^{1/2 - 1}a}{1 - a}
\]

Let \( a = 1 - \epsilon \). We have

\[
\frac{1 - (1 - a)^{1/2 - 1}a}{1 - a} = \frac{1 - \epsilon^{1/2 - 1}(1 - \epsilon)}{\epsilon} = \frac{1 - \epsilon^{1/2 - 1}}{\epsilon} + \epsilon^{1/2 - 1}
\]

As \( \epsilon < 1, \epsilon^{1/2 - 1} \leq 1 \). Thus, the approximation ratio (for the dominating set problem) is at most:

\[
\frac{1 - \epsilon^{1/2 - 1}}{\epsilon} + 1 = \frac{1 - \epsilon^{1/2 - 1} \ln \epsilon}{\epsilon} + 1 \leq \frac{1 - \left(1 + \frac{\epsilon}{1 - \epsilon} \ln \epsilon\right)}{\epsilon} + 1
\]

\[
= \frac{\epsilon^{1/2 - 1} \ln \frac{1}{\epsilon}}{\epsilon} + 1 \leq \frac{1}{\epsilon} (1 + 2\epsilon) + 1 = \frac{1}{\epsilon} + 2\epsilon + \frac{1}{\epsilon} + 1,
\]

where in the above we have used that since \( 0 < \epsilon \leq 1, \frac{\epsilon}{1 - \epsilon} \ln \frac{1}{\epsilon} < 1 \). Further, for any \( 0 < y < 1 \) and \( 0 < x \leq 1/2 \), we have the following inequalities: \( e^{-y} \geq 1 - y \) and \( \frac{1}{1 + x} \leq 1 + 2x \). Note that for our case we can always find a dominating set of size at most \( W/2 \), that is, \( \epsilon \leq 1/2 \). The proof is complete by noting that \( \epsilon = OPT_f/W \).

We remark that \( \epsilon = \frac{OPT_f}{W} \ln \frac{W}{OPT_f} \) in general is at most 1. However, if \( OPT_f = o(W) \), then \( \epsilon = o(1) \). This result is close to the best known bound of \( (OPT_f - \frac{1}{2}) \ln \frac{n}{OPT_f} + OPT_f \) from the analysis of greedy algorithm for set cover (and hence, applicable to dominating set too) in [70].

85
5.4 An Approximation Algorithm for the Unweighted Spanning Star Forest Problem

In this section, we will describe a 0.71-approximation algorithm for the Unweighted Spanning Star Forest problem. We will use the following two known results.

**Theorem 5.3 ([65])** For any connected unweighted graph $G$ of minimum degree at least 2, if the number of vertices $n \geq 8$, there is a polynomial time algorithm (denoted by ORACLE-ALG) to compute a spanning star forest of $G$ of size at least $3n/5$.

**Theorem 5.4 ([65])** For any tree $T$ rooted at $r$, let $OPT_{ct}(T)$ and $OPT_{lf}(T)$ be the optimal value of spanning star forest of $T$ given the condition that $r$ is declared a center and leaf, respectively. Then $OPT_{ct}(T)$ and $OPT_{lf}(T)$ can be computed in polynomial time.

Starting with the given connected graph $G$, we will generate a subgraph from $G$ recursively as follows: Whenever there is a vertex in the current graph of degree 1, remove the vertex and the edge incident to it from the graph. Denote the final resulting subgraph to be $G'$. Note that $G'$ is connected and every vertex in it has degree at least 2. Let

$$S = \{ v_i \in G' \mid \text{at least one edge incident to } v_i \text{ is dropped in the above process} \}.$$  

For simplicity, assume $S = \{ v_1, \ldots, v_h \}$ and let $(V(G) \setminus V(G')) \cup S$ denote the induced subgraph on the vertex set $(V(G) \setminus V(G')) \cup S$.

Consider the subgraph $(G \setminus G') \cup S$: it is easy to verify that $(G \setminus G') \cup S$ is composed of $h$ disconnected trees rooted at vertices in $S$. Denote these trees by $T_1, \ldots, T_h$, where the root of $T_j$ is $v_j$. Let $OPT_{ct}(T_j)$ and $OPT_{lf}(T_j)$ be the optimal value of spanning star forest for $T_j$ with the condition that $v_j$ is declared a center and leaf, respectively. According to Theorem 5.4, $OPT_{ct}(T_j)$ and $OPT_{lf}(T_j)$ can be computed in polynomial time. Define

$$S_1 = \{ v_j \in S \mid OPT_{ct}(T_j) < OPT_{lf}(T_j) \}$$

$$S_2 = \{ v_j \in S \mid OPT_{ct}(T_j) \geq OPT_{lf}(T_j) \}$$

Let $N'(S_2)$ be the set of neighbors of $S_2$ in $G'$. Observe that $|N'(S_2)| \geq 2$ (otherwise, all vertices in $S_2$ would have been removed earlier). Consider the subgraph $G' \setminus S_2$ and assume that there are $k$ vertices in $G' \setminus S_2$. We add two extra vertices $u$ and $v$ and connect $u$ and $v$ to all vertices in $N'(S_2)$. Let the resulting graph be $G^*$ (see Figure 5.1 for an example). Note that $G^*$ is a connected graph of minimum degree at least 2. Thus by Theorem 5.3, we can compute a spanning star forest of $G^*$ of size at least $\frac{3}{5} \cdot (k + 2)$ in polynomial time.

Now we are ready to describe our algorithm.
Figure 5.1: Illustration of Graph $G$ (left) and $G^*$ (right).

**TREECUTTING-ALG.**

1. For each $i \in S_2$, declare $i$ a center.
2. If the number of vertices in $G' \setminus S_2$ is smaller than (say) 1000,
3. compute the optimal spanning star forest of $G'$
given vertices in $S_2$ are centers.
4. Else,
5. compute spanning star forests of $G^*$ by **ORACLE-ALG** and
**ROUNDING-ALG.**
6. declare each $v_i \in G' \setminus S_2$ either a center of leaf according to
$$\max\{\text{ORACLE-ALG}(G^*), \text{ROUNDING-ALG}(G^*)\}.$$  
7. Given the choices made for the vertices in $S$, compute the best possible spanning star forest for $T_1,...,T_h$.

Note that all vertices in $S_2$ are declared centers. Thus, in Step 6, the declaration of each vertex $v_i \in G' \setminus S_2$ is feasible (it is either covered by another vertex in $G' \setminus S_2$ or by a vertex in $S_2$). Therefore, the algorithm outputs a feasible spanning star forest solution.
In the following discussions, let $\alpha(G)$ and $\beta(G)$ be the value returned by Oracle-ALG($G$) and Rounding-ALG($G$), respectively. It can be seen that

$$\text{TreeCutting-ALG}(G) \geq \max \{\alpha(G^*), \beta(G^*)\} - 2 + \sum_{v_i \in S_1} \text{OPT}(T_i \setminus v_i) + \sum_{v_j \in S_2} \text{OPT}_{ct}(T_j).$$

(5.2)

where “−2” is because in the worst case, both $w$ and $v$ are leaves in the output of Oracle-ALG($G^*$) or Rounding-ALG($G^*$), but they do not contribute to the solution of $G' \setminus S_2$.

Observe that for any graph $G''$ and any vertex $w \in G''$, given a spanning star forest solution where $w$ is a leaf, we can easily get a solution where $w$ is a center by switching the declaration of $w$ from leaf to center. Thus,

$$\text{OPT}(G'' | w \text{ is a center}) \geq \text{OPT}(G'' | w \text{ is a leaf}) - 1.$$ 

For any $v_j \in S_2$, note that

$$\text{OPT}(G' | v_j \text{ is a center}) + \text{OPT}_{ct}(T_j) \geq \text{OPT}(G' | v_j \text{ is a leaf}) - 1 + \text{OPT}_{ct}(T_j)$$

$$\geq \text{OPT}(G' | v_j \text{ is a leaf}) - 1 + \text{OPT}_{lf}(T_j),$$

where the second inequality follows from the definition of $S_2$. Therefore,

$$\text{OPT}(G) = \max \{\text{OPT}(G' | v_j \text{ is a center}) + \text{OPT}_{ct}(T_j), \text{OPT}(G' | v_j \text{ is a leaf}) + \text{OPT}_{lf}(T_j) - 1\}$$

$$= \text{OPT}(G' | v_j \text{ is a center}) + \text{OPT}_{ct}(T_j)$$

In other words, in the optimal solution of $G$, we can always assume vertices in $S_2$ are declared centers.

For any $v_i \in S_1$, we know essentially $\text{OPT}_{ct}(T_i) = \text{OPT}_{lf}(T_i) - 1$. Note that the root $v_i$ contributes zero to $\text{OPT}_{ct}(T_i)$ and one to $\text{OPT}_{lf}(T_i)$. That is, regardless of the contribution of $v_i$, the contribution of vertices in $T_i \setminus \{v_i\}$ in $\text{OPT}_{ct}(T_i)$ and $\text{OPT}_{lf}(T_i)$ is the same. In other words, for any declaration of $v_i$ (either center or leaf), we can always get the same optimal value for $T_i \setminus \{v_i\}$.

Therefore,

$$\text{OPT}(G) = \text{OPT}(G | \text{every } v_j \in S_2 \text{ is a center})$$

$$= \text{OPT}(G' | \text{every } v_j \in S_2 \text{ is a center})$$

$$+ \sum_{v_i \in S_1} \text{OPT}(T_i \setminus v_i) + \sum_{v_j \in S_2} \text{OPT}_{ct}(T_j).$$

(5.3)

Thus, when $k$ is small (i.e., TreeCutting-ALG goes through Step 2,3), where recall that $k$ is the number of vertices in $G' \setminus S_2$, TreeCutting-ALG($G$) = OPT($G$). Hence, we can assume that $k$ is large (i.e., TreeCutting-ALG goes through Step 4,5,6).
Assume that the optimal LP value satisfies $LP_{OPT}(G^*) = a \cdot (k + 2)$, where recall that $G^* = (G' \setminus S_2) \cup \{u, v\}$. Hence,

\[
\frac{\text{TreeCutting-Alg}(G)}{OPT(G)} \geq \max\{\alpha(G^*), \beta(G^*)\} - 2 + \sum_{v_i \in S_1} OPT(T_i \setminus v_i) + \sum_{v_j \in S_2} OPT_{ct}(T_j) \tag{5.4}
\]

\[
\geq \max\{\alpha(G^*), \beta(G^*)\} - 2 + \sum_{v_i \in S_1} OPT(T_i \setminus v_i) + \sum_{v_j \in S_2} OPT_{ct}(T_j) \tag{5.5}
\]

\[
\geq \frac{\max\{\alpha(G^*), \beta(G^*)\} - 2}{LP_{OPT}(G^*)} \tag{5.6}
\]

\[
\geq \max \left\{ \frac{\frac{3}{2}(k + 2)}{a \cdot (k + 2)} \cdot \frac{\beta(G^*)}{LP_{OPT}(G^*)} \right\} - \frac{2}{a \cdot (k + 2)} \tag{5.7}
\]

\[
= \max \left\{ \frac{0.6}{a} (1 - a)^{\frac{1}{a} - 1} \right\} - \frac{2}{a \cdot (k + 2)} \tag{5.8}
\]

\[
> 0.71 \tag{5.9}
\]

where (5.4) follows from (5.2) and (5.3), (5.5) follows from the fact that the summations are non negative, (5.6) follows from the fact that the LP optimal is larger than the integral optimal value, (5.7) follows from Theorem 5.3, (5.8) follows from Lemma 5.1, and (5.9) follows by an estimation using a computer aided numerical analysis (Figure 5.2).

![Figure 5.2](image-url)  

Figure 5.2: The approximation ratios for Oracle-Alg and Rounding-Alg. The horizontal line is 0.71.

In conclusion, we have the following result.

**Theorem 5.5** TreeCutting-Alg gives a $0.71$-approximation ratio solution for the Unweighted Spanning Star Forest problem.
5.5 An Approximation Algorithm for the Node-Weighted Spanning Star Forest Problem

In this section, we present a 0.64-approximation algorithm for the node-weighted spanning star forest problem. Consider the following simple algorithm.

**TRIVIAL-ALG**

1. Compute a spanning tree $T$ of the graph $G$, and pick an arbitrary vertex $r$ as its root. Let $h$ denote the height $T$ rooted at $r$. For each integer $k$, let $N_k$ denote the set of vertices at a distance of $k$ (in the tree) from the root $r$.

2. Output the spanning star forest with the higher weight of the following:
   - centers: $N_0 \cup N_2 \cup \ldots$, leaves: $N_1 \cup N_3 \cup \ldots$
   - centers: $N_1 \cup N_3 \cup \ldots$, leaves: $N_0 \cup N_2 \cup \ldots$

Essentially, the two spanning star forests are obtained by picking alternate levels in the spanning tree $T$.

It is easy to see that the following holds for TRIVIAL-ALG.

**Proposition 5.6** TRIVIAL-ALG always outputs a solution with value at least $W/2$.

**Theorem 5.7** There exists a polynomial time algorithm that solves the Node-Weighted Spanning Star Forest problem with an approximation factor of

$$\min_{a \in [1/2,1]} \max \left( \frac{1}{2a}, (1-a)^{\frac{1}{a}-1} \right) > 0.64$$

**Proof.** Consider the algorithm that runs TRIVIAL-ALG and ROUNding-ALG and picks the better of the two solutions—this algorithm obviously has polynomial running time. Let $aW$ denote the value of the LP optimum. From Proposition 5.6, the TRIVIAL-ALG produces a spanning star forest with weight at least $W/2$, and hence an approximation ratio of at least $\frac{W/2}{aW} = \frac{1}{2a}$. Clearly this also implies that $a > \frac{1}{2}$. The claim on the approximation ratio follows from Lemma 5.1. The lower bound on the ratio follows by an estimation using a computer aided numerical analysis (Figure 5.3).

5.6 Hardness of Approximation

The hardness results are obtained by a reduction from the following strong hardness for MAX3SAT.

**Theorem 5.8 ([44])** For every $\epsilon > 0$, given a 3-CNF formula $\phi$ it is $NP$-hard to distinguish between the following two cases:
There exists an assignment satisfying $1 - \epsilon$ fraction of the clauses in $\phi$.

No assignment satisfies more than $\frac{7}{8} + \epsilon$ fraction of the clauses in $\phi$.

Further, the hardness result holds even if each variable $x_i$ is constrained to appear positively and negatively an equal number of times, i.e. the literals $x_i, \overline{x}_i$ appear in equal number of clauses.

**Theorem 5.9** For any $\eta > 0$, it is $NP$-hard to approximate the Edge-Weighted Spanning Star Forest problem within $\frac{19}{20} + \eta$.

**Proof.** Let $\phi$ be a 3-CNF formula on $n$ variables $\{x_1, x_2, \ldots, x_n\}$. Further let $C_1, C_2, \ldots, C_m$ be the set of clauses in $\phi$. From Theorem 5.8, we can assume that each literal appears positively and negatively an equal number of times. For each $i$, let $d_i$ denote the number of clauses containing $x_i$ (respectively $\overline{x}_i$). Without loss of generality, we assume that $d_i \geq 2$ for all $i$. This can be achieved by just repeating the formula $\phi$ three times. A simple counting argument shows that $\sum_{i=1}^{n} d_i = \frac{3m}{2}$.

Create an edge-weighted graph $G_\phi$ as follows:

- Introduce one vertex $u_i$ for each literal $x_i$ and $v_i$ for literal $\overline{x}_i$, and one vertex $w_j$ for each clause $C_j$. Formally $V = \{u_1, \ldots, u_n\} \cup \{v_1, \ldots, v_n\} \cup \{w_1, \ldots, w_m\}$.

- Introduce an edge between $u_i$ and $w_j$, if clause $C_j$ contains literal $x_i$. Similarly, add an edge $(v_i, w_j)$ if clause $C_j$ contains literal $\overline{x}_i$. Furthermore, for all $i$, introduce an edge between $u_i$ and $v_i$. Formally, $E = \{(u_i, w_j) \mid C_j$ contains $x_i\} \cup \{(v_i, w_j) \mid C_j$ contains $\overline{x}_i\} \cup \{(u_1, v_1), \ldots, (u_n, v_n)\}$.

- For all $i$, the weight on the edge $(u_i, v_i)$ is equal to $d_i$. The rest of the edges have weight 1.

**Completeness:** Suppose there is an assignment to the variables $\{x_1, \ldots, x_n\}$ that satisfies $1 - \epsilon$ fraction of the clauses. Define a spanning star forest as follows:

- Centers: $\{u_i \mid x_i = true\} \cup \{v_i \mid x_i = false\} \cup \{C_j \mid C_j$ is not satisfied$\}$.
• Every satisfied clause $C_j$ contains at least one literal which is assigned true. Thus there is a center adjacent to each of the vertices $w_j$ corresponding to a satisfied clause. Since for each $i$, one of $u_i$ or $v_i$ is a center, the other vertex can be a leaf. Thus the set of leaves is given by: $\{u_i \mid x_i = false\} \cup \{v_i \mid x_i = true\} \cup \{w_j \mid C_j \text{ is satisfied}\}$.

Therefore, the total edge weight of the spanning star forest is given by

$$\sum_{i=1}^{n} d_i + |\{w_j \mid C_j \text{ is satisfied}\}| = \sum_{i=1}^{n} d_i + (1 - \epsilon)m = \frac{3m}{2} + (1 - \epsilon)m = \left(\frac{5}{2} - \epsilon\right)m.$$

**Soundness:** Consider the optimal spanning star forest solution $OPT$ of $G_\phi$. Without loss of generality, we can assume that for each $i$, exactly one of $\{u_i, v_i\}$ is a center, and the other is a leaf attached to it. This is because:

• If both $u_i$ and $v_i$ are centers, then modify the spanning star forest by deleting all the leaves attached to $v_i$, and making $v_i$ a leaf of $u_i$. The total weight of the spanning star forest solution does not decrease, since we delete at most $d_i$ edges of weight 1 and introduce an edge of weight $d_i$.

• If one of $u_i$ and $v_i$ is a center (say $u_i$) and the other (i.e. $v_i$) is a leaf but not attached to $u_i$, then we can disconnect $v_i$ from its center and attach it to $u_i$. This operation increases the weight of the spanning star forest by $d_i - 1$, which contradicts to the optimality of the solution.

• If both $u_i$ and $v_i$ are leaves, then making $u_i$ a center and attaching $v_i$ to it will increase the weight of the solution by $d_i - 2$, again a contradiction.

From the spanning star forest solution $OPT$, obtain an assignment to $\phi$ as follows: $x_i = true$ if $u_i$ is a center in $OPT$ and $x_i = false$ otherwise. If vertex $w_j$ is a leaf in $OPT$, then there is a center (say $u_i$) adjacent to it, which implies that clause $C_j$ is satisfied by the assignment of $x_i$. A similar argument applies when the vertex $w_j$ is adjacent to a center $v_i$. Therefore, the total weight of $OPT$ is given by

$$\sum_{i=1}^{n} d_i + |\{w_j \mid C_j \text{ is satisfied}\}| = \frac{3m}{2} + |\{w_j \mid C_j \text{ is satisfied}\}|$$

In particular, if at most $\left(\frac{7}{8} + \epsilon\right)$-fraction of the clauses in $\phi$ can be satisfied, then the weight of $OPT$ is at most $\frac{3m}{2} + \left(\frac{7}{8} + \epsilon\right)m = \left(\frac{19}{8} + \epsilon\right)m$.

From the completeness and soundness arguments, it is $NP$-hard to distinguish whether $G_\phi$ has a spanning star forest of weight $\left(\frac{7}{8} - \epsilon\right)m$ or $\left(\frac{19}{8} + \epsilon\right)m$. Thus it is $NP$-hard to approximate the Edge-Weighted Spanning Star Forest problem within a factor of $\left(\frac{19}{8} + \epsilon\right)/\left(\frac{7}{8} - \epsilon\right)$. The claim follows by picking a small enough $\epsilon$.

The proof of the next theorem is similar to the previous one.

**Theorem 5.10** For any $\eta > 0$, it is $NP$-hard to approximate the Node-Weighted Spanning Star Forest problem within $\frac{31}{32} + \eta$. 

92
Proof. Let $\phi$ be a 3-CNF formula on $n$ variables $\{x_1, x_2, \ldots, x_n\}$ and $m$ clauses $C_1, C_2, \ldots, C_m$. From Theorem 5.8, we can assume that each literal appears positively and negatively an equal number of times. For each $i$, let $d_i$ denote the number of clauses containing $x_i$ (respectively $\bar{x}_i$).

Create a node-weighted graph $G_\phi$ as follows:

- Introduce three vertices $a_i, u_i, v_i$ for each variable $x_i$, and one vertex $w_j$ for each clause $C_j$. Formally $V = \{a_1, \ldots, a_n\} \cup \{u_1, \ldots, u_n\} \cup \{v_1, \ldots, v_n\} \cup \{w_1, \ldots, w_m\}$.
- Introduce an edge between $u_i$ and $w_j$, if clause $C_j$ contains literal $x_i$. Similarly, add an edge $(v_i, w_j)$ if clause $C_j$ contains the literal $\bar{x}_i$. Furthermore, for all $i$, introduce edges $(a_i, u_i), (u_i, v_i), (v_i, a_i)$. Formally, $E = \{(u_i, w_j) \mid C_j$ contains $x_i\} \cup \{(v_i, w_j) \mid C_j$ contains $\bar{x}_i\} \cup \{(a_i, u_i), (u_i, v_i), (v_i, a_i), (a_i, u_n), (u_n, v_i), (v_n, a_i)\}$
- For all $i$, the weight of nodes $a_i, u_i, v_i$ is equal to $d_i$. The weight of the rest of nodes is 1.

Completeness: Suppose there is an assignment to the variables $\{x_1, \ldots, x_n\}$ that satisfies $1 - \epsilon$ fraction of the clauses. Define a spanning star forest solution as follows:

- Centers: $\{u_i \mid x_i = true\} \cup \{v_i \mid x_i = false\} \cup \{C_j \mid C_j$ is not satisfied$\}$.
- Every satisfied clause $C_j$ contains at least one literal which is assigned true. Thus there is a center adjacent to each of the vertex $w_j$ corresponding to a satisfied clause. Since for each $i$, one of $u_i$ or $v_i$ is a center, the other remaining two in $\{a_i, u_i, v_i\}$ can be leaves. Thus the set of leaves is given by : $\{u_i \mid x_i = false\} \cup \{v_i \mid x_i = true\} \cup \{w_j \mid C_j$ is satisfied$\} \cup \{a_i\}$.

The total node weight of the spanning star forest solution is given by

$$\sum_{i=1}^{n} 2d_i + |\{w_j \mid C_j$ is satisfied$\}| = \sum_{i=1}^{n} 2d_i + (1 - \epsilon)m = 3m + (1 - \epsilon)m = (4 - \epsilon)m.$$

Soundness: Consider the optimal spanning star forest solution $OPT$ of $G_\phi$. Without loss of generality, we can assume that for each $i$, exactly one of $\{u_i, v_i\}$ is a center in $OPT$. This is because:

- If both $u_i$ and $v_i$ are centers, then modify $OPT$ by deleting all the leaves of $v_i$, and making $v_i$ a leaf of $u_i$. The total weight of $OPT$ does not decrease, since we delete at most $d_i$ leaves from $v_i$ of weight 1 and introduce one new leaf of weight $d_i$.

- If both $u_i$ and $v_i$ are leaves, then $a_i$ has to be a center. Making $a_i$ a leaf and $u_i$ a center (and attaching $a_i$ to $u_i$) does not decrease the total weight of $OPT$.

From the spanning star forest solution $OPT$, we obtain a assignment to $\phi$ as follows: $x_i = true$ if $u_i$ is a center and $x_i = false$ otherwise. If a vertex $w_j$ is a leaf in $OPT$, then there is a center (say $u_i$) adjacent to it, which implies that $C_j$ is satisfied by the assignment of $x_i$. A similar argument applies when the vertex $w_j$ is adjacent to a center $v_i$. 

93
Note that at most two of the three nodes \( \{a_i, u_i, v_i\} \) can be leaves. Therefore the total weight of leaves in \( OPT \) is at most

\[
\sum_{i=1}^{n} 2d_i + |\{w_j \mid C_j \text{ is satisfied}\}| = 3m + |\{w_j \mid C_j \text{ is satisfied}\}|
\]

In particular, if at most \((\frac{7}{8} + \epsilon)\)-fraction of the clauses in \( \phi \) can be satisfied, then the weight of \( OPT \) is at most \( 3m + (\frac{7}{8} + \epsilon)m = (\frac{31}{8} + \epsilon)m \).

From the completeness and soundness arguments, it is \( NP \)-hard to distinguish whether \( G_\phi \) has a spanning star forest of weight \((4 - \epsilon)m\) or \((\frac{31}{8} + \epsilon)m\). Thus it is \( NP \)-hard to approximate the NODE-WEIGHTED SPANNING STAR FOREST problem to within a factor of \((\frac{31}{8} + \epsilon)/(4 - \epsilon)\). The claim follows by picking a small enough \( \epsilon \).

\[\Box\]

### 5.7 Future Research

Two main direction are open for future research. The approximability of the spanning star forest problem is still an open question. There is a gap in all three variants discussed in this paper. For the edge-weighted version, for example, the gap is between 0.5 and 0.95. The applicability of our rounding technique is of separate interest. Optimization problems for which the integrality gap of the appropriate LP does not match the approximation ratio obtained by the natural rounding scheme are good candidates to look at.
Bibliography


הינה لتמצאות זר ל棵树 (שהוא תונחי של הגורק G) המכיל את כל צמתי G ובו המספים הקשולים
מקסימליים. אנו מציגים אלגוריתם קירוב שעושה קירוב של 0.71, המשפר את ההוטוושה ביוויך
היחידה תוך כדי-env כ-6 קירוב של 0.6 (ראה [65]). נמוך-آن-ועט-ו-ברבאם
מספיקים בה לכל זמצ-ElementsByTagName-ה униים-הנה למדאעם זר ל棵树 המכיל את כל צמתי G
כשם משמאל, עלות העלום מקסימליים. בבראש זה אנו מציגים אלגוריתם קירוב המשיג 6 קירוב
של 0.64, לבעו, אנו ממשרים את תוצאת Kosten הקירוב העיתビュー לגרסת המימושקרים
בפועם זר ל棵树 המapsible.
The document contains a list of bullet points in Hebrew. Unfortunately, the text is not legible enough to transcribe accurately.
תחזיר ומרחב

בuvreודה וזע עסוקים בחקור קבלי התחנות הכרחיים של מערכות של מגדלי

בחקלא הראות לאعودة, לא תמميد עבש חסיכות בהמה מוספים שיקני, או סוכני, נמצאים
באנטרופיות המתקשות. השחקנים הרג יווים לכל חמיד מוהר, על תואר חמיד לא
שלייטו על בנו. יכלו תנועת השחקנים הולכי הבколоוש של כל השחקנים הספקיים. משג
הפתרון או חיוושי שיוויים של השחקנים, ישאר מימי "נסトン" שב אפ שחקן לא
שליםו משקל. ולצ לא לייל נצב בוט, יא תח NoSuchElementExceptionה אפשרית של הפרטים.
שנ约束כספים בטטרוNER שחקנייה ה órgãoיה של השחקנים. במקצת מעשים שיוויי השחקן
הויבא הרג מפקח או מצדו חכימה חכימה הכרחיות לשערoningen מסות. ה
בערודה 23 זה

עוסקים בשתי הש ¥אר הב ¥אר הב ¥אר:

• שיווי משקל במשתמשים מקובים – שחקנייה קובעיים ייום שחקנייה או שיטפונים המתקשים

מצבי השח ¥איבים קבלי התחנות מקובים או אנטריאקיים רכיבים סוכני מתפונים פעולות. הק
מהובדות בשיחים ובו פעולות שפה פעמים יותר. מים ידידיים בוברות או פואר סוכני שפה פעולות. הק
המכוניות ולהנהגת בוכ ממקונים. על פעולות שפה פעולות על ידי סוכני חכימה בברוالف
לכטשל, הש ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ ¥ $_
המחק طويل לחתוך, העמדה הרבות יוצרות מונח. ספי אוחרא, למידת רכבי, זכר משותף
המחק הוא מחנה המיתוג הקטש מברך. ספי קידם הבדור פיתוחו שלושה ושבעה
לשהרי ובארה התמורות אל נוזאר באלק. התבונת, העטרה, החכמה של ספי, בזון שתייה
רוחות החרות, היי קידם פופ.

אני מודה לدني רד. רשמי, אני לך התמחה של בברך שכנא את. משעשע - אפיל יותר. אני הראה
לצבר את התמחה המוחו של עולאם התאורה או לכל שכר את עולאם הפשית: אם ההרות ששלביך
העולמות נראת קשש, יהיה איפשרי.

כמו כן בברך להמחת להמחתי למדעי המחשב של האוניברסיטה של אוניברסיטת, שرافטו
אוני. 
כוסקט דאוד브 שכנא פגאלה. תודור מחונת_alert קרן יזיר דלדר שביבת אפי
ברזוסט ובעד הבולים השנה בין לי התנשה לא פועל. התודור הבויר מנוגרת אכדית
נעזרת, טעNEY תובתנה עם יונה החשש שפרישתת את דומן ואני ללמוד. אני לי התנדוב
לחסונת קרבא צוות עס אנא קלי. ספי אוחרא, בברזסוס אלבNOP. אני מודה
לחם של שום את החוזה להמחית כפי שיתיה, בזון יוגיש. על הכרいま המיתוג והחרבות
מלואיםầmרא שוי.

אני מודה לקבוצת החורות לפי הבוחר המחמק של מייקוספורט בברזסוד, ואתנוגנטון, על ארחא.
בקריו, בברוזאציה את עיינו ועונור. השראה מבחיית אכדית. אני מודה לקפוקות למדעי המחשב
בטרסלי על התמחה בשבע עמר התואר עלי. שייתיה לי בת בברך חלף משמעתי מתי
האכדית. תודור מיתוגת אללי בהמה וירדנה קותל על עזרות הרבד.

אם אני כי התמחתющихся על מחנה של פ숀יפי. תודור מיתוגיה למיכאל שפריאי יאשר עדכי.
אם אני כי התמחות מש ::= בעד זה מידוע התחלית בברזסוד ומלחתית. מייאל משושי למחק את
האופי המתרק ישאר לוהט, ולאי מסל סטר אוחרא, מילאלא או הרבד.

הtlementים המ𬯎רים את האחרי שייח קחשמ פשועות מדח הקדימים של החזון בני בו מודעה עצום
לזרוס אדנים, ספי, שמתופף למשרטר והידך המחכים לשון התתאה, היא התמי הארשון
לתקשים לאנשיו מצタイトル. חלק בכבד המחיה הבוערה של מחשבות לקדר יזיהו של משמש
מדדו ממוחש שקיבלה מתינו. בזון שיוו בו, הוא בברך, עדו יזיהו לקדר מושנשים
למשלופי, לכל שישו בבררציה. עדו עם הרזה хотя סומ/dataTablesות ולצבר.
כי אם הפך בבר剿ני בזון שחיית ראשון, תודור עלידוסית להחר את בי הזה והירה גכ
לאי ייב חומ, הוא פlesaiי או חזרתי. אני מודה לי לעשף את הרעבים ואת החשש בברך
למלימה.

רעיונות, אני תישאר התובה על דף אלה התמוך ממידים ארוכים ופורחים והקורות עמידים רביר.
ברזוסט ובעד הבולים שלגון גלוי. ספי פהיא, אראלי רודיא, ייב רב. משיה טוטלבול, בבר
סקסוס, דוד אטמבל, ייב בברדוגר עליא לייא-איא. לבוסס, את התבדל אליי פלא стрתובא. בורנני
להבזח למשפתות הירה ולצבר, לעיתם הכפיות הברה והשתוללות.

אני מודה לטניכן על התמחה הכפיות הברה שהשתוללת.
ייזום בסטיבאות מרובות סוכנים ואלגוריתמי
千伏וב לבעיית NP קשוח גורפים

תיבור על מחקר

לשם מילות חלKI של הדירישה לקבלי התיאור
דוקטור לפילוסופיה

רזי אנגלברג

הוגש לסכס ה tecnnii – מכון טכנולוגי לישראל
אדר תשמ"ט חיפה מרץ 2009
יזיוב ב氓יבואת מרובות סוכנין ואלגוריתמי
קירוב לאוניית NP קשים בגרפים

רועי אנגלברג