Algorithms for Networks With Uncertainty

by

Samuel Mitchell Haney

Department of Computer Science
Duke University

Date: ___________________

Approved:

_____________________
Debmalya Panigrahi, Supervisor

_____________________
Bruce Maggs

_____________________
Ashwin Machanavajjhala

_____________________
Pankaj K. Agarwal

_____________________
Rajmohan Rajaraman

Dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in the Department of Computer Science
in the Graduate School of Duke University
2019
Abstract

Algorithms for Networks With Uncertainty

by

Samuel Mitchell Haney

Department of Computer Science
Duke University

Date: __________________________

Approved:

______________________________
Debmalya Panigrahi, Supervisor

______________________________
Bruce Maggs

______________________________
Ashwin Machanavajjhala

______________________________
Pankaj K. Agarwal

______________________________
Rajmohan Rajaraman

An abstract of a dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science in the Graduate School of Duke University 2019
Abstract

In this dissertation, we study algorithmic problems motivated by the optimization of networks under uncertainty. We summarize our contributions:

- We propose and give algorithms for the \textit{all-or-one $k$-server}, a generalization of classical $k$-server problem. In the $k$-server problem, a set of $k$ servers must serve requests in a metric space with minimum movement. These requests arrive sequentially, and the algorithm does not have access to future requests. In the all-or-one $k$-server generalization, each request is either for a particular server, or is a general request that may be served by any server. We give an $O(\log k)$-competitive randomized algorithm for this problem on a uniform metric.

- Motivated by the problem of deploying distributed applications on a network, we initiate the algorithmic study of \textit{graph retraction}, which seeks a mapping of a graph to a subgraph so as to minimize the maximum stretch of any edge, subject to the constraint that each vertex in the subgraph is mapped to itself. When the subgraph is acyclic, the problem can be solved in polynomial time, and therefore we consider most basic hard instance of the problem: retraction to cycles. Our main results are an $O(\min\{k, \sqrt{n}\})$-approximation for retracting any graph on $n$ nodes to a cycle with $k$ nodes, and an optimal algorithm when the graph is planar.

- We study the symmetric matching interdiction problem. This problem can be
simply stated as follows: find a matching whose removal minimizes the size of the maximum matching in the remaining graph. We show that this problem is APX-hard, and obtain a 3/2-approximation algorithm. We additionally introduce symmetric interdiction as a general model. We give a general framework that relates optimization to symmetric interdiction for a broad class of optimization problems.

We are motivated by applications in traffic engineering, where a network operator wishes to route traffic in a datacenter, but cannot distinguish between malicious and legitimate traffic.

- We study the effect of strategic behavior on network design. In multicast and broadcast games, agents in a graph attempt to connect to a root node at minimum cost to themselves, sharing the cost of each edge along their path to the root equally with the other agents using the edge. Such games can have many Nash equilibria, and networks formed dynamically by these agents could end up in any one of these equilibria, and may be very expensive. The main open problem in this field has been to bound the ratio of least expensive Nash equilibrium to the least expensive network, called the price of stability (PoS).

We make progress towards a better price of stability bound for multicast games. In particular, we show a constant upper bound on the PoS of multicast games for quasi-bipartite graphs. These are graphs where all edges are between two terminals (as in broadcast games) or between a terminal and a nonterminal, but there is no edge between nonterminals. This represents a natural class of intermediate generality between broadcast and multicast games.
# Contents

Abstract iv

List of Tables ix

List of Figures x

Acknowledgements xi

1 Introduction 1

1.1 Overview of Results and Structure of Thesis 5

1.1.1 All-or-one $k$-server 5

1.1.2 Network Scheduling: Retraction 6

1.1.3 Denial of Service Attacks: Symmetric Interdiction 7

1.1.4 Network Design with Selfish Incentives: Multicast Games 8

1.2 Bibliographic Notes 9

2 All-or-One $k$-Server 10

2.1 Introduction 10

2.1.1 Problem Definition 11

2.1.2 Results and Chapter structure 11

2.1.3 Related Work 11

2.1.4 Techniques: All-or-one uniform $k$-server 12

2.2 Preliminaries 13

2.3 Deterministic algorithm for all-or-one uniform $k$-server 18
4.5 Randomized Symmetric Matching Interdiction 87
4.6 Other Problems: Acyclic Subgraph Interdiction 89

5 Price of Stability of Undirected Multicast Games 92
5.1 Introduction 92
  5.1.1 Our Results 94
  5.1.2 Related Work 95
  5.1.3 Chapter Roadmap 96
5.2 Overview of Techniques 96
5.3 Preliminaries 100
5.4 Algorithm 103
5.5 Analysis 110
  5.5.1 Termination 111
  5.5.2 Cost Analysis 121

6 Conclusions 127

A Representation as a convex combination 129

Bibliography 131

Biography 141
List of Tables

5.1 Single sink Network Design: Optimization and Cost sharing games. . 94
# List of Figures

2.1 Illustration of cases in the proof of Lemma 13 .......................... 37  
3.1 Embedding of points from proof of Lemma 18 .............................. 49  
3.2 Using non-intersecting curves to find an embedding from face $F$ to $H$. 55  
3.3 Iterative triangulation of faces. ........................................... 61  
4.1 Illustration of the construction of cycle edges. ............................ 80  
4.2 Illustration of the construction of clause edges. .......................... 80  
4.3 Illustration of the removal of matching $M_x$. ............................ 82  
4.4 Enumeration of satisfied clauses. ........................................... 83  
4.5 Enumeration of unsatisfied clauses. ....................................... 83  
4.6 Illustration of non-optimal matching. ..................................... 85  
4.7 Illustration of matching without adjacent clause edges. ............... 86  
4.8 Illustration of types of connected components of $G \setminus M$. ......... 87  
4.9 Tight lower bound for spanning tree interdiction. ....................... 91  
5.1 An example with a price of anarchy of $\Omega(n)$ ....................... 93  
5.2 Types of critical improving moves. ........................................ 104
Acknowledgements

Thank you to Debmalya Panigrahi for advising me during my PhD. Debmalya has been an excellent advisor and mentor, as well as an outstanding researcher. He has always been supportive of me, and willing to sacrifice his time to help me.

Thank you to the rest of my committee – Pankaj K. Agarwal, Ashwin Machanavajjhala, Bruce Maggs, and Rajmohan Rajaraman – for being my teachers and collaborators.

Thank you to all my collaborators, John Abowd, Bolin Ding, Rupert Freeman, Matthew Graham, Mark Kutzbach Mehraneh Liaee, Ashwin Machanavajjhala, Bruce Maggs, Debmalya Panigrahi, Rajmohan Rajaraman, Ravi Sundaram, and Lars Vilhuber, without whom this work would not have been possible.

Thank you to the department staff for their help with equipment, travel, and logistics. Thank you especially to Marilyn Butler for her help with countless issues, I don’t know what I would have done without her.

Thank you to my fellow students at Duke, especially Rupert, Nat, and Andrew for being friends, collaborators, and office mates.

Thank you to my advisor and teacher Ben Hescott for introducing me to computer science theory when I was an undergraduate at Tufts University, and helping me make the decision to attend graduate school.

Thank you to my family, Sue, Doug, and Amelia, for their love and support. I wouldn’t be where I am today without you.
Thank you Katie for your love, support, and patience throughout my time at Duke.
The Internet is one of the most important developments of the modern era. It has grown at a staggering rate over the last few decades: Cisco estimated in 2012 that more than 8 billion devices were connected to the internet, and that this number would increase to 50 billion by 2020 [1]. It is no surprise, then, that the study and optimization of networks like the ones that make up the internet has been a cornerstone of computer science research for decades. Algorithms for flows, cuts, and network design are some of most celebrated in the area of algorithmic research.

These standard network optimization problems, while important, can fail to take into account the challenges stemming from uncertainty in the modern internet. The standard model for network optimization assumes there is a single network operator who, given all the inputs to a problem, performs some optimization on the network. We consider two broad classes of uncertainty that deviate from this model. In the first, we assume there is a single network operator, but that she does not have access to future inputs to the optimization problem in advance. In the second, we consider settings with multiple agents, each concerned with optimizing some aspect of the network for themselves.
There are many models of uncertainty about future inputs, depending on what assumptions we make. For example, if the inputs follow a pattern we can use a stochastic model where inputs are drawn from a distribution. In this thesis, we take a more pessimistic approach to future inputs: that is, we assume nothing. One standard model for this level of pessimism is known as the online model. In this model, inputs arrive over time and the algorithm makes irrevocable decisions without access to future inputs. We evaluate these algorithms against the best an algorithm could have done had it known all inputs in advance. A wide range of problems have been considered in online settings, including job scheduling [64], $k$-server [113, 92, 58], set cover [4], network design [5], and many more. Another approach when future inputs are unknown is to simplify the problem instance so solutions can be found easily on future inputs. The challenge here is to perform the simplification in a way such that the solutions to the simplified problem are also good solutions to the original problem. Examples include low-distortion tree embeddings [50], where we want to embed a graph into a tree while approximately preserving distances, and sparsification [61], where we reduce the number of edges in a graph while approximately preserving the size of cuts.

Next, we consider uncertainty arising from settings with strategic behavior. These settings differ from our standard model of optimization by introducing multiple agents with different objectives. These settings can cause uncertainty in a few ways. First, agents may lie if we rely on them to report information about themselves, resulting in incorrect or incomplete information about the system, making optimization challenging. Next, even if we have full information about the agents, agents may not be satisfied with an optimized solution and want to deviate. In this case, agent decisions can produce a wide range of solutions of varying quality.

The network optimization problems we consider are motivated by problems faced in data centers and networks. These problems include routing traffic, scheduling
jobs, caching, and network design. This dissertation will take a theoretical approach to solving problems. The problems we solve will be mathematical models of our motivating scenarios, and we will use formal proofs to analyze the algorithms we give.

In the first part of the thesis, we consider caching and scheduling jobs on a network with uncertainty about the future. We study caching via $k$-server, a well-studied problem that models caching, paging, and service on a network. In the $k$-server problem, a set of $k$ servers must serve requests in a metric space with minimum movement. The $k$-server problem uses the previously mentioned online model, i.e. requests arrive over time and the algorithm serves the requests without knowledge of future requests. The $k$-server problem has been the subject of a massive amount of work (see Section 2.1.3 for a brief overview) both because of its applications, and as a proxy for studying the online model in general. Inspired by recent work on generalizations of $k$-server [87], we propose and give algorithms for a generalization in which servers have identities, and requests can be made for particular servers.

We next consider deploying distributed applications on a network. We model a scenario in which we want to deploy distributed jobs on a network, and certain parts of the job are constrained to run only on certain nodes of the network (e.g. because they require specific resources provided by those nodes). Our formalization models both the problem of scheduling a single job on a network and of representing a network with another (virtual) network. The latter case can be used to simplify the network topology to schedule future, unknown jobs more efficiently. Our model is a type of graph embedding problem, where we want to find a mapping from vertices of one graph to another. In our problem, we are given a partial map that must be extended to a full map while minimizing the maximum increase in distance between vertices. Graph embedding problems have received large amounts of attention, and our problem is closely related to other graph embedding problems. These problems
vary the constraints on how vertices are mapped, and the objective, and include 0-extension, the minimum bandwidth problem, and low-distortion embeddings. We refer the reader to Section 3.1.2 for an overview of these problems and related results.

In the second part of the thesis we consider routing problems in settings with strategic behavior. In the first setting, we consider routing in a datacenter in which some users act adversarially. In what are known as denial of service attacks, these users request traffic to be routed with the goal of preventing the routing of legitimate traffic requested by other users. We consider the problem of routing traffic in a datacenter where the operator cannot distinguish legitimate users from adversarial users. We model this problem as a type of interdiction problem, where the goal is to remove edges from a graph to minimize the optimal solution for some optimization problem on the resulting graph. Interdiction problems for various optimization problems have been studied extensively (see Section 4.1.2). Traditionally, in these problems we are constrained to only remove some fixed number of edges from the graph. We study a different interdiction problem in which the constraints on the edges we can remove from the graph are the same as the constraints for the optimization problem whose optimal solution we are trying to minimize (motivating the choice of the name symmetric interdiction for this variant).

Next, we consider how strategic incentives, even when agents are cooperative, can affect routing in a network. In particular, we consider routing trees that emerge when agents are allowed to choose their routing paths in a network, and each agent seeks to minimize its cost. These games, known as network design games, are special cases of congestion games [102], a well-studied class of games. The dynamics of agent decisions can produce various routing trees, of varying quality. The primary goal of research in this area is to analyze the possible routing trees that can be produced, and compare their costs to the cost of the best possible routing tree. We refer the reader to Section 5.1.2 for a summary of work on network design games. We study
equilibria in network design games with fair cost allocation, meaning that the cost of each edge is divided equally among all agents using the edge.

1.1 Overview of Results and Structure of Thesis

In this section, we describe our problems more formally and give a summary of our results in each instance.

1.1.1 All-or-one k-server

$k$-server is a classical problem in computer science and perhaps the best known example of a problem in an online setting, one of the most common models for optimization under uncertainty. In this setting, an algorithm must make irrevocable decisions without knowledge of future inputs. We judge the performance of such algorithms against the best an algorithm could do if it had access to all future inputs.

In the $k$-server problem, a set of $k$ servers have to serve online requests on a metric space while minimizing total movement. On a uniform metric, this problem is called online paging. These problems inherently assume that the servers are anonymous, i.e., they do not have individual identities. In many applications, however, the servers have distinct identities, and clients can insist that a request only be served by a particular server. We call this the all-or-one $k$-server problem. This is a special case of the generalized $k$-server problem, which is a far-reaching extension of $k$-server that has attracted attention recently [13].

Our main result is an $O(\log k)$-competitive randomized algorithm for the all-or-one $k$-server problem on the uniform metric. Unlike $k$-server, the all-or-one $k$-server problem does not lend itself immediately to a linear programming (LP) relaxation. Instead, we give an (integer) LP that is not a relaxation, but is faithful to the objective of the all-or-one $k$-server problem up to a constant factor on the uniform
metric. We then solve this LP online, and our main technical contribution is in designing an online rounding algorithm for this LP. We also obtain a $3k$-competitive deterministic algorithm for this problem. Both results are asymptotically tight.

Our results on all-or-one $k$-server appear in Chapter 2.

1.1.2 Network Scheduling: Retraction

We consider the following problem. Suppose we have a distributed job whose components must be assigned to run on nodes of a network. Certain components of the job require resources present only at certain nodes, and therefore must be assigned to those nodes. Our goal is to assign the remaining components of the job so that pairs of components that need to communicate with each other are assigned close to one another on the network. Alternatively, we can view this problem as finding a mapping to simulate one network with another.

Motivated by this question, we initiate the algorithmic study of graph retraction, which seeks a mapping of a graph to a subset of the vertices of the graph so as to minimize the maximum stretch of any edge, subject to the constraint that each vertex in this subset is mapped to itself.

To see the connection to our motivation, note that an equivalent formulation of the problem is to find a mapping of some graph $G$ (the distributed job and its communication requirements) to a graph $H$ (the network) such that the maximum stretch of any edge is minimized and some vertices of $G$ are constrained to be mapped to particular vertices of $H$.

In this thesis we consider retractions to cycles, since this is the first non-trivial formulation of the problem (the problem is solvable in polynomial time when the target graph is acyclic).

This problem has its roots in the rich theory of retraction of topological spaces, and has strong ties to well-studied metric embedding problems such as minimum
bandwidth and 0-extension.

Our first result is an $O(\min\{k, \sqrt{n}\})$-approximation for retracting any graph on $n$ nodes to a cycle with $k$ nodes. We also show a surprising connection to Sperner’s Lemma that rules out the possibility of improving this result without a novel lower bound on the optimal stretch. Nevertheless, if the problem is restricted to planar graphs, we show that we can overcome these integrality gaps using an optimal combinatorial algorithm, which is the technical centerpiece of the paper. Building on our planar graph algorithm, we also obtain a constant-factor approximation algorithm for retraction of points in the Euclidean plane to a uniform cycle.

These results appear in Chapter 3.

1.1.3 Denial of Service Attacks: Symmetric Interdiction

We study optimization in presence of malicious users in a setting with minimal information. We ask, how can a network operator route traffic in a datacenter when they cannot distinguish between malicious and legitimate traffic? In such a low-information setting, we consider minimizing regret: the amount of legitimate traffic that could have been routed, but was not. This motivates us to propose a class of problems that we call symmetric interdiction problems, where the goal is to remove a feasible solution such that the remaining maximum feasible solution is minimized.

We first give a general framework that relates optimization to symmetric interdiction for a broad class of optimization problems.

Next, we focus on the symmetric matching interdiction problem. This problem can be simply stated as follows: find a matching whose removal minimizes the size of the maximum matching in the remaining graph. We show that this problem is APX-hard, and obtain a $3/2$-approximation algorithm that improves on the approximation guarantee provided by the general framework.

These results appear in Chapter 4.
Finally, we study the effects of strategic behavior on the classic problem of network design. Broadly, the problem of network design is to create a routing network to satisfy some connectivity requirements between nodes, while minimizing an objective – typically the total cost. While this model assumes the network is designed by a central planner, we consider a strategic setting.

We assume the agents that make up the network choose their routing path, and each agent pays some cost that depends on this choice. More concretely, we suppose each agent wishes to connect to a predetermined root node. Each agent picks a path to the root and pays a fraction of the cost of each edge on the path, equal to the cost of the edge divided by the number of agents using that edge in their path. That is, the cost of each edge is split equally among all users of the edge.

Our goal is to compare the costs of stable states (i.e. states in which no agent can improve its cost by switching to a new path, called a Nash equilibrium), with the social optimum, i.e. the set of paths of minimum total cost. Two equilibria of particular interest are the most expensive and least expensive equilibria. The most expensive equilibrium can cost far more than the social optimum, and therefore research in this field has focused on the least expensive equilibrium.

Since the work of Anshelevich et al. [7] that introduced network design games, the main open problem in this field has been to determine the ratio between the best Nash equilibrium and the social optimum, known as the price of stability (PoS), of multicast games. For the special case of broadcast games (every vertex is a terminal, i.e., has an agent), a series of works culminated in a constant upper bound on the PoS ([56, 88, 23]). However, no significantly sub-logarithmic bound is known for multicast games. We make progress toward resolving this question by showing a constant upper bound on the PoS of multicast games for quasi-bipartite graphs.
These are graphs where all edges are between two terminals (as in broadcast games) or between a terminal and a nonterminal, but there is no edge between nonterminals. This represents a natural class of intermediate generality between broadcast and multicast games. In addition to the result itself, our techniques overcome some of the fundamental difficulties of analyzing the PoS of general multicast games, and are a promising step toward resolving this major open problem.

Our results are presented in Chapter 5.

1.2 Bibliographic Notes

The content presented in Chapters 2 and 3 is based on joint work with Mehraneh Liaee, Bruce M. Maggs, Debmalya Panigrahi, Rajmohan Rajaraman, and Ravi Sundaram. The content presented in Chapter 4 is joint work with Bruce M. Maggs, Biswaroop Maiti, Debmalya Panigrahi, Rajmohan Rajaraman, and Ravi Sundaram [72]. The content presented in Chapter 5 is joint work with Rupert Freeman and Debmalya Panigrahi [60].

My work on differential privacy completed during my PhD [71, 70] is not included in this dissertation.
2.1 Introduction

In the \textit{k-server} problem, requests arriving online on a metric space have to be served by a set of \( k \) servers while minimizing total movement. If the metric space is a uniform metric (i.e. a metric space where the distances between any two distinct points are the same), this problem is called \textit{online paging}. The online paging problem and the \( k \)-server problem have been two of the most prominent problems in online algorithms over the last 30 years.

These problems inherently assume that the servers do not have individual identities, i.e., are anonymous. In many applications of these problems, however, the servers do have distinct identities and clients can request specific servers if they so desire. For instance, consider the canonical application of dispatching technicians to address service calls. Typically, a client can request a specific technician, or alternatively ask for any available technician.
2.1.1 Problem Definition

We are given a metric space on a set of locations $Z$, and a set of $k$ servers $S$. At each time step we get an online request of the form $(z, q)$, where $z \in Z$ is the location of the request, and $q \in \{\ast\} \cup S$. The algorithm must serve the request by moving either any server to $z$ in the case that $q = \ast$, and by moving server $s$ to $z$ in the case that $q = s$. We call this problem all-or-one $k$-server.

2.1.2 Results and Chapter structure

In this chapter we present the following results:

- In Section 2.3, we give an $O(3k)$-competitive deterministic algorithm for all-or-one $k$-server on a uniform metric.

- In Section 2.4 we show a $(2k - 1)$ lower bound on the competitive ratio for deterministic algorithms, and a quadratic lower bound on the competitive ratio of the natural generalization of the popular LRU algorithm to our problem.

- In Section 2.5 We give an $O(\log k)$-competitive randomized algorithm for all-or-one $k$-server on a uniform metric, which is asymptotically tight.

2.1.3 Related Work

In online paging, several $k$-competitive deterministic algorithms were obtained by Sleator and Tarjan [113] and this bound is tight. In $k$-server, after a long line of work [58, 66, 18], Koutsoupias and Papadimitriou gave a $(2k - 1)$-competitive deterministic algorithm, which remains the best result for general metric spaces to date. (Better bounds are known for special metrics such as the line and the tree, e.g., [39, 40].) Allowing randomization in the algorithm leads to an exponential improvement in the competitive ratio of these problems. For online paging, several $O(\log k)$-competitive algorithms are known [57, 94, 2, 11], and this bound is also
(asymptotically) tight. Progress in $k$-server has been slower [25, 108, 17, 19, 11, 44, 12] but several recent breakthroughs [10, 29, 89, 14] have now attained a competitive ratio of polylog($k$) for this problem.

The all-or-one $k$-server problem is a (very) special case of a far-reaching extension of $k$-server called generalized $k$-server introduced by Koutsoupias and Taylor [87, 110, 111, 112]. In this problem, each server resides in its own metric space and a request comprises a location on each metric space. Serving the request amounts to moving any one of the servers to the request location on its own metric. Interestingly, this problem has an exponential separation with $k$-server, even if each server resides on a uniform metric. All-or-one $k$-server is also a special case of the classic problems of metrical task (MTS) and metrical service systems (MSS) [17, 26, 27, 41].

2.1.4 Techniques: All-or-one uniform $k$-server

Here, we briefly describe our techniques for our main result: the randomized algorithm for all-or-one $k$-server. Recent research in online paging and $k$-server has been driven by a general recipe for solving online problems where a covering LP relaxation of the problem is solved and rounded online (see [30]). Indeed, for online paging, such a covering LP relaxation was given by Young [121], which was used later to obtain randomized algorithms for online paging (and online weighted paging) by Bansal, Buchbinder, and Naor [11] using the above technique. The same high level plan extends to $k$-server as well [10], although both solving the LP relaxation and rounding it online are much more challenging for general metrics.

In contrast, we do not know of an LP relaxation of the all-or-one $k$-server problem that can be solved online. The main difficulty is that server identities introduce packing constraints that prevent a monotone online solution. Instead, we circumvent this difficulty by introducing a different LP that is not a relaxation of the all-or-one $k$-server problem, but can be solved online. Importantly, we show that given an
integer solution to this LP, we can obtain a feasible solution for the all-or-one \( k \)-server problem, and vice-versa, losing only a constant factor in both directions. We give a (standard) online algorithm to obtain a fractional solution to this LP, but the main technical work of our paper goes into designing an online randomized rounding algorithm that converts the fractional solution to an integer one.

A remark about the LP formulation is in order here. The fact that the LP represents the all-or-one \( k \)-server objective up to a constant heavily relies on the metric being uniform. In general, the gap between the two can be as large as \( O(\Delta) \), where \( \Delta \) is the aspect ratio of the metric space. So, even with the recent developments in the \( k \)-server problem, there are fundamental hurdles to extending our results to general metrics. As a first step, an interesting future research direction would be to obtain an LP relaxation of this problem that can be solved online, for any metric space (or for hierarchical well-separated trees (HSTs), which are equivalent up to a logarithmic factor by standard results in metric embeddings [16, 51]).

2.2 Preliminaries

We start by recalling the definition of the all-or-one \( k \)-server problem on a uniform metric.

*Problem Definition*

Let \( Z = \{z_1, \ldots, z_n\} \) be a set of locations in a uniform metric. Let \( S = \{s_1, \ldots, s_k\} \) be the set of servers.

We call the pair \((z, q)\) a *demand*, where \( z \in Z \) is a location and \( q \in Q = S \cup \{*\} \) is a request type. A *request* is a time-indexed demand, namely the request at time \( t \), denoted \( r_t \), is for some location \( z \) and is of some type \( q \), which we denote by \( \text{dem}(r_t) = (z, q) \). Note the difference between a demand and a request: if multiple requests at different times have the same location and type, they correspond to the
same demand.

We define a server configuration to be a function mapping servers to locations in the metric. A server \( s \) can serve a request \( r_t \) with \( dem(r_t) = (z, q) \) if the server configuration at time \( t \) maps \( s \) to location \( z \), and \( dem(r_t) = (z, *) \) or \( dem(r_t) = (z, s) \). We call a demand \((z, s)\) for \( s \in S \) specific, and a demand \((z, *)\) unspecified. We call a request \( r_t \) specific if \( dem(r_t) \) is specific, and unspecified if \( dem(r_t) \) is unspecified.

A solution to all-or-one uniform-metric \( k \)-server is a sequence of server configurations, one for each time \( t \), such that the server configuration at time \( t \) can serve request \( r_t \). This problem must be solved online; that is, requests are revealed one by one, and we must produce a new server configuration to serve the request at time \( t \) before proceeding to the next time step. Additionally, previous configurations cannot be modified. The cost associated with changing from a configuration at time \( t \) to time \( t + 1 \) is the number of servers whose locations change. Our goal is to produce the sequence of server configurations with minimum cost.

**Independence, representatives, and demand sets**

Our general approach to giving an algorithm that produces this sequence of server configurations has two parts:

- First, we give an algorithm that, given requests \( r_1, \ldots, r_T \), produces a set of demands \( Y_t \) at each \( t \), such that \( dem(r_t) \in Y_t \). Note that \( Y_t \) will contain demands corresponding to requests that arrived before time \( t \), in addition to the new request that arrived at time \( t \).

- Next, given a set of demands \( Y_t \) at each time \( t \), we produce a sequence of server configurations such that any possible request \( r_t \) with \( dem(r_t) \in Y_t \) can be served by the server configuration at time \( t \).

The second part, i.e., producing a server configuration from the sequences of sets
of demands, appears in this section. Sections 2.3 and 2.5 give different algorithms for implementing the first part, i.e., producing the sequence of sets of demands from a sequence of requests.

There are two main details in this approach that we need to address. First, the sets of demands must have some limitations. A set $Y_t$ cannot contain every possible demand, because a single server configuration cannot serve any set of demands. Second, for some sequence of sets of demands, we need a way of accounting for the cost of the sequence of server configurations produced from the sets of demands.

We first discuss the limitations on $Y_t$. We say that a server $s$ in a server configuration covers a demand $(z, q)$, if $s$ can serve any request $r$ with $dem(r) = (z, q)$, i.e., $s$ is at location $z$ and $q = s$ or $q = \ast$. We say that a server configuration covers a set of demands, if for each demand, the configuration contains some server that covers the demand. Then, we are only interested in sets of demands that can be covered by a server configuration. To determine whether there exists a server configuration that covers $Y$, we need to know what subsets of demands can be covered by a single server. We call a set of demands independent if for any server configuration, no two demands are covered by the same server. Equivalently, a set of demands is not independent if and only if it contains a pair of demands $(z, q)$ and $(z', q')$ with $z = z'$ and $q = \ast$.

We next claim that for any set of demands $Y$, there is a unique independent subset of $Y$, which we call the representatives of $Y$ or $rep(Y)$, such that there exists a server configuration that covers $Y$ if and only if there exists a server configuration that covers $rep(Y)$.

Let $rep(Y)$ be defined as follows: for each location $z$ in $Y$, take all specific demands at location $z$, if they exist, to be in $rep(Y)$. If there is no specific demand at location $z$ take the unspecified demand at location $z$ if it exists. Note $rep(Y)$ is independent since we never select both a specific and unspecified demand at the
same location.

We next show that this definition of $rep(Y)$ gives us the desired property:

**Fact 1.** A configuration of servers covers a set of demands $Y$ if and only if it covers $rep(Y)$.

*Proof.* Suppose $(z, q) \notin rep(Y)$. Then by definition, $q = *$ and there exists and $s \in S$ with $(z, s) \in rep(Y)$. Since $(z, s)$ is covered, the configuration must have server $s$ in location $z$, and this server covers $(z, *)$. \qed

Finally, we show the conditions $Y$ needs to satisfy if there is a server configuration covering $Y$. We call a set of requests $Y$ a demand set if there is a server configuration that covers $Y$. This is equivalent to the following pair of conditions:

**Lemma 2.** A set of demands $Y$ is a demand set if and only if the following two properties hold:

**Property 1** The number of representatives of $Y$ is at most $k$, i.e. $|rep(Y)| \leq k$.

**Property 2** For any server $s \in S$ and locations $z, z' \in Z$ with $z \neq z'$, $(z, s)$ and $(z', s)$ are not both in $Y$.

*Proof of Lemma 2.* First, suppose $Y$ does not satisfy at least one of the above properties. We show there is no configuration of servers that can cover all demands of $Y$. If Property 1 is violated, $|rep(Y)| \geq k + 1$. Since the set of representatives is independent, no server can cover two demands of $rep(Y)$, and therefore it would take at least $k + 1$ servers to cover the set.

If Property 2 is violated, there are a pair of demands $(z, s)$ and $(z', s)$ for some server $s$ and $z \neq z'$. Each of these demands can only be covered by server $s$, and $s$ cannot cover both since they are at distinct locations.
In the other direction, suppose $Y$ satisfies both of the properties. We give a configuration of servers that covers $Y$. By Fact 1, as long as we cover $\text{rep}(Y)$, $Y$ will be covered. We construct a configuration to cover $\text{rep}(Y)$: For each $(z, s) \in \text{rep}(Y)$ where $s \in S$, we assign server $s$ to location $z$ in the configuration. We can do this by Property 2, since there is only a single location for each specific request for $s$. Suppose $k'$ is the number of servers remaining after we have assigned servers to specific requests. The number of unspecified requests in $\text{rep}(Y)$ is at most $k'$ by Property 1 (and since every specific request is a representative), and therefore we can arbitrarily assign the remaining $k'$ servers to each cover one of these general requests.

To complete the section, we give an algorithm in Lemma 3 that produces server configurations from demand sets.

**Lemma 3.** Let $Y$ and $Y'$ be two demand sets. Let $r_s$ denote the number of specific demands in $\text{rep}(Y') \setminus Y$ and $r_u$ denote the number of unspecified demands in $\text{rep}(Y') \setminus Y$. For any configuration of servers covering $Y$, there is a set of server moves of total at most $2r_s + r_u$ such that the new configuration covers $Y'$.

**Proof of Lemma 3.** Suppose we have a function $\text{cover}(H, y)$ that, given a demand set $H$ covered by the current server configuration, updates the server configuration to cover $H \cup \{y\}$ at a total cost of 2 server moves if $y$ is specific and 1 move if $y$ is unspecified. We first show how to use $\text{cover}(H, y)$ to prove the lemma, then we give a definition of $\text{cover}(H, y)$.

We initially set $H = Y \cap Y'$. Since the given configuration of servers covers $Y$, it must also cover $H$. We pick an arbitrary $y \in \text{rep}(Y')$, $y \notin H$, and run $\text{cover}(H, y)$. We update $H \leftarrow H \cup \{y\}$. If $\text{rep}(Y') \subseteq H$, then $\text{rep}(Y')$ is covered by the server configuration, and there by Fact 1 so is $Y'$, and we terminate. Otherwise, repeat
with an arbitrary \( y \in \text{rep}(Y') \), \( y \not\in H \). Note that \( H \) remains a demand set, since \( H \cup \{y\} \subseteq Y' \) and \( Y' \) is a demand set.

This loop runs at most \(|\text{rep}(Y')\setminus(Y \cap Y')| = |\text{rep}(Y')\setminus Y|\) times, and each time runs \( \text{cover}(H, y) \) once for a total cost of \( 2 \cdot |\text{rep}(Y')\setminus Y| \).

Finally we give the definition of \( \text{cover}(H, y) \). If \( y \) is already covered by the current server configuration, we are done. If not, then by Fact 1, \( y \in \text{rep}(H \cup \{y\}) \). Consider the set \( \text{rep}(H \cup \{y\})\setminus\{y\} \). \( |\text{rep}(H \cup \{y\})\setminus\{y\}| \leq k \) by Property 1, and therefore \( |\text{rep}(H \cup \{y\})\setminus\{y\}| \leq k - 1 \). \( \text{rep}(H \cup \{y\})\setminus\{y\} \) is covered by the current server configuration, and since this set is independent, there is a set of \( M \) of \( |\text{rep}(H \cup \{y\})\setminus\{y\}| = k - 1 \) servers in the current configuration covering \( \text{rep}(H \cup \{y\})\setminus\{y\} \). Therefore, there is some is some server \( s \notin M \) that can be moved without uncovering any demand of \( \text{rep}(H \cup \{y\})\setminus\{y\} \). Let \( y = (z, q) \) If \( q = * \) or \( q = s \), move server \( s \) to location \( z \) to cover \( y \). \( \text{rep}(H \cup \{y\}) \), and therefore \( H \cup \{y\} \) is now covered by \( M \cup s \), and we have made one server move.

Otherwise, \( q = s' \) for some server \( s' \neq s \), and server \( s' \) is at location \( z' \). Move \( s' \) to location \( z \) and move \( s \) to location \( z' \). Again, we claim that \( \text{rep}(H \cup \{y\}) \) is now covered by \( M \cup \{s\} \): \( y \) is covered by \( s' \), and \( s \) covers \((z', *)\) if \((z', *) \in H \). The only other request that could have been uncovered by moving \( s' \) out of location \( z' \) is \((z', s')\). However, by Property 2 this is impossible since \((z', s')\) and \((z, s')\) cannot both be in \( H \), as \( H \) is a demand set. In this case, we have moved at most 2 servers.

### 2.3 Deterministic algorithm for all-or-one uniform \( k \)-server

In this section, we give a 3\( k \)-competitive deterministic algorithm using the analysis framework from Section 2.2. The high-level goal is to give a sequence of demand sets \( Y_1, \ldots, Y_T \) such that \( \text{dem}(r_t) \in Y_t \) for all \( t \), and use Lemma 3 to produce a sequence of server configurations from the demand sets. We suppose that in the initial configuration, all servers start at a location that will never be requested. We
let $Y_0 = \{\}$ be a dummy demand set for this initial configuration of servers. Note that $Y_0$ is a valid demand set, and $Y_0$ is covered by the initial server configuration. We next show how to produce $Y_t$ from $Y_{t-1}$. Let $dem(r_t) = (z,q)$. If $Y_{t-1} \cup \{(z,q)\}$ is valid demand set, set $Y_t = Y_{t-1} \cup \{(z,q)\}$. Otherwise, we set $Y_t = \{(z,q)\}$. The initial server configuration covers demand set $Y_0$, and we can apply Lemma 3 to produce the $t$th server configuration from the $(t-1)$th configuration. Since $dem(r_t) \in Y_t$ and the $t$th server configuration covers $Y_t$, the configuration can also serve request $r_t$.

We conclude by bounding the cost against OPT. We divide the sequence of requests in rounds. A new round begins each time $Y_t$ is reset to contain a single element. We bound the cost of our algorithm against the optimal solution for each round.

**Lemma 4.** *The total cost of the above algorithm in each round is at most $3k$.***

*Proof.* To prove the lemma, we must show that over the course of a round, we add at most $k$ specific demands and at most $k$ unspecified demands to the set of representatives of the demand set. Using Lemma 3 gives a schedule with desired cost $3k$.

By Lemma 2, the demand set cannot contain more than $k$ representatives. Recall that any specific request is always a representative. Since we only add demands to the demand set over the course of the round, any specific request that becomes a representative remains a representative for the round. This shows that a total of $k$ specified demands can added to the representatives.

Next, note that each time an unspecified demand that had been a representative ceases to be one, it is because a specific demand at the same location is added to the set. This specific demand is a representative, and remains a representative for the rest of the round. The number of representatives in the end can be no more than $k$, so there can be a total of at most $k$ unspecified demands that are ever representatives.
during the round.

**Theorem 5.** The above algorithm is $3k$-competitive.

*Proof.* We show that the optimal solution pays a cost of 1 for each round. Without loss of generality, suppose that in the optimal solution, if a server needs to move, it does so at the earliest possible time. At the end of each round, adding one more element to the demand set gives a set of requests that is not a demand set, i.e., the set cannot be covered by any configuration of servers. Therefore, to cover this element, OPT must necessarily move at least one server. Because we have assumed OPT moves servers at the earliest possible time, this server move must happen before the end of the round. Therefore, OPT pays a cost of at least one per round, and the competitive ratio of the algorithm is $4k$. □

2.4 Deterministic lower bounds for all-or-one uniform $k$-server

We present a lower bound for deterministic online algorithms for the all-or-one $k$-server problem. We begin with a quadratic lower bound on the performance of a natural generalization of LRU.

*Lower bound on LRU*

Consider the following LRU algorithm for the all-or-one $k$ server problem: On any specified server request, it moves requested server to the location of new request. On any non-specified server request, it sends the last recently used server to the location of new request.

We construct an instance with $k + 1$ locations from set $\{1, 2, \ldots, k + 1\}$ and $k$ servers with server $s_i$ at location $i$. The adversarial sequence of requests consists of an initial sequence, followed by a repetition of $k - 1$ phases, phase $i$ consists of $2k - (i + 1)$ rounds. At the end of each phase, the adversary maintains two invariants:
(i) for any $i < j$, $s_i$ was used earlier than $s_j$; (ii) all servers are located in their initial locations except server $k - i$; if $i = 1$, server $k - 1$ is at location $k + 1$, while for $i \geq 2$, server $k - i$ is at location $k - i + 1$.

The first phase consists of $2k - 2$ requests. Every odd numbered request is $(k + 1, \ast)$; for $1 \leq j \leq k - 2$, the $j$th even numbered request is $(j, s_j)$, while the last request is $(k, s_k)$. Thus, the request sequence in the first phase is the following:

$$(k + 1, \ast), (1, s_1), (k + 1, \ast), (2, s_2), \ldots, (k + 1, \ast), (k - 2, s_{k-2}), (k + 1, \ast), (k, s_k).$$

It is easy to verify that LRU incurs a move for every request: server $s_j$ serves the $j$th odd numbered (unspecified) request and then moves again to serve its specific request. This results in a cost of $2k - 2$.

Phase $i > 1$ consists of $2k - 2i - 1$ requests. Each of the first $k - i$ odd numbered requests is $(k - i + 1, \ast)$; the $j$th even numbered request for $1 \leq j \leq k - i - 1$ is $(j, s_j)$. Of the remaining $i$ requests, the $j$th request is $(k - i + j, s_{k-i+j})$. Thus, the request sequence in phase $i > 1$ is given by:

$$(k - i + 1, \ast), (1, s_1), (k - i + 1, \ast), (2, s_2), \ldots, (k - (i + 1), s_{k-(i+1)}), (k - i + 1, \ast)$$

followed by

$$(k - i + 1, s_{k-i+1}), \ldots, (k, s_k).$$

From the initial server ordering and the invariants maintained, LRU moves one server for each of the first $2(k - (i + 1)) + 1$ requests: server $s_j$ serves the $j$th odd numbered request for $1 \leq j \leq k - i - 1$ and then moves back to $j$ serve its specified request. Thus, LRU incurs a cost of $2k - 2i - 1$. The server movements ensure invariant (ii), while the last $i$ requests of the requests ensure invariant (i) at the end of the phase.

Thus, after $k - 1$ phases, the cost of LRU is

$$\left(\sum_{i=1}^{k-1} 2(k - i)\right) - (k - 1) = k(k - 1) - (k - 1) = (k - 1)^2.$$
On the other hand, an optimal solution serves all unspecified requests in the first phase using server $s_k$, by moving it at unit cost once to location $k + 1$, and then moving it $s_k$ back to location $k$ for its specified request, for a total cost of 2. In all subsequent phases, there is no cost incurred. The adversary can repeat the sequence of requests forever, resulting in a competitive ratio of $(k - 1)^2/2$.

A $(2k - 1)$-lower bound

We show a lower bound of $2k - 1$ on the competitive ratio of any deterministic algorithm for the all-or-one $k$-server problem on uniform metrics. We construct an instance with $k + 1$ locations $\{1, 2, \ldots, k + 1\}$; without loss of generality, server $s_i$ is initially at location $i$ for $1 \leq i \leq k$.

Let $ALG$ be an arbitrary deterministic algorithm for the problem. The adversarial sequence of requests comprises a set of phases. Each phase consists of $2k$ requests, such that odd numbered requests are unspecified requests and even numbered requests are specific requests. The sequence of requests is as follows:

$$(k + 1, *), (x_2, s_{x_2}), (k + 1, *), (x_4, s_{x_4}), \cdots, (k + 1, *), (x_{2k-2}, s_{x_{2k-2}}), (k + 1, *),$$

where $x_{2i}$ is the index of server which $ALG$ has sent to serve $(2i - 1)$-th request.

In each phase, cost of $ALG$ is $2k - 1$ to serve the requests. Since the algorithm moves at most $k$ servers, an optimal solution pays 1 by sending only server $s_{x_{2k}}$ to location $k + 1$ for the first request, keeping it there for whole phase; the rest of the servers serve other specific requests without any move. After a phase, all servers are located on distinct locations, enabling the start of a new phase.

Thus, for any $T$, we can give a sequence of $2kT$ requests for which the algorithm incurs a cost of $(2k-1)T$, while an optimal solution incurs a cost of $T$. This completes the proof that competitive ratio of any deterministic algorithm is at least $2k - 1$ for all-or-one $k$ server problem.
2.5 Randomized algorithm for all-or-one uniform $k$-server

In this section we give a randomized $O(\log k)$-competitive algorithm for all-or-one uniform $k$-server. We start by giving an LP that captures the problem up to a constant factor. That is, we show that

- for any schedule of servers, we can produce an integral LP solution whose objective is at most twice the cost of the schedule, and
- for any integral solution to the LP, we can produce a schedule whose cost is a constant factor more than the LP objective.

The LP and the proof of the above two claims appear in Section 2.5.1. Next, we give a primal-dual $O(\log k)$-competitive algorithm in Section 2.5.2 to obtain a fractional solution to the LP online. Finally, in Section 2.5.3 we show how to round the fractional LP solution online.

2.5.1 An LP for all-or-one uniform $k$-server

We would like to encode Properties 1 and 2 from Lemma 2 as constraints in the LP, so that the LP produces a valid demand set at each time $t$. The LP, defined below, has a variable $x(r)$ for each request $r$, which can be interpreted in two ways. First, we can view $x(r)$ as the indicator that the server that serves request $r$ moves before the next request with the same demand as $r$ is served. We can also view $x(r)$ as indicating whether $dem(r) = (z, q)$ should be in our demand set at times $t$ for which $r$ is the most recent request with demand equal to $(z, q)$ (where $x(r) = 1$ indicates it should not be in the demand set).

Using this second interpretation, constraints 2.1 and 2.2 correspond to Properties 1 and 2 from Lemma 2. That is, constraint 2.1 says that the demand set produced by the LP (which must contain $dem(r_t)$ at time $t$) cannot contain more than $k$ representatives. (In fact, the LP constraint asserts that any independent subset, including
the set of representatives cannot contain more than \(k\) elements.) Constraint 2.2 says that only one demand with specific type \(s\) can be in the demand set at any time, for all \(s\).

We now formally define the LP. For demand \((z, q)\) let \(\text{last}_t(z, q)\) denote the most recent request with demand equal to \((z, q)\); \(\text{last}_t(z, q)\) is undefined if there has not yet been a request with demand \((z, q)\). Let \(B_t\) be the set of demands that have appeared at least once up to time \(t\). That is, \(B_t = \{(z, q) \in Z \times Q : \text{last}_t(z, q)\ \text{is defined}\}\). For each specific demand for server \(s\), let \(D_t\) contain the most recent request with that demand unless the request is the most recent request for \(s\) regardless of location.

More precisely, let \(\text{penult}_t(s)\) denote the request with the highest time index not exceeding \(t\) among the set \(\{r : \text{dem}(r) = (z, s), z \in Z\}\). That is, \(\text{penult}_t(s)\) denotes the most recent specific request for server \(s\) regardless of location. Then \(D_t = \{\text{last}_t(z, s) : (z, s) \in B_t, s \in S\} \backslash \{\text{penult}_t(s) : s \in S\}\).

\[
\begin{align*}
\min \sum_{r \in R} x(r) \\
\text{s.t.} \quad \sum_{(z, q) \in I \backslash \{\text{dem}(r_t)\}} x(\text{last}_t(z, q)) & \geq |I| - k & \forall I \subseteq B_t \text{ s.t. } I \text{ is independent}, \forall t \\
x(r) & \geq 1 & \forall r \in D_t, \forall t \\
0 & \leq x(r) \leq 1 & \forall r \in R
\end{align*}
\]

Lemma 6. \(LP \leq 2 \cdot OPT\), where \(OPT\) is cost of an optimal schedule for the given instance.

Proof. Suppose the optimal schedule moves a server \(s\) out of location \(z\) at time \(t\). In the LP, we set \(x(\text{last}_t(z, s)) = 1\) and \(x(\text{last}_t(z, \ast)) = 1\). Each time a server moves, the LP objective increases by at most 2, and therefore \(LP \leq 2 \cdot OPT\). It remains to show that this setting of variables produces a feasible LP solution.
Note that at any time \( t \), \( x(last_t(z,q)) = 0 \) implies that the server that served request \( last_t(z,q) \) must still be at location \( z \) (otherwise we would have set the variable to 1). Suppose there is some \( t \) for which a constraint (2.1) is violated. Then, there is some independent set \( I \subseteq B_t \), such that

\[
\sum_{(z,q) \in I \setminus \{ \text{dem}(r_t) \}} x(last_t(z,q)) < |I| - k, \tag{2.3}
\]

i.e.,

\[
\sum_{(z,q) \in I \setminus \{ \text{dem}(r_t) \}} (1 - x(last_t(z,q))) > k - 1. \tag{2.4}
\]

So \( I \setminus \{ \text{dem}(r_t) \} \) contains at least \( k \) demands \((z,q)\) such that the server that served \( last_t(z,q) \) has not moved. Since the set \( I \) is independent, none of these \( k \) servers can serve \( r_t \) without moving, and therefore we have a contradiction.

Next, suppose the violated constraint at time \( t \) is (2.2). Then, there is a pair of requests \( r = last_t(z,s) \) and \( r' = last_t(z',s) \), with \( z \neq z' \) such that \( r \) came before \( r' \) in the request sequence, and both \( x(last_t(z,s)) \) and \( x(last_t(z',s)) \) are 0. Only server \( s \) can serve these requests, and therefore \( s \) must be at both location \( z \) and location \( z' \) at time \( t \), giving a contradiction. \( \Box \)

Our next goal is to show that an integral LP solution can be converted into a schedule. To do this, we use the framework of Section 2.2. That is, we give a sequence of demand sets \( Y_1, \ldots, Y_T \) such that \( \text{dem}(r_t) \in Y_t \) for all \( t \). We use Lemma 3 to produce a sequence of server configurations from the demand sets.

First, we bound the total number of demands against OPT.

**Lemma 7.** If \( \mathcal{D} \) is the set of distinct demands in an instance, then \( |\mathcal{D}| \leq 2 \cdot OPT \).

**Proof.** Let \( L \) denote the set of times in \([T]\) when a demand is requested for the first time. Thus, \( |L| = |\mathcal{D}| \). For \( t \in L \), consider \( r_t \) and let \( r_t = (z,q) \). This is the first time demand \((z,q)\) has appeared in a request and we will charge \((z,q)\) to the optimal
solution. If \( q = s \in \mathcal{S} \), OPT must move server \( s \) to location \( z \) to serve request \( r_t \), and we charge \((z, q)\) to this move. If \( q = * \) and this is the first time location \( z \) has been requested, again OPT must move some server to \( z \) to serve \( r_{t+1} \), and we charge \((z, q)\) to this move. Finally, if \( q = * \) and this is not the first time location \( z \) has been requested, let \( r \) be the first request at location \( z \). OPT must have moved some server to \( z \) to serve the request, and we charge to this move. This last case can happen at most once at each location, and therefore each move by the optimal solution is charged at most twice.

\[ \square \]

**Theorem 8.** For any integral solution to the LP, there is a sequence of server configurations whose cost is at most a constant factor larger than the optimal sequence of server configurations.

**Proof.** We suppose that in the initial configuration, all servers start at a location that will never be requested. We let \( Y_0 = \{\} \) be a dummy demand set for this initial configuration of servers. Let the demand set at time \( t \) be \( Y_t = \{(z, q) : x(last_t(z, q)) = 0\} \cup \{dem(r_t)\} \). The initial server configuration covers demand set \( Y_0 \), and we can apply Lemma 3 to produce the \( t \)th server configuration from the \((t-1)\)th configuration. Since \( dem(r_t) \in Y_t \) and the \( t \)th server configuration covers \( Y_t \), the configuration can also serve request \( r_t \).

There are two things we need to do to complete the proof. First, we show that \( Y_t \) is, in fact, a valid demand set the way it is defined above. Second, we bound the cost of the sequence of demand sets. By Lemma 3, this gives us the cost of the sequence of server configurations.

We claim that \( Y_t \) is a valid demand set. By constraint 2.1 and the fact that the
set of representatives is independent, we have

\[ \sum_{(z,q) \in \text{rep}(Y_t) \setminus \text{dem}(r_t)} x(\text{last}_t(z,q)) \geq |\text{rep}(Y_t)| - k \]  

(2.5)

i.e.,

\[ \sum_{(z,q) \in \text{rep}(Y_t) \setminus \text{dem}(r_t)} (1 - x(\text{last}_t(z,q))) \leq k - 1. \]  

(2.6)

Since \( x(\text{last}_t(z,q)) = 0 \) for all \((z,q)\) in \( Y_t \setminus \{\text{dem}(r_t)\} \), this implies \( |\text{rep}(Y_t) \setminus \{\text{dem}(r_t)\}| \leq k - 1 \) and therefore \( |\text{rep}(Y_t)| \leq k \). This shows that \( Y_t \) satisfies Property 1 of Lemma 2.

We next claim that \( Y_t \) satisfies Property 2 of Lemma 2. By constraint 2.2, for each server \( s \), we can have \( x(r) = 0 \) only if \( r \) is the most recent request for server \( s \). Therefore, \( Y_t \) contains at most one demand \((z,s)\), where \( z \) is most recent location where server \( s \) was requested.

We next bound the cost of the schedule produced by the sequence \( Y_0, Y_1, \ldots, Y_T \). By Lemma 3, the cost incurred by our schedule at time \( t + 1 \) is at most \( 2 \cdot |Y_{t+1} \setminus Y_T| \).

Note that \( \text{last}_t(z', q') = \text{last}_{t+1}(z', q') \) for all \( z', q' \) except for \( z' = z \) and \( q' = q \) where \((z,q) = \text{dem}(r_{t+1})\). Therefore, \((z,q)\) can be the only demand in \( Y_{t+1} \setminus Y_t \). We show how to charge 1 unit to either the LP objective or the optimal schedule. If demand \((z,q)\) has appeared in the request sequence up to and including time \( t \), then \( x(\text{last}_t(z,q)) = 1 \), (otherwise \((z,q) \in Y_t \) by definition of \( Y_t \)). In this case we charge \((z,q)\) to \( x(\text{last}_t(z,q)) \). No \( x(r) \) gets charged more than once, since we only charge \( x(r) \) when \( \text{dem}(r) = \text{dem}(r_{t+1}) = (z,q) \), and therefore when \( \text{last}_t(z,q) = \text{last}_{t+1}(z,q) \).

If this is the first time demand \((z,q)\) appears in a request, we bound \((z,q)\) against OPT by Lemma 7.

\[ \Box \]

2.5.2 The primal dual algorithm for obtaining a fractional solution

**Lemma 9.** There is an \( O(\log k) \)-competitive algorithm for solving the LP online fractionally.
We first give a very brief overview of $O(\log k)$-competitive algorithm for solving the LP.

We start by writing the dual of our LP. In the online problem, the LP is revealed over time. At time $t$, we get a new primal variable corresponding to request $r_t$, as well as possibly several new primal constraints, each corresponding to an independent set. We pick an arbitrary violated constraint, and raise the corresponding dual variable along with all primal variables in the constraint until the constraint is satisfied. We repeat with a new constraint, until all the constraints introduced at time $t$ are satisfied. We then proceed to the next time step.

By definition, the primal solution produced by the algorithm is feasible. To bound its cost, we first show that the primal objective is at most a constant factor more than the cost of the dual objective. Next, we show that each constraint in the dual is feasible up to a factor of $O(\log k)$. Together, this implies the algorithm is $O(\log k)$-competitive.

We next expand on the details of the algorithm. We first write the dual of the LP.

$$\max \sum_t \sum_{I \in B_t, I \text{ independent}} (|I| - k)\alpha(I, t) + \sum_t \sum_{r \in D_t} \beta(r, t) - \sum_{r \in R} \gamma(r)$$

s.t. $\sum_t \sum_{I \subseteq B_t \setminus r_t \text{ s.t. } \exists(z, q) \in I, r=last_t(z, q)} \alpha(I, t) + \sum_{t \in D_t} \beta(r, t) - \gamma(r) \leq 1 \quad \forall r \in R$

The algorithm is as follows. At each time $t$, we get a new primal variable and a set of new primal constraints. Pick some violated primal constraint.

Suppose the violated constraint is (2.2) and let the index of the constraint be $(r, t)$. We increase $x(r)$ and the corresponding dual variable $\beta(r, t)$ at the same rate until $x(r) = 1$ and the primal constraint is satisfied.
Next, suppose the violated constraint is (2.1) and let the index of the constraint be \((I, t)\). We increase the corresponding dual variable \(\alpha(I, t)\). Additionally, for each \((z, q) \in I\) such that with \(x(last_t(z, q)) < 1\) and letting \(r = last_t(z, q)\), increase \(x(r)\) at rate
\[
\frac{dx(r)}{d\alpha(I, t)} = x(r) + \frac{1}{k}.
\]
For each \((z, q) \in I\), such that with \(x(last_t(z, q)) = 1\) and letting \(r = last_t(z, q)\), increase \(\gamma(r, t)\) at the same rate as \(\alpha(I, t)\). We stop increasing the variables once the primal constraint is satisfied. Repeat the above steps until all the constraints introduced at time \(t\) are satisfied, then proceed to \(t + 1\).

By definition, the primal solution produced by the algorithm is feasible. To bound its cost, we first bound the cost of the primal against the dual objective. Next, we show that each constraint in the dual is feasible up to a factor of \(O(\log k)\). Together, this implies the above algorithm is \(O(\log k)\)-competitive.

**Lemma 10.** While increasing variables in the above algorithm, the change in the primal objective is at most twice the change in the dual objective.

*Proof.* If the violated constraint is (2.2), the primal and dual objectives increase by the same amount. We give a proof when the violated constraint is (2.1). Let \(L\) denote the set of \((z, q) \in I \setminus \{dem(r_t)\}\) such that \(x(last_t(z, q)) < 1\). The change in the dual objective is
\[
(|I| - k) \cdot \Delta \alpha(I, t) - (|I| - |L| - 1) \cdot \Delta \alpha(I, t) = (|L| - k - 1) \cdot \Delta \alpha(I, t).
\]
The change in the primal objective is
\[
\sum_{(z,q) \in L} \Delta x(\text{last}_t(z,q)) = \sum_{(z,q) \in L} \left(x(\text{last}_t(z,q)) + \frac{1}{k}\right) \cdot \Delta \alpha(I,t)
\]

\[= \left[ \sum_{(z,q) \in I \setminus \{\text{dem}(r_1)\}} x(\text{last}_t(z,q)) - \sum_{(z,q) \in I \setminus \{\text{dem}(r_1)\}} x(\text{last}_t(z,q)) + \frac{1}{k} |L| \right] \cdot \alpha(I,t)\]

\[\leq \left( |I| - k - |I \setminus \{\text{dem}(z_t)\}| + \frac{1}{k} |L| \right) \cdot \Delta \alpha(I,t)
\]

(by primal constraint 2.1, and since \(x(\text{last}_t(z,q)) = 1\) for \((z,q) \notin L\))

\[= \left( \frac{1}{k} |L| + |L| - k + 1 \right) \cdot \Delta \alpha(I,t).\]

Note that \(|L| \geq k\), since otherwise the primal constraint \((I,t)\) would not be violated. Therefore, we have

\[\left( |L| \frac{1}{k} + |L| - k + 1 \right) \leq 2 \cdot (|L| - k + 1),\]

which completes the proof.

\[\square\]

**Lemma 11.** Every dual constraint is satisfied up to a factor of \(O(\log k)\).

**Proof.** Consider some dual constraint \(r\). By the definition of our algorithm, at most one variable \(\beta(r,t) > 0\) and additionally \(\beta(r,t) \leq 1\), so

\[\sum_{t \in \mathcal{D}_t} \beta(r,t) \leq 1.\]

Recall that the update rule for \(\alpha\) is

\[\frac{dx(r)}{d\alpha(S,t)} = x(r) + \frac{1}{k}\]

if \(x(r) < 1\) and

\[\frac{d \gamma(r)}{d \alpha(S,t)} = 1\]

30
otherwise. Additionally, consider some fixed some request \( r \). For any \( I \) and \( t \) such that \((z, q) \in I \) and \( r = \text{last}_t(z, q)\), one of either \( x(r) \) or \( \gamma(r) \) is always increasing while \( \alpha(S, t) \) is increasing. Therefore we have

\[
\sum_t \sum_{I \subseteq B_t \setminus r_t \text{ s.t. } \exists (z, q) \in I, r = \text{last}_t(z, q)} \alpha(I, t) - \gamma(r) \leq \int_0^1 \left(x(r) + \frac{1}{k}\right) \, dr \\
= \left[ \ln \left(x(r) + \frac{1}{k}\right) \right]_0^1 \\
= \ln(k + 1).
\]

Putting these together,

\[
\sum_t \sum_{I \subseteq B_t \setminus r_t \text{ s.t. } \exists (z, q) \in I, r = \text{last}_t(z, q)} \alpha(I, t) + \sum_{t \in D_t} \beta(r, t) - \gamma(r) \leq O(\log k) \quad \forall r \in R.
\]

### 2.5.3 Online rounding for obtaining an integral solution

In this section, we show how to convert an online fractional solution to the LP into a randomized algorithm for all-or-one \( k \) server. We denote by \( LP_t \) the fractional solution to the LP at time \( t \). We let \( x_t(r) \) be the value of \( x(r) \) in \( LP_t \).

At each time \( t \), our algorithm will produce a (random) demand set \( Y_t \). As we did in Theorem 8 (deterministic algorithm), we use Lemma 3 to produce a sequence of server configurations from these demand sets which has a total cost of at most \( \sum_{t=2}^T |Y_t \setminus Y_{t-1}| \). Therefore, for the rest of the section we focus on generating these demand sets.

Like in the proof of Theorem 8, we want the value of \( 1 - x_t(\text{last}_t(z, q)) \) to determine whether \((z, q)\) should appear in demand set \( Y_t \). \( x_t(.) \) is possibly no longer integral, so \( 1 - x_t(\text{last}_t(z, q)) \) will give us the probability that \((z, q)\) appears in \( Y_t \) (except for the current request \( r_t \), which must appear with probability 1).
We will maintain a probability distribution over demand sets at each time $t$, and denote by $y_t(z, q)$ the total weight of the demand sets containing $(z, q)$. We say that the distribution is *consistent* with the LP solution at time $t$ if

$$y_t(z, q) = \begin{cases} 
0 & \text{for all } (z, q) \\
1 - x_t(last_t(z, q)) & \text{s.t. } (z, q) \text{ has not yet appeared in any request}, \\
1 & \text{for } (z, q) = \text{dem}(r_t). 
\end{cases}$$

To simplify the algorithm and proof, we discretize the distribution so each demand set in the distribution has the same weight, $\epsilon$, for some small enough $\epsilon > 0$ (note that a demand set can appear multiple times). Suppose $Y_{i,t}$ denotes the $i$th demand set in the distribution at time $t$.

Our randomized algorithm is as follows: We suppose that in the initial configuration, all servers start at a location that will never be requested. We let $Y_0 = \emptyset$ be a dummy demand set for this initial configuration of servers. We pick some index $\hat{i}$ uniformly at random, and for each time $t$, we set $Y_t = Y_{\hat{i}, t}$. The initial server configuration covers demand set $Y_0$, and we can apply Lemma 3 to produce the $t$th server configuration from the $(t - 1)$th configuration. Since $\text{dem}(r_t) \in Y_{i,t}$ for all $i$ and $t$ and the $t$th server configuration covers $Y_t$, the configuration can also serve request $r_t$.

The expected cost of this randomized algorithm at time $T$ is

$$\mathbb{E} \left[ \sum_{t=1}^{T} |Y_t \setminus Y_{t-1}| \right] = \sum_{t=1}^{T} \mathbb{E} [|Y_t \setminus Y_{t-1}|] = \sum_{t=1}^{T} \left( \epsilon \cdot \sum_{i=1}^{\frac{1}{\epsilon}} |Y_{i,t-1} \setminus Y_{i,t}| \right). \quad (2.7)$$

In the rest of this section, we give an algorithm that, given a consistent distribution $Y_{T-1}$ over demand sets at time $T - 1$ and a new fractional solution $LP_T$, finds a new distribution over valid demand sets $Y_T$ for time $T$ such that

- the new distribution $Y_T$ is consistent with $LP_T$, and
the cost in equation (2.7) till time $T$ can be charged to the fractional LP solution.

Note that $Y_0$ is a consistent distribution, so the above properties are satisfied for the base case, i.e., at $T = 0$.

To produce distribution $Y_T = \{Y_{i,T}\}$ from $Y_{T-1} = \{Y_{i,T-1}\}$, we start by setting $Y_{i,T} = Y_{i,T-1}$ for all $i$, and then make changes to restore consistency. For each $(z, q)$, we compare the values $y_{T-1}(z, q)$ and $y_T(z, q)$. If $y_{T-1}(z, q) = y_T(z, q)$, then $(z, q)$ already appears in the correct number of demand sets of the distribution at time $T$.

Next suppose $y_{T-1}(z, q) > y_T(z, q)$. To make the distribution consistent at time $T$, we need to remove $(z, q)$ from $(y_{T-1}(z, q) - y_T(z, q))/\epsilon$ demand sets. We do this by removing $(z, q)$ from any $(y_{T-1}(z, q) - y_T(z, q))/\epsilon$ demand sets that contain $(z, q)$. For any demand set $i$ where $(z, q)$ was removed, $|Y_{i,T} \setminus Y_{i,T-1}| = 0$, and therefore this change doesn’t incur any cost. Additionally, each new demand set is still valid because we have only removed demands.

Finally, we consider the case where $y_{T-1}(z, q) < y_T(z, q)$. Unlike the previous cases, this will require adding $(z, q)$ to various demand sets, which will incur a cost. Since the LP solution is monotone, and by the definition of $y_T(z, q)$, it must be the case that $(z, q) = dem(r_T)$ and therefore $y_T(z, q) = 1$. We need to add $(z, q)$ to the $1 - y_{T-1}(z, q)$ fraction of the demand sets not yet containing $(z, q)$. Then, the total cost would be

$$\epsilon \cdot \sum_{i=1}^{1/\epsilon} |Y_{i,T} \setminus Y_{i,T-1}| = \epsilon \cdot \sum_{i:(z,q)\notin Y_{i,T-1}} |Y_{i,T} \setminus Y_{i,T-1}| = 1 - y_{T-1}(z, q).$$

(2.8)

We bound this cost, summed over all times till $T$, against the LP and the optimal schedule in Lemma 12 below. Unfortunately, just adding $(z, q)$ to these demand sets and bounding the cost is not enough. Besides incurring a cost, adding $(z, q)$ to demand sets may invalidate the demand sets. Therefore, we will make additional
changes so that each $Y_{t,T}$ is a valid demand set, without changing any value of $y_T(z,q)$. These changes will incur an additional cost, which we will show is at most $O(1 - y_{T-1}(z,q))$. We start by bounding the total cost we have incurred up to time $T$.

We abuse notation slightly and let $y_t(dem(r_t))$ denote $y_t(z,q)$ where $dem(r_t) = (z,q)$.

**Lemma 12.**

$$\sum_{t=1}^{T} (1 - y_{t-1}(dem(r_t))) \leq O(OPT + LP_T)$$

where $OPT$ is the value of the optimal schedule and $LP_T$ is a fractional solution to the $T$th LP.

**Proof.** Let $L$ denote the set of times in $[T]$ when a demand that has not appeared earlier is requested. Let $\overline{L} = [T] \setminus L$.

$$\sum_{t=1}^{T} (1 - y_{t-1}(dem(r_t))) = \sum_{t \in L} (1 - y_{t-1}(dem(r_t))) + \sum_{t \in \overline{L}} (1 - y_{t-1}(dem(r_t)))$$

$$= \sum_{t \in L} 1 + \sum_{t \in \overline{L}} (1 - (1 - x_{t-1}(r_t)))$$

(by definition of $y_t(\cdot)$ and $L$)

$$= |L| + \sum_{t \in \overline{L}} x_{t-1}(r_t) \leq 2 \cdot OPT + LP_T.$$  

(by Lemma 7 and the LP)

What remains is to show that we can restore the validity of the demand sets while paying a cost of $O(\Delta)$ after adding $dem(r_T) = (z,q)$ to the $\Delta = 1 - y_{T-1}(z,q)$ fraction of the demand sets it is missing from. Incrementally, it is enough to show that we can add $r_T$ to one additional demand set, then restore validity of the demand
sets while paying a cost of $O(\epsilon)$ (i.e., adding a constant number of demands to the demand sets). In restoring validity, we only transfer demands, i.e., remove a demand from one demand set and add it to another. Therefore, the fraction of demand sets a demand appears in will not change. For the rest of the section, we drop the subscript $T$ to simplify notation.

Let $r$ be the current request. First, we pick an arbitrary demand set $Y_i$ that does not contain $dem(r) = (z, q)$ and add it. That is, set $Y_i \leftarrow Y_i \cup \{(z, q)\}$. Recall that Lemma 2 gives the two properties for validity of a demand set. We first claim that demand set $Y_i$ still satisfies Property 2 of Lemma 2. Property 2 could only be violated if $q = s$ for some $s \in S$, and $Y_i$ contained $(z', s)$ for some $z' \neq z$. However, constraint (2.2) ensures $x(last_T(z', s)) = 1$ for all $z' \neq z$. Therefore, $(z', s)$ cannot be present in any demand set.

It is possible that $Y_i$ no longer satisfies Property 1 of Lemma 2. That is, $|rep(Y_i)| = k + 1$. To restore Property 1, we transfer a constant number of demands between configurations to make $Y_i$ valid and keep all the other demand sets valid.

The remainder of the algorithm and analysis relies on the following pair of lemmas, whose proofs appear at the end of the section.

**Lemma 13.** Suppose $|rep(Y_i)| = k + 1$, and suppose there is some $j$ such that $|rep(Y_j)| = k - 1$. Then we can modify $Y_i$ and $Y_j$ to $Y'_i$, $Y'_j$ by transferring demands, such that $|rep(Y'_i)| = k$ and $|rep(Y'_j)| \leq k$ and the total cost is $O(\epsilon)$, i.e. $|Y'_i \setminus Y_i| + |Y'_j \setminus Y_j| = O(1)$.

If there exists such a $j$, then performing the modifications in the lemma gives a distribution in which all demand sets are valid, as desired. However it is possible that no such $j$ exists, i.e., $|rep(Y_j)| = k$ for all $j \neq i$. In that case, we can apply the following lemma.
Lemma 14. Suppose $|\text{rep}(Y_i)| = k + 1$ and $|\text{rep}(Y_p)| = k$ for all $p \neq i$. Then, there is some pair of demand sets $Y_j, Y_\ell$ that can be modified to $Y_j', Y_\ell'$ by transferring demands such that,

- $|\text{rep}(Y_j')| < |\text{rep}(Y_j)|$ and $|\text{rep}(Y_\ell')| = |\text{rep}(Y_\ell)|$, and
- The total cost is $O(\epsilon)$, i.e. $|Y_j\setminus Y_j'| + |Y_\ell\setminus Y_\ell'| = O(1)$.

So, if $|\text{rep}(Y_p)| = k$ for all $p \neq i$, we can reduce $|\text{rep}(Y_j)|$ to $|\text{rep}(Y_j')| - 1$, without invalidating any other demand set. Either $j = i$ and we are done, or $j \neq i$ and so $|\text{rep}(Y_j)| = k - 1$. From here, we can apply Lemma 13 to reduce $|\text{rep}(Y_j)|$ from $k + 1$ to $k$, giving us a distribution with valid demand sets. We conclude the section with proofs of the lemmas.

Proof of Lemma 13. We let $\text{rep}_z(Y)$ denote the representatives of $Y$ restricted to location $z$. Suppose there is some $j$ such that $|\text{rep}(Y_j)| = k - 1$. By the Pigeonhole principle, there is a location $z$ such that $|\text{rep}_z(Y_i)| > |\text{rep}_z(Y_j)|$. Given such a location, we show how to transfer a constant number of demands to produce $Y_i', Y_j'$ such that $|\text{rep}(Y_i')| = k$ and $|\text{rep}(Y_j')| \leq k$. In Figure 2.1 we go through the various cases.

Since $|\text{rep}_z(Y_i)| > |\text{rep}_z(Y_j)|$, there must be some representative in $\text{rep}_z(Y_i)$ that is not in $\text{rep}_z(Y_j)$. We would like to move this representative from $Y_i$ to $Y_j$, decreasing the number of representatives in $Y_i$ by 1. The only scenario in which this does not work is when there are two demands in $Y_i$ at location $z$, $(z, *)$ and $(z, s)$ for some $s$. Here, $(z, s)$ is the only demand in $\text{rep}_z(Y_i)$, but moving it causes $(z, *)$ to become a representative, so $|\text{rep}(Y_i)|$ does not decrease. Since $\text{rep}_z(Y_j)$ is empty in this case, we move both $(z, *)$ and $(z, s)$ to $Y_j$. \hfill \square

Proof of Lemma 14. Suppose $|\text{rep}(Y_j)| = k$ for all $j$, and there is no pair $Y_j, Y_\ell$ satisfying the conditions in the lemma. We show a contradiction.

First, we show that the demand sets must satisfy the following two properties:
Figure 2.1: We show how to reduce \(|\text{rep}(Y_i)|\) by 1 while increasing \(|\text{rep}(Y_j)|\) by at most 1, given some location \(z\) for which \(|\text{rep}(Y_i)| > |\text{rep}(Y_j)|\). Additionally \(|Y_j \setminus Y_i| \leq 2\) and \(|Y_i \setminus Y_j| = 0\). Representatives are highlighted with grey boxes.

Property 1: Suppose for location \(z\) that \(y(z, \cdot) \leq \sum_{s \in S} y(z, s)\). Then for every \(Y_i\), 
\((z, \cdot) \in Y_i\) implies \((z, s) \in Y_i\) for some \(s \in S\).

Property 2: Suppose for location \(z\) that \(y(z, \cdot) > \sum_{s \in S} y(z, s)\). Then \((z, s) \in Y_i\) implies \((z, s') \notin Y_i\) for all \(s' \neq s\) and \((z, \cdot) \in Y_i\). That is if \(y(z, \cdot) > \sum_{s \in S} y(z, s)\), any demand set contains at most one demand of a specific type at location \(z\), and any demand set containing a demand of a specific type at location \(z\) must also contain \((z, \cdot)\).

Next, we give proofs of the two properties. Let \(\text{rep}_z(Y)\) denote representatives of \(Y\), restricted to location \(z\). If Property 1 is not true, then there exists some \(Y_i\) with \(\text{rep}_z(Y_i) = \{(z, \cdot)\}\), and some \(Y_j\) that satisfies either of the following conditions: \((z, \cdot) \notin Y_j\) or \(|\text{rep}_z(Y_j)| \geq 2\). In either case, we can move a demand from \(\text{rep}_z(Y_j)\) to \(Y_i\), decreasing the number of representatives of \(Y_j\) by 1, without changing the number of representatives of \(Y_i\). This contradicts our assumption that such a move does not exist.
Similarly, if Property 2 is false, then there exists some $Y_\ell$ with $rep_z(Y_\ell) = \{(z, *)\}$, and some $Y_j$ that satisfies either of the following conditions: $rep_z(Y_j) = \{(z, s)\}$ for some $s \in S$, or $|rep_z(Y_j)| \geq 2$. In the first case, we move $(z, s)$ from $Y_j$ to $Y_\ell$, and in the second case we move one of the demands of $rep_z(Y_j)$ to $Y_\ell$. We reach the same contradiction as in the proof of Property 1.

Next, we use Properties 1 and 2 to reach a contradiction. Note that $y(z, q) \leq 1 - x(last_T(z, q))$ (the quantities are equal in all cases, except for the current request $r$). Rewriting constraint (2.1) of the LP, for any independent set $I$, we have

$$
\sum_{(z, q)\in I} (1 - x(last_T(z, q))) \leq k - 1
$$

i.e.,

$$
\sum_{(z, q)\in I} y(z, q) \leq k. \quad (2.9)
$$

Let $1_{Y_\ell}(z, q)$ be the indicator function:

$$
1_{Y_\ell}(z, q) = \begin{cases} 
1 & \text{if } (z, q) \in Y_\ell, \text{ and } \\
0 & \text{otherwise.}
\end{cases}
$$

Then, we have

$$
\epsilon \cdot \sum_{\ell=1}^{1/\epsilon} \sum_{(z, q)\in I} 1_{Y_\ell}(z, q) = \sum_{(z, q)\in I} y(z, q) \leq k \quad \text{(by Eqn. (2.9))}. \quad (2.10)
$$

We construct a specific independent set $I$, and use Eqn. (2.10) to show a contradiction: At each location $z$, take $(z, *)$ in $I$ if $y(z, *) > \sum_{s\in S} y(z, s)$, and take $(z, s)$ in $I$ for all $s \in S$ otherwise. $I$ is independent since we take either the unspecified demand or all specific demands at each location.

For any $Y_\ell$, Properties 1 and 2 ensure that for any representative $(z, q) \in rep(Y_\ell)$, either $(z, q) \in I$, or $|rep_z(Y_\ell)| = 1$ and $(z, *) \in I$. Therefore,

$$
\sum_{(z, q)\in I} 1_{Y_\ell}(z, q) = |rep(Y_\ell)|. \quad (2.11)
$$
Therefore, 

\[ \epsilon \cdot \sum_{\ell=1}^{1/\epsilon} \sum_{(z,q) \in I} 1_{Y_{\ell}}(z,q) = \epsilon \cdot \sum_{\ell=1}^{1/\epsilon} |\text{rep}(Y_{\ell})| = \epsilon \cdot \left[ \left( \frac{1}{\epsilon} - 1 \right) \cdot k + (k + 1) \right] > k, \]

which contradicts Eqn. (2.10). \qed
3

Retraction to Cycles

3.1 Introduction

Originally introduced in 1930 by K. Borsuk in his PhD thesis [28], retraction is a fundamental concept in topology describing continuous mappings of a topological space into a subspace that leaves the position of all points in the subspace fixed. Over the years, this has developed into a rich theory with deep connections to fundamental results in topology such as Brouwer’s Fixed Point Theorem [76]. Inspired by this success, graph theorists have extensively studied a discrete version of the problem in graphs, where a retraction is a mapping from the vertices of a graph to a given subgraph that produces the identity map when restricted to the subgraph (i.e., it leaves the subgraph fixed). For a rich history of retraction in graph theory, we refer the reader to [75]. Define the stretch of a retraction to be the maximum distance between the images of the endpoints of any edge, as measured in the subgraph. We use stretch- \( k \) retraction to mean a retraction whose stretch is \( k \); in particular, a stretch-1 retraction is a mapping where every edge of the graph is mapped to either an edge of the subgraph, or both its ends are mapped to the same vertex of the
In this chapter, we study the algorithmic problem of finding a *minimum stretch retraction* in a graph. This problem belongs to the rich area of metric embeddings, but somewhat surprisingly, has not received much attention in spite of the deep but non-constructive results in the graph theory literature. The graph retraction problem has a close resemblance to the well-studied 0-extension problem [32, 81, 82] (and its generalizations such as metric labeling [86, 43]), which is also an embedding of a graph $G$ to a metric over a subset of terminals $H$ with the constraint that each vertex in $H$ maps to itself. The two problems differ in their objective: whereas 0-extension seeks to minimize the *average* stretch of edges, graph retraction minimizes the *maximum* stretch. The different objectives lead to significant technical differences. For instance, a well-studied linear program called the earthmover LP has a nearly logarithmic integrality gap for 0-extension. In contrast, we show that a corresponding earthmover LP for graph retraction has integrality gap $\Omega(\sqrt{n})$. A well-studied problem in the metric embedding literature that considers the maximum stretch objective is the *minimum bandwidth* problem, where one seeks an isomorphic embedding of a graph into a line (or cycle) that minimizes maximum stretch. In contrast, in graph retraction, we allow homomorphic maps\(^2\) but additionally require a subset of vertices (the anchors) to be mapped to themselves.

From an applications standpoint, our original motivation for studying minimum-stretch graph retraction comes from a distributed systems scenario where the aim is to map processes comprising a distributed computation to a network of servers where

---

\(^1\) In the literature, a stretch-1 retraction is often simply referred to as a retraction or a retract. Also, in many studies, a (stretch-1) retraction requires that the two end-points of an edge in the graph are mapped to two end-points of an edge in the subgraph. These studies differentiate between the case where the subgraph being retracted to is reflexive (has self-loops) or irreflexive (no self-loops). In this sense, our notion of graph retraction corresponds to their notion of retraction to a reflexive subgraph.

\(^2\) A *homomorphic* map is one where an image can have multiple pre-images, while an *isomorphic* map requires that every image has at most one pre-image.
some processes are constrained to be mapped onto specific servers. The objective is to minimize the maximum communication latency between two communicating processes in the embedding. Such anchored embedding problems can be shown to be equivalent to graph retraction for general subgraphs, and arise in several other domains including VLSI layout, multi-processor placement, graph drawing, and visualization [74, 73, 104].

3.1.1 Problem definition, techniques, and results

We begin with a formal definition of the minimum stretch retraction problem.

**Definition 1.** Given an unweighted guest graph $G = (V, E)$ and a host subgraph $H = (A, E')$ of $G$, a mapping $f : V \rightarrow A$ is a retraction of $G$ to $H$ if $f(v) = v$ for all $v \in A$. For a given retraction $f$ of $G$ to $H$, define the stretch of an edge $e = (u, v) \in E(G)$ to be $d_H(f(u), f(v))$, where $d_H$ is the distance metric induced by $H$, and define the stretch of $f$ to be the maximum stretch over all edges of graph $G$. The goal of the minimum-stretch graph retraction problem is to find a retraction of $G$ to $H$ with minimum stretch. We refer to the vertices of $A$ as anchors.

The graph retraction problem is easy if the subgraph $H$ is acyclic (see, e.g., [96]); therefore, the first non-trivial problem is to retract a graph into a cycle. Indeed, this problem is NP-hard even when $H$ is just a 4-cycle [54]. Given this intractability result, a natural goal is to obtain an algorithm for retracting graphs to cycles that approximately minimizes the stretch of the retraction. This problem is the focus of our work. While there has been considerable interest in identifying conditions under which retracting to a cycle with stretch 1 is tractable [69, 75, 118], there has been no work (to the best of our knowledge) on deriving approximations to the minimum stretch.

We consider the following lower bound for the problem: if anchors $u$ and $v$ are distance $\ell$ in $H$, and there exists a path of $k$ vertices in $G$ between $u$ and $v$, then every
retraction has stretch at least $\ell/k$. This lower bound turns out to be tight when $H$ is acyclic, which is the reason retraction to acyclic graphs is an easy problem. However, this lower bound is no longer tight when $H$ is a cycle. For example, consider a grid graph where $H$ is the border of the grid. The lower bound given above says that any retraction has stretch at least $O(1)$. However, using the well-known Sperner’s lemma, we show that the optimal retraction is $\Omega(\sqrt{n})$.

Using just the simple distance based lower bound, we show that the gap on the grid is in fact the worst possible by giving a $O(\min\{k, \sqrt{n}\})$-approximation for the problem, where $k$ is the number of vertices of $H$. Our algorithm works by first mapping vertices of the graph into a grid, then projecting vertices outward to the border from the largest hole in the grid, which is the largest region containing no vertices.

**Theorem 15.** There is a deterministic, polynomial-time algorithm that computes a retraction of a graph to a cycle with stretch at most the minimum of $k/2$ and $O(\sqrt{n})$ times the optimal stretch.

We give evidence that the gap induced by Sperner’s lemma on a grid graph is fundamental, showing an $\Omega(\min\{k, \sqrt{n}\})$ integrality gap for natural linear and semi-definite programming relaxations of the problem. To overcome this gap, we focus on the special case of planar graphs, of which the grid is an example. Retraction in planar graphs has been considered in the past, most notably in a beautiful paper of Quillot [98] who uses homotopy theory to characterize stretch-1 retractions of a planar graph to a cycle. Quillot’s proof, however, does not yield an efficient algorithm. We provide an exact algorithm for retraction in planar graphs by developing the gap induced by Sperner’s lemma on a grid into a general lower bound on the optimal stretch for planar graphs.
Theorem 16. There is a deterministic, polynomial-time algorithm that computes a retraction of a planar graph to a cycle with optimal stretch.

Unfortunately, our techniques rely heavily on the planarity of the graph, and do not appear to generalize to arbitrary graphs. We leave the question of obtaining a better approximation for general graphs open.

3.1.2 Related work

List homomorphisms and constraint satisfaction

The graph retraction problem is a special case of the list homomorphism problem introduced by Feder and Hell [54], who established conditions under which the problem is NP-complete. Given graphs $G$, $H$, and $L(v) \subseteq V(H)$ for each $v \in V(G)$, a list homomorphism of $G$ to $H$ with respect to $L$ is a homomorphism $f : G \to H$ with $f(v) \in L(v)$ for each $v \in V(G)$.

Several special cases of graph retraction and variants of list homomorphism have been subsequently studied (e.g., [53, 75, 117, 118]). These studies have established and exploited the rich connections between list homomorphism and Constraint Satisfaction Problems (CSPs). Though approximation algorithms for CSPs and related problems such as Label Cover have been extensively studied, the objective pursued there is that of maximizing the number of constraints that are satisfied. For our graph retraction problem, this would correspond to maximizing the number of edges that have stretch below a certain threshold. Our notion of approximation in graph retraction, however, is the least factor by which the stretch constraints need to be relaxed so that all edges are satisfied.

0-extension, minimum bandwidth, and low-distortion embeddings

From an approximation algorithms standpoint, the graph retraction problem is closely related to the 0-extension and minimum bandwidth problems [55, 24, 67, 115, 45,
105]. In the 0-extension problem, one seeks to minimize the average stretch, which can be solved to an \(O(\log k/\log \log k)\) approximation using a natural LP relaxation [32, 49]. In contrast, we give polynomial integrality gaps for the graph retraction problem. In the minimum bandwidth problem, the objective is to find an embedding to a line that minimizes maximum stretch, but the constraint is that the map must be isomorphic rather than that the anchor vertices must be fixed. In a seminal result [55], Feige designed the first polylogarithmic-approximation using a novel concept of volume-respecting embeddings. A slightly improved approximation was achieved in [46] by combining Feige's approach with another bandwidth algorithm based on semidefinite-programming [24]. Interestingly, the minimum bandwidth problem is NP-hard even for (guest) trees, while graph retraction to (host) trees is solvable in polynomial time. Conversely, the bandwidth problem is solvable in time \(n^b\) for bandwidth \(b\) graphs [68], while graph retraction to a cycle is NP-complete even when the host cycle has only four vertices. Nevertheless, it is conceivable that volume-respecting embeddings, in combination with random projection, could lead to effective approximation algorithms for graph retraction to a cycle in a manner similar to what was achieved for VLSI layout on the plane [115]. Also related are the well-studied variants of linear and circular arrangements, but the objective functions of these problems are average stretch, as opposed to maximum stretch. Finally, another related area is that of low-distortion embeddings (e.g., [77]), where recent work has considered embedding one specific \(n\)-point metric to another \(n\)-point metric [84, 93, 9] similar to the graph retraction problem. But low-distortion embeddings typically require non-contracting isomorphic maps, which distinguishes them significantly from the graph retraction problem.
3.1.3 Chapter roadmap

In Section 3.2 we give an algorithm for retracting a graph to a cycle, proving Theorem 15. In Section 3.3 we show the connection between improving the approximation ratio in Theorem 15, and Sperner’s Lemma. Finally, in Section 3.4 we give an exact algorithm for retracting a graph to cycle, proving Theorem 16.

3.2 Retracting an arbitrary graph to a cycle

In this section, we give an algorithm for retracting an arbitrary graph to a cycle over a subset of vertices of the graph. Let $G$ denote the guest graph over a set $V$ of $n$ vertices, with shortest path distance function $d_G$. Let $H$ denote the host cycle with shortest path distance function $d_H$ over a subset $A \subseteq V$ of $k$ anchors.

Arguably, the simplest lower bound on the optimal stretch is the distance-based bound $\ell(G, H) = \max_{u, v \in A} d_H(u, v)/d_G(u, v)$, since every retraction places a path of length $d_G(u, v)$ in $G$ on a path of length at least $d_H(u, v)$ in $H$.

We now present our algorithm (Algorithm 1), which achieves a stretch of $\min\{k/2, \ell(G, H)\sqrt{n}\}$. Here, we give a high level overview of the algorithm. The first step of algorithm is to embed the input graph $G$ into a grid of size $k/4 \times k/4$ subject to some constraints. The second step is to find the largest empty sub-grid $D$ such that no point is mapped inside of $D$ and center of $D$ is within a desirable distance from center of grid $M$. And final step is to project the points in grid $M$ to its boundary with respect to center of sub-grid $D$.

We now show how to implement the first step of Algorithm 1. Our goal is to embed each vertex $u \in G$ to some point $g(u)$ in a $k/4 \times k/4$ grid such that for every $u, v$, we have the following inequality, where $d_\infty(x, y)$ denotes the $L_\infty$ distance between $x$ and $y$.

$$d_\infty(g(u), g(v)) \leq \ell(G, H)d_G(u, v) \quad (3.1)$$
Algorithm 1 Algorithm for retracting an arbitrary graph to a cycle

**Input:** Graph $G$, host cycle $H$
**Output:** Embedding function $f$

**Embedding in a grid:** Determine embedding $g$ from $G$ into a $k/4 \times k/4$ grid $M$ such that $H$ is embedded one-to-one to the boundary of $M$ and for every $u, v \in V$, $d_{\infty}(g(u), g(v)) \leq \ell(G, H) d_G(u, v)$.

**Find largest hole:** Find the largest square sub-grid $D$ of $M$ such that (a) its center $c$ is at $L_\infty$ distance at most $k/16$ from the center of $M$ and (b) there is no vertex $u$ in $G$ for which $g(u)$ is in the interior of $D$.

**Projection embedding:** For all $v$ in $G$:

1. $R(v) \leftarrow$ ray originating from the center of $D$ and passing through $g(v)$.
2. $f(v) \leftarrow$ the anchor on the boundary of grid $M$ nearest in the clockwise direction to the intersection of $R(v)$ with the boundary of $M$.

return $f$

Additionally, we require that $H$ is embedded to the boundary of the grid, such that adjacent anchors lie on adjacent grid points.

**Lemma 17.** For every $G$, we can find an embedding $g$ satisfying inequality 3.1.

**Proof.** We incrementally construct the embedding $g$. Initially, we place the anchors on the boundary of the grid so that the boundary is isometric to $d_H$. (This can be done since $H$ forms a uniform cycle.) Since $d_{\infty}(g(u), g(v)) \leq d_H(u, v)$ and $d_H(u, v) \leq \ell(G, H) d_G(u, v)$, inequality 3.1 holds for all anchors $u$ and $v$ in $H$.

We next inductively embed the remaining vertices of $G$. Suppose we need to embed vertex $v_i$, and vertices $U = v_1, \ldots, v_{i-1}$ have already been embedded. Assume inductively that the embedding of the vertices of $U$ satisfies inequality 3.1 for the vertices in $U$.

Let $B_{\infty}(g(u), r)$ denote the $L_\infty$ ball around $g(u)$ with radius $r$ (note that these balls are axis-aligned squares). Let $x$ be any point in $\bigcap_{u \in U} B_{\infty}(g(u), \ell(G, H) d_G(u, v_i))$. If we set $g(v_i) = x$, then inequality 3.1 holds for all points in $U \cup \{v_i\}$. We now show that this intersection is nonempty (it is straightforward to find an element in the intersection). The set of axis aligned squares has Helly number 2; there-
fore it is enough to show that for every \( u, u' \in U \), \( B_x(g(u), \ell(G, H)d_G(u, v_i)) \) and \( B_x(g(u'), \ell(G, H)d_G(u', v_i)) \) intersect. If not, then
\[
d_x(g(u), g(u')) > \ell(G, H)(d_G(u, v_i) + d_G(u', v_i)) \geq \ell(G, H)d_G(u, u').
\]
This contradicts our induction hypothesis that the set of vertices in \( U \) satisfies inequality 3.1, and completes the proof of the lemma.

In the following lemma, we analyze the projection embedding step of the algorithm.

**Lemma 18.** Suppose \( r \) is the side length of the largest empty square \( D \) inside \( M \). Then for any vertices \( u \) and \( v \) in \( G \), \( d_H(f(u), f(v)) \) is at most \( 1 + (10\sqrt{2}k/r)d_x(g(u), g(v)) \).

*Proof.* For any point \( x \), let \( \pi(x) \) denote the intersection of the boundary of \( M \) and the ray from the center \( c \) of \( D \) passing through \( x \). Note that for any vertex \( v \) in \( G \), \( f(v) \) is the anchor in \( H \) nearest in clockwise direction to \( \pi(g(v)) \). We show that for any \( x, y \in M \), the distance between \( \pi(x) \) and \( \pi(y) \) along the boundary of \( M \) is at most \( (10\sqrt{2}k/r)d_x(x, y) \).

We first argue that it is sufficient to establish the preceding claim for points on the boundary of \( D \), at the loss of a factor of \( \sqrt{2} \). Let \( x \) and \( y \) be two arbitrary points in \( M \) but not in the interior of \( D \). Let \( x' \) (resp., \( y' \)) denote the intersection of \( R(x) \) (resp., \( R(y) \)) and the boundary of \( D \). From elementary geometry, it follows that \( d(x', y') \leq d(x, y) \), where \( d \) is the Euclidean distance; since \( d_\infty(x, y) \geq d(x, y)/\sqrt{2} \) and \( d_\infty(x', y') \leq d(x', y') \), we obtain \( d_\infty(x', y') \leq \sqrt{2}d_\infty(x, y) \). Since \( \pi(x) = \pi(x') \) and \( \pi(y) = \pi(y') \), establishing the above statement for \( x' \) and \( y' \) implies the same for \( x \) and \( y \), up to a factor of \( \sqrt{2} \).

Consider points \( x \) and \( y \) on the boundary of \( D \). We consider three cases. In the first two cases, \( x \) and \( y \) are on the same side of \( D \). In the first case (Figure 3.1a), \( \pi(x) \) and \( \pi(y) \) are on the same side of the boundary of \( M \) and segment \( \overline{\pi(x)\pi(y)} \) is
(a) Points $x$ and $y$ are on the same side of square $D$, the same side of square $D$, Points $x$ and $y$ (resp. and points $\pi(x)$ and $\pi(y)$ and points $\pi(x)$ and $\pi(y)$) are are on one side of bound- are on one side of boundary anywhere on the boundary ary of $M$ parallel to seg- of M orthogonal to segment of $D$ (resp. on the boundary ary of $M$)

(b) Points $x$ and $y$ are on the same side of square $D$, and points $\pi(x)$ and $\pi(y)$ are on one side of the boundary of $M$, and segment $\pi(x)\pi(y)$ is orthogonal to segment $xy$. In this case, w.l.o.g. assume that $\pi(y)$ is closer to center $c$ than $\pi(x)$ with respect to $d_c$ distance. Let point $z$ be a point on segment $c\pi(x)$ such that segments $\pi(y)$ and $\pi(x)z$ are parallel. From center $c$ extend a line parallel to segment $\pi y$ until it hits the side of $M$ on which $\pi(x)$ and $\pi(y)$ are. Let $w$ be the intersection. Using elementary geometry and similarity argument, we have the following:

$$\frac{|\pi(x)\pi(y)|}{|z\pi(y)|} = \frac{|\pi(x)w|}{|cw|} \leq \frac{k}{k/16} = 4 \quad \text{and} \quad \frac{|z\pi(y)|}{w} = \frac{\pi(y)w}{r} \leq \frac{k}{4r}$$

We thus obtain $\frac{|\pi(x)\pi(y)|}{|xy|} \leq k/r$. For the third case (Figure 3.1c), we observe that $d_{x}(x,y)$ is at least half the shortest path between $x$ and $y$ that lies within the boundary of $D$. This latter shortest path consists of at most five segments, each
residing completely on one side of the boundary of $D$. We apply the argument of the first and second case to each of these segments to obtain that the distance between $\pi(x)$ and $\pi(y)$ is at most $10kd_x(x, y)/r$.

To complete the proof, we note that distance between anchor nearest (clockwise) to $\pi(x)$ and anchor nearest (clockwise) to $\pi(y)$ is at most one plus the distance between $\pi(x)$ and $\pi(y)$. Therefore, the $d_H(f(u), f(v))$ is at most $1+10\sqrt{2kd_x(g(u), g(v))/r}$.

Finally, we prove Theorem 15 using Algorithm 1.

**Theorem 19.** Algorithm 1 computes a retraction of $G$ to the cycle $H$ with stretch at most the minimum of $k/2$ and $O(\sqrt{n})$ times the optimal stretch.

**Proof.** By Lemma 17, the embedding $g$ satisfies inequality 3.1 for every $u$ and $v$ in $G$. By a straightforward averaging argument, there exists a square of side length $k/(8\sqrt{n})$ whose center is at $L_\infty$ distance at most $k/16$ from the center of $M$ and which does not contain $g(u)$ for any $u$ in $V$. By Lemma 18, the projection embedding ensures that for any $u$ and $v$ in $V$, $d_H(f(u), f(v))$ is at most $1+O(\sqrt{n})\ell(G, H)d_G(u, v)$. Since the distance in $H$ cannot exceed $k/2$, the claim of the theorem follows.

**3.3 The Sperner bottleneck**

We show that we cannot improve on the approximation ratio in Theorem 19 using only the distance-based lower bound. In particular, we show a connection to Sperner’s Lemma which our algorithm is not able to leverage. Our proof uses Sperner’s lemma to give a lower bound the a grid graph. In spite of this gaps, we show that the grid is not a particularly challenging instance of the problem. In fact, in Section 3.4, we give an exact algorithm for for retraction in planar graphs, of which the grid is an example. Our constructive result, while using planarity exten-
sively, suggests that there might be a general technique for addressing the Sperner bottleneck described in here.

Consider the following instance: the guest graph $G$ is the $\sqrt{n} \times \sqrt{n}$ grid, and the host $H$ is the cycle of $G$ formed by the $4\sqrt{n}$ vertices on the outer boundary of $G$. It is easy to see that the distance-based lower bound has a value of 2 on this instance. On the other hand, using Sperner’s Lemma from topology, we show that a stretch of $o(\sqrt{n})$ is ruled out:

**Lemma 20.** The optimal stretch achievable for an $n$-vertex grid is at least $2\sqrt{n}/3$.

*Proof.* Suppose we triangulate the grid by adding northwest-to-southeast diagonals in each cell of the grid. Consider the following coloring of the boundary with 3 colors. Divide the boundary $H$ into three segment, each consisting of a contiguous sequence of at least $[4\sqrt{n}/3]$ vertices; all vertices in the first, second, and third segment are colored red, green, and blue, respectively. Let $f$ be any retraction from $G$ to $H$. Let $c_f$ denote the following coloring for $G \setminus H$: the color of $u$ is the color of $f(u)$. By Sperner’s Lemma [114], there exists a tri-chromatic triangle. This implies that there are two vertices within distance at most two in $G$ that are at least $4\sqrt{n}/3$ apart in the retraction $f$, resulting in a stretch of at least $2\sqrt{n}/3$. □

Note that $k = \Theta(\sqrt{n})$ in this instance, so the above lemma also rules out an $o(k)$ approximation using the distance-based lower bound.

### 3.4 Retracting a planar graph to a cycle

The main result of this section is the following theorem.

**Theorem 21.** Let $G$ be a planar graph and $H$ a cycle of $G$. Then there is a polynomial time algorithm that finds a retraction from $G$ to $H$ with optimal stretch.
We begin by presenting some useful definitions and elementary claims in Section 3.4.1. We then present an overview of our algorithm in Section 3.4.2. Finally, we present the algorithm and its analysis, leading to the proof of Theorem 21.

3.4.1 Preliminaries

We begin with a simple lemma that reduces the problem of finding a minimum-stretch retraction to the problem of finding a stretch-1 retraction, in polynomial time. Formally, suppose we have an algorithm $\mathcal{A}$ that, given graphs $G$ and $H$ either finds a stretch-1 retraction from $G$ to $H$, or proves that no such retraction exists.
Then, we can use this algorithm to find the minimum stretch embedding of $G$ into $H$, using Lemma 22 below. Let $G_k$ be the graph where we replace each edge $e \in G, e \notin H$ with a path of $k$ edges. Clearly, $G_k$ can be computed in polynomial time.

**Lemma 22.** $G$ can be retracted to $H$ with stretch $k$ if and only if $G_k$ can be retracted in $H$ with stretch-1.

*Proof of Lemma 22.* Suppose $G$ can be embedded with stretch $k$ in $H$, and let $f : V(G) \to V(H)$ be this mapping. We define an embedding $f' : G_k \to H$ with stretch 1. For $(u,v) \in G$ let $u = y_0, y_1, \ldots, y_k = v$ be the vertices on the path in $G_k$ corresponding to edge $(u,v)$ in $G$. We show how to embed this path into $H$. We know that $d_H(f(u), f(v)) \leq k$. Let $f(u) = x_0, x_1, \ldots, x_j = f(v)$ be the shortest path between $f(u)$ and $f(v)$ in $H$ (note that $j \leq k$). Then, the retraction $f'$, defined by setting $f'(y_i)$ to be $x_{\min(i,j)}$, has stretch 1.

Conversely, a mapping that produces a stretch-1 retraction of $G_k$ into $H$, when restricted to the vertices in $G$, gives a stretch-$k$ retraction of $G$ into $H$. $\square$

We show in Lemma 24 that degree-1 vertices can be eliminated. We first state and prove Lemma 23 which will be used in the proof of Lemma 24, then we state and prove Lemma 24.
Lemma 23. \textit{If there is a stretch-1 retraction }\hat{f}\text{ from }G\text{ to subgraph }\hat{G}\text{ of }G, \text{ then there is a stretch-1 retraction }f\text{ from }G\text{ to subgraph }H\text{ of }\hat{G}\text{ if and only if there is a stretch-1 retraction }g\text{ from }\hat{G}\text{ to }H. \text{ Furthermore, }f\text{ can be computed from }\hat{f}\text{ and }g\text{ in polynomial time.}

\textbf{Proof.} One direction follows immediately: \hat{G}\text{ is a subgraph of }G, \text{ so a stretch-1 retraction of }G\text{ to }H\text{ implies the same for }\hat{G}\text{ to }H. \text{ For the other direction, let }g\text{ be a stretch-1 retraction from }\hat{G}\text{ to }H.\text{ Define }f:G\to H\text{ as follows: if }v\in V(\hat{G}), \text{ then }f(v) = \hat{f}(v); \text{ otherwise, }f(v) = \hat{f}(g(v)).\text{ Then }f\text{ is a stretch-1 retraction from }G\text{ to }H. \text{ Clearly, }f\text{ can be computed in polynomial time from }\hat{f}\text{ and }g. \qed

Lemma 24. \textit{Without loss of generality, we can assume }G\text{ is 2-vertex connected.}

\textbf{Proof.} Suppose }G\text{ is not 2-vertex connected, and let vertex }v\text{ be a vertex cut. \text{ Let }G_1\text{ and }G_2\text{ be the disconnected components created after removing }v\text{ from }G. \text{ Since }H\text{ is a cycle, }H\text{ is contained completely in either }G_1\text{ or }G_2. \text{ Suppose WLOG that }H\in G_1.\text{ Mapping every vertex in }G_2\text{ to }v\text{ yields a stretch-1 retraction to }G' = G\backslash G_2.\text{ By Lemma 23, there is a stretch-1 retraction from }G'\text{ to }H\text{ if and only if there is a stretch-1 retraction from }G\text{ to }H. \text{ We can repeat this procedure until we obtain a 2-connected graph.} \qed

Lemmas 22 and 24 apply to general graphs. In the rest of this subsection, we focus our attention on planar graphs. We note that all the transformations in Lemmas 22 and 24 preserve planarity of the graph. In 2-connected planar graph, every face of a plane embedding is bordered by a simple cycle. Finally, we can assume that there is a planar embedding of }G\text{ with }H\text{ bordering the outer face. If this is not the case, }G\backslash H\text{ contains at least two connected components, which can each be retracted independently.}

Next, we give some definitions related to planar graphs. We call }G\text{ triangulated if it is maximally planar, i.e., adding any edge results in a graph that is not planar.
Equivalently, $G$ is triangulated if every face of a plane embedding (including the outer face) of $G$ has 3 edges. We will make use of the Jordan curve theorem, which says that any closed loop partitions the plane into an inner and outer region (see e.g. [8]). In particular, this implies that any curve crossing from the inner to the outer region intersects the loop. For some cycle $C$ in $G$ and a plane embedding of $G$, we denote the subset of $\mathbb{R}^2$ surrounded by $C$ as $R_C$ (including the intersection with $C$ itself). We say that $R \subset \mathbb{R}^2$ is inside cycle $C$ of $G$ for a plane embedding if $R \subseteq R_C$. If $R$ is inside $C$, we also say that $C$ surrounds $R$. Alternatively, we say that $C$ surrounds $R$. In a slight abuse of notation, we say $C$ surrounds subgraph $G'$ of $G$ for some fixed plane embedding, if $C$ surrounds the subset of $\mathbb{R}^2$ on which $G'$ is drawn in the plane embedding.

### 3.4.2 Overview of our algorithm

Consider some plane embedding of graph $G$ such that $H$ is the subgraph of $G$ bordering $G$’s outer face. We give a polynomial-time algorithm that finds a stretch-1 retraction from $G$ to $H$ or proves that none exists. Using Lemma 22, this immediately yields an algorithm that finds a minimum stretch retraction from $G$ to $H$.

Fix a planar embedding of $G$, let $H$ be defined as above, and let $F$ be a bounded face of $G$. A key component of our algorithm is to find a suitable set of curves connecting $F$ to $H$. Our aim is to find a set of $k = |V(H)|$ curves in $\mathbb{R}^2$ such that the following hold.

- Each curve begins at a distinct vertex of $F$ and ends at a distinct vertex of $H$.
- The curves do not intersect each other.
- A curve that intersects an edge of $G$ either contains the edge, or intersects the edge only at its vertices.
- Each curve lies totally in $R_H \setminus F$.
(a) A graph $G$. The outer cycle $H$ and the face $F$ are partitioned in bold.

(b) Non-intersecting curves partition the region contained in $H$ but not in $F$.

(c) Vertices on $\ell_1$ are mapped to $u$, and vertices on $\ell_2$ are mapped to $v$. All other vertices in the region are mapped arbitrarily to $u$ or $v$.

Figure 3.2: Using non-intersecting curves to find an embedding from face $F$ to $H$.

We call curves with these properties *valid* with respect to $F$. We argue that the curves partition $R_H \setminus F$ (up to their boundaries being duplicated) into a set of regions. Each of these regions is defined by the subset of $\mathbb{R}^2$ surrounded by the closed loop made up of two of the aforementioned curves, a single edge of $H$, and a path on the boundary of $F$.

Given a face $F$ and a set of curves valid with respect to $F$, we can give a stretch-1 retraction from $G$ to $H$. In essence, the curves partition the graph into regions such that all vertices in a particular region map to one of two end-points of a particular edge of $H$. See Figure 3.2 for an illustration.

Of course, it is not obvious that a valid set of curves exists for a given face, and, if it does, how to compute it. We show that if the graph has a stretch-1 retraction, then there is some face $F$ with $k$ valid curves, and that we can efficiently compute them. Our algorithm (Algorithm 2) iterates over all faces in the graph, in each case finding the maximum number of valid curves it can with respect to that face. The number of valid curves we can find is the length of the shortest cycle surrounding $F$. If the shortest cycle $C$ surrounding $F$ has length $\ell$, then it is impossible to find
more than $\ell$ valid curves with respect to $F$: By the Jordan curve theorem, each curve
must intersect $C$, and by the definition, valid curves do not intersect each other and
can intersect $C$ only at its vertices. Our construction of the valid curves shows that
this is tight (i.e. we can always find $\ell$ curves). We show that if a stretch-1 retraction
exists, then there is some face for which $\ell = k$. Algorithm 2 gives an outline of the
algorithm.

Algorithm 2 Outline for finding a stretch-1 retraction, or proving that none exists.
1: for inner face $F$ in $G$ do
2: Compute maximum number of valid curves between $F$ and $H$ $p_1, \ldots, p_\ell$
3: if $\ell = k$ then
4: Compute stretch-1 retraction from $G$ to $H$ using $p_1, \ldots, p_k$
5: end if
6: end for
7: If no retraction was computed, report no stretch-1 retraction exists

3.4.3 Algorithm and analysis

This section gives the details of various components of Algorithm 2, and provides a
proof of correctness. The following is an outline of the rest the section:

1. Lemma 26 shows how to compute a stretch-1 retraction using the $k$ valid curves
   in line 4 of Algorithm 2.

2. Next, Lemma 27 shows that if a stretch-1 retraction exists, there must be some
   face $F$ in the graph such that the smallest cycle surrounding $F$ has length $k$.

3. Finally, Lemma 29 gives a construction of largest set of valid curves for a given
   face $F$ from line 2, and shows that the number of curves computed equals the
   length of the smallest cycle surrounding $F$.

We begin by showing in Lemma 25 a somewhat obvious fact: A set of valid
curves partition $R_H \setminus F$, and each region of the partition contains a single edge of $H$. 
We then show in Lemma 26 that this partition can be used to produce a stretch-1 embedding. See Figure 3.2 for pictorial presentation of these two lemmas.

**Lemma 25.** Let \( p_1, \cdots, p_k \) be a of curves that are valid with respect to \( F \). Let \( Z \) denote the set of faces of \( H \cup F \cup \bigcup_i p_i \) excluding the outer face and \( F \). Then, each face \( f \in Z \) is bordered by exactly 1 edge of \( H \), and every vertex of \( G \setminus \bigcup_i p_i \) is in a unique face of \( Z \).

**Proof.** Consider the faces of \( H \cup F \cup \bigcup_i p_i \). \( H \) and \( F \) still define faces since the paths \( p_i \) fall in \( R_H \setminus F \). Let \( p_{u,v} \) be an edge of \( H \), and consider \( X = p_i \cup (u,v) \cup p_j \cup p_F(i,j) \) where \( p_i \) is the path containing \( u \), \( p_j \) is the path containing \( v \), and \( p_F(i,j) \) is the path on the boundary of \( F \) between the vertices where \( i \) and \( j \) meet \( F \) such that \( F \) is not contained in \( X \). If \( p_i \) and \( p_j \) are both degenerate (i.e., each is empty), then \( (u,v) = p_F(i,j) \). Otherwise \( X \) is a simple cycle. We claim that \( X \) defines a face. In particular, we show that the path \( p_F(i,j) \) contains no other vertex of path \( p_z \) for all \( z \neq i,j \). Suppose it does and let \( w \) be that vertex. Let \( w' \) be the vertex adjacent to \( w \) on \( p_z \). Then \( w' \in R_H \setminus F \), and so \( w' \in X \). The other end of path \( p_z \), call it vertex \( y \), is in \( H \), but \( y \neq u,v \). By the Jordan curve theorem, \( p_z \setminus w \) must cross \( X \). Since the graph is planar, \( p_z \setminus w \) must contain a vertex of \( F,H,p_i \), or \( p_j \). Any of these outcomes leads to a contradiction. \( \square \)

**Lemma 26.** Given a non-outer face \( F \) and a set \( \{p_1, p_2, \ldots, p_k\} \) of curves that are valid with respect to \( F \), we can construct a stretch-1 retraction from \( G \) to \( H \) in polynomial time.

**Proof.** Let \( Z \) be as defined in Lemma 25. For each vertex \( w \) on \( p_i \), map \( w \) to the unique vertex \( v \in H \cap p_i \). Otherwise, map \( w \) to \( u \) or \( v \), where \( (u,v) \) is the unique edge of \( H \) contained in the same face of \( Z \) as \( w \). Fix a face \( f \) of \( Z \). Let \( (u,v) \) be the unique edge of \( H \) contained in \( f \). Any edge \( (x,y) \) contained in \( f \) also has \( x,y \in f \),
and so $x$ and $y$ are each mapped to either $u$ or $v$. Thus, this retraction to $H$ has stretch 1.

As mentioned earlier, we will show that our construction produces $\ell$ valid curves for face $F$, where $\ell$ is the minimum length cycle surrounding $F$. So we must show that if a stretch-1 retraction exists, there is some $F$ such that every cycle surrounding $F$ has length at least $k$.

**Lemma 27.** Fix a plane embedding of $G$ where $H$ defines the outer face of the embedding and suppose there is a stretch-1 retraction $G$ to $H$. Then there exists a non-outer face $F$ such that every cycle surrounding $F$ has length at least $k$.

**Proof.** We prove a related claim that implies the statement in the lemma. Fix some stretch-1 retraction of $G$ to $H$. Then there exists a non-outer face $F$ such that for every cycle $C$ in the set of cycles surrounding $F$, and for each vertex $v \in H$, there is some vertex of $C$ mapped to $v$. This implies that each of these cycles has length at least $k$, since the statement says that vertices of $C$ are mapped to $k$ vertices of $H$.

The claim is very similar to Sperner’s lemma, and the proof is similar as well. Let $\phi : V(G) \to V(H)$ denote the retraction. We associate a score with each cycle $C$ of the graph: Order the vertices of the cycle in clockwise order, denoted $v_1, v_2, \ldots, v_j, v_{j+1} = v_1$. Consider the sequence $\phi(v_1), \ldots, \phi(v_j), \phi(v_{j+1})$. Let the score of $C$ be 0 to start. For each pair $\phi(v_i), \phi(v_{i+1})$, we have: either $\phi(v_i) = \phi(v_{i+1})$, or $\phi(v_i)$ and $\phi(v_{i+1})$ are adjacent in $H$. If $\phi(v_{i+1})$ is clockwise of $\phi(v_i)$ (i.e. if they are in the same order as on $C$), add 1 to the score of $f$. If they are in counterclockwise order, subtract 1. If they are the same vertex, the score remains the same. If $\phi(v_1), \ldots, \phi(v_j)$ does not contain every vertex on the outer cycle, the score of $C$ must be 0, since each edge along the path $\phi(v_1), \ldots, \phi(v_{j+1})$ is traversed exactly the same number of times in each direction. On the other hand, a cycle with a non-zero score must have a score that is divisible by $k$. 

58
Next, we claim that the score of cycle $C$ is the same as the sum of the scores of the cycles defining the faces contained in $C$. To see this, consider the total contribution to the scores of these cycles from any fixed edge. If the edge is not in cycle $C$, it is a member of exactly 2 faces contained in $C$, and contributes either 0 to both faces, or $-1$ to one and 1 to the other. Edges in $C$ are each a member of just one face surrounded by $C$. Therefore, the score of cycle $C$ is the same as the sum of scores of its surrounded faces. Since the score of cycle $H$ is $k$, there must be some face $f$ that has non-zero score.

Finally, we show that there is some face with nonzero score such that every cycle surrounding the face also has nonzero score. Suppose this is not the case. Then, every face with a non-zero score is surrounded by a cycle with score 0, which implies that the sum of all scores of faces with non-zero scores is 0. This is a contradiction, since it implies that the sum of scores of all internal faces in the graph is 0.

We complete the section by giving a construction of the maximum number of valid curves with respect to some face $F$, and show that the number of curves is $\ell$, where $\ell$ is the smallest cycle surrounding $F$. Our curves will be disjoint paths in a supergraph $G_\Delta(F)$ of $G$. It is necessary to relate the maximum number of disjoint paths we can find to the length of the shortest cycle surrounding $F$. The following lemma establishes this connection.

**Lemma 28.** Let $G$ be a triangulated graph. The graph induced by any minimum $s$-$t$ vertex cut is the shortest simple cycle separating $s$ and $t$.

**Proof of Lemma 28.** Consider the dual graph $G^*$ for some planar embedding of $G$. This graph is constructed by placing a vertex $u^*$ in each face of $G^*$, and adding an edge between $u^*$ and $v^*$ if the faces are adjacent in $G$. Note that there is a correspondence between vertices of $G$ and faces of $G^*$, as well as between faces of $G$ and vertices of $G^*$. Additionally there is a one-to-one correspondence between edges.
It is well known that $X \subseteq E$ contains an $s-t$ cut if and only if $X^*$ contains the edges of some simple cycle separating $s$ from $t$. We will use this fact to prove our result.

We need to show that a set of vertices $Y$ in $G$ separates $s$ from $t$ if and only if the subgraph induces by $Y$ contains a cycle separating $s$ from $t$. One direction is straightforward: if the induced graph on $Y$ forms a cycle separating $s$ from $t$, then applying the Jordan curve theorem tells us that any path from $s$ to $t$ must cross this cycle (and thus contain a vertex of $Y$). To prove the converse, we use the fact given above. Let $Y$ be an $s-t$ vertex cut in $G$ and let $E_Y$ be the set of edges in the graph induced by $Y$. Then from the fact above, $E_Y^*$ in the dual graph $G^*$ contains a cycle $C^*$ separating $s$ and $t$. Because the graph is triangulated, we can show that $E_Y$ also contains a separating cycle: edges $e_1^*$ and $e_2^*$ that are adjacent in $E_Y^*$ correspond to edges $e_1$ and $e_2$ falling on the same face of $G$. Because each face of $G$ has only 3 edges, $e_1$ and $e_2$ must therefore be adjacent. Therefore, $C$ (the edges corresponding to $C^*$) must also be a cycle. 

If $G$ was already triangulated, we could compute a set of vertex disjoint paths from $F$ to $H$ (note that a set of vertex disjoint paths yields a set of valid curves). By Menger’s theorem and Lemma 28, we would find $\ell$ paths, where $\ell$ is the shortest cycle surrounding $F$. $G$ may not be triangulated, so instead we could first triangulate $G$ and then compute the paths. However, the number of paths we find in this case is the length of the shortest cycle surrounding $F$ in the triangulation of $G$, which may be smaller than $\ell$. We prevent this from happening by producing a triangulation that adds vertices as well as edges.

**Lemma 29.** Fix a planar embedding of $G$ with $H$ as the outer face, and let $F$ be other face. Then we can compute $\ell$ valid curves in polynomial time, where $\ell$ is the length of the shortest cycle surrounding $F$. 

60
Proof. We build a triangulated graph $G_\Delta(F)$ from the planar embedding of $G$. First, add vertices and edges to every face of $G$, excluding the outer face and $F$. We do this in a way such that (1) every face except $F$ and the outer face is a triangle, and (2) the distance between any $u, v \in G$ is preserved. The construction is iterative. From each face with more than 3 edges, we create one new face that has one fewer edge. One step of this construction is shown in Figure 3.3.

Note that distances are preserved inductively, and we make progress by reducing the size of some face. The graph we produce has 3 edges bordering each face, except for the outer face and $F$. In all, the number of vertices and edges added to each face of $G$ is polynomial in the number of edges bordering the face.

Finally, we add vertices $s$ and $t$, and edges from $s$ to each vertex of $F$ and from $t$ to each vertex of $C$. The resulting graph is triangulated, and we call this graph $G_\Delta(F)$.

At this point, we can find the maximum set of vertex disjoint paths between $s$ and $t$ in $G_\Delta(F)$, by setting vertex capacities to 1 and computing a max flow between $s$ and $t$. Because we have preserved distances between vertices of $G$ in our construction of $G_\Delta(F)$, the length of the minimum cycle surrounding $F$ must be $\ell$. Therefore, the
number of disjoint paths we find must also be \( \ell \).

Finally, we claim that this set of disjoint paths from \( F \) to \( H \) in \( G_\Delta(F) \) is a set of valid curves for \( G \). This is because \( G \) is a subgraph of \( G_\Delta(F) \), and therefore the criteria for valid curves are still met after removing the vertices and edges of \( G_\Delta(F) \setminus G \).

We conclude by tying together the pieces of the section to show we proved Theorem 21.

*Proof of Theorem 21.* Fix a face \( F \). By Lemma 28, we determine the set of \( \ell \) disjoint paths from \( F \) to \( H \) where the surrounding minimum cycle is of length \( \ell \). By Lemma 27, there is a stretch-1 retraction only if there exists a face \( F \) whose surrounding min-cycle is of length \( k \). So if there is no stretch-1 retraction, we find \( < k \) disjoint paths for all faces, and our algorithm returns “no”. Otherwise, there exists a face \( F \) for which the surrounding min-cycle is of length \( k \), and this gives a set of \( k \) valid paths. Then, by Lemma 26, the retraction that we construct has stretch 1. \( \square \)
4

Symmetric Interdiction

4.1 Introduction

A recent study of malicious network traffic observed at Microsoft data centers [95] made the surprising observation that a large volume of attack traffic originated from virtual machines hosted within the data centers themselves. The machines generating these attacks may have been compromised, or they may have been rented with stolen credit cards or on a free-trial basis. While the authors of the study used heuristics to identify traffic that was obviously malicious, in general it is very difficult to distinguish legitimate traffic from malicious traffic. In particular, an attacker in possession of a “botnet” of compromised machines can launch a denial-of-service attack against a service simply by using these machines to send a large number of legitimate-looking requests to the servers that implement the service.

The following question then arises: how does a network operator decide which connection requests to admit if she cannot distinguish between legitimate and malicious requests? One natural strategy is to minimize \textit{regret}: the number of legitimate requests that are not served but might have been otherwise. This motivates us to
define the symmetric interdiction model in this paper, where the goal is to select a feasible set of edges whose removal minimizes the maximum feasible set in the remaining graph. We give a general framework for converting algorithms for a broad class of optimization problems to algorithms for the corresponding symmetric interdiction problems.

We instantiate our general model in the symmetric matching interdiction problem (abbreviated smi in the rest of the paper), where the goal is to select a matching whose removal minimizes the maximum matching in the remaining graph. The smi problem models our motivating scenario. Suppose clients located in a data center issue requests to servers in the same data center, where each client and each server has the capacity to participate in a single client-server interaction. Each client provides the operator of the data center with a list of servers it would like to contact, and the operator selects a matching of clients and servers. The operator would prefer to prioritize legitimate requests, but cannot distinguish between legitimate and malicious clients. By minimizing the size of the remaining maximum matching, an optimal solution to the smi problem bounds the number of legitimate requests that are not satisfied but might otherwise have been. For the smi problem, we show hardness results, and give a carefully designed algorithm that improves upon the result obtained from the general framework.

4.1.1 Our results and chapter structure

Consider a generic optimization problem \( \Pi \) that is specified by an input graph \( G = (V, E) \), by a set \( \mathcal{F} \) of subgraphs of \( G \) which constitute feasible solutions to the problem, and a maximization (resp., minimization) objective function \( f \) on graphs. An example of \( \Pi \) is the maximum matching problem: \( \mathcal{F} \) is the set of all matchings and the function \( f \) returns the number of edges in the matching. For the optimization problem \( \Pi \), we define the symmetric interdiction problem \( I(\Pi) \) as follows: the goal
is to produce a subgraph $H = (V, F)$ of $G$ such that $H$ is in $\mathcal{F}$ and minimizes (resp., maximizes) the optimum value of $f$ achievable on the remaining graph $(V, E \setminus F)$. Thus, the symmetric matching interdiction (SMI) problem is given a graph $G$ and seeks a matching $M$ of $G$ so as to minimize the maximum matching in $G \setminus M$.

Our first result is a general framework for converting optimization algorithms to symmetric interdiction algorithms for a broad class of problems. This result, described informally below, is stated formally in Theorem 32 and proved in Section 4.2.

**Theorem 30 (Informal).** An $\alpha$-approximation to a packing problem $\Pi$ implies a $(1 + \alpha)$-approximation to the corresponding symmetric interdiction problem $I(\Pi)$, modulo some technical conditions.

Next, we focus on the SMI problem. Theorem 32 implies that any maximum matching algorithm is a 2-approximation algorithm for this problem. In fact, we show that any maximal matching also achieves an approximation factor of 2. However, this is the limit of the general framework in the sense that there are graphs where a maximum matching has an approximation factor of exactly 2 for the SMI problem. Our main algorithmic contribution is to obtain a more careful algorithm for the SMI problem that obtains an approximation factor of 1.5. We complement this result with a proof of APX-hardness of the problem by giving an approximation lower bound of $(1 + \epsilon)$ for small but fixed positive $\epsilon$.

**Theorem 31.** There is a polynomial-time deterministic algorithm for the symmetric matching interdiction problem with an approximation factor of 1.5. Moreover, the symmetric matching interdiction problem is APX-hard.

**Extensions**

We consider a randomized variant of the SMI problem in Section 4.5. Specifically, we show that if the interdictor is allowed to use randomness that is invisible to
the optimizer, then the SMI problem becomes polynomial time solvable. Another natural extension of the SMI problem that captures practical parameters arising in networking is the capacitated case, where every edge has a capacity and the input and output ports have maximum capacities on the total amount of network flow that can be routed through them. If the edge capacities are unsplittable and both the interdictor’s and optimizer’s solutions are edge subsets, then the corresponding optimization problem is a special case of the previously studied demand matching problem [109]. Existing results for the optimization problem, applied to our interdiction framework, gives an approximation algorithm for the interdiction problem using Theorem 30. Improving this result using a more specific algorithm, such as the one that we give for the SMI problem, is left as an open problem in this work.

Finally, our symmetric interdiction framework can be applied to other diverse combinatorial optimization problems. See Section 4.6 for a brief discussions on other symmetric interdiction problems.

4.1.2 Related work

Interdiction variants of classical graph optimization problems have attracted considerable research interest in recent years. Typically, these problems are modeled as a two-step game between an interdictor and an optimizer. In the first step, the interdictor removes a limited number of edges from the graph, with the goal of worsening the objective of the optimizer who solves the graph optimization problem on the remaining graph in the second step. For instance, in the matching interdiction problem, the goal is to remove at most $k$ edges (for a given $k$) such that the size of the maximum matching in the remaining graph is minimized [122]. One can similarly define interdiction variants for maximum flow [47, 62, 63, 120, 123, 42, 6, 37, 91], minimum spanning tree [124, 59], and many other classic graph optimization problems [36, 116, 78, 85]. The main distinction between this model and the symmetric
interdiction model is that both the interdictor and the optimizer in our problem are constrained by the same feasibility conditions, whereas the interdictor was constrained by a budget on the number of edges in previous work.

The SMI problem is similar to the matching interdiction problem studied by Kamalian et al. [80, 79], the key difference being that the interdictor’s matching is also required to be a maximum matching in their case. We show that this restriction can produce suboptimal SMI solutions; indeed, the results of [80, 79] have no implication for SMI. More broadly, interdiction problems have a long history, having been studied for military applications in the Cold War [106]. Closer to our work, they have been used to model competitive markets in economic theory. In particular, in the Stackelberg model [119], two firms compete sequentially on the quantity of output they produce of a homogeneous good. Furthermore, both players play by the same rules and therefore must operate under the same constraints. This is conceptually identical to our symmetric interdiction model and we hope that this model will be applied to other domains in the future.

4.2 Symmetric Interdiction: A General Framework

In this section, we give a general theorem that relates symmetric interdiction problems to their corresponding optimization problems for a broad class of optimization problems called packing problems. This includes many classical problems such as maximum matching, knapsack, maximum flow, etc. Formally, packing problems are those that can be encoded by the linear program (LP) given below, where all entries of the coefficient matrix $\mathbf{A}$, and that of vectors $\mathbf{b}$ and $\mathbf{c}$ are non-negative:

$$\text{maximize } \mathbf{c}^T \mathbf{x}, \text{ subject to } \mathbf{A} \mathbf{x} \leq \mathbf{b} \text{ and } 0 \leq \mathbf{x} \leq 1.$$  

(4.1)
Suppose $x$ is a feasible solution to LP (4.1). Then, we define the residual LP of $x$ as:

$$\text{maximize } c^\top y, \text{ subject to } Ay \leq b \text{ and } 0 \leq y \leq 1 - x.$$ (4.2)

The symmetric interdiction problem is to find a feasible solution $x$ that minimizes the optimal solution to the residual LP of $x$. While we only focus on packing problems in this paper, we note that one can analogously define symmetric interdiction for covering problems\(^1\) as well. Additionally, we note that all results in this section hold when $x$ and $y$ are constrained to be integral.

We call LP (4.1) the optimization problem. In this section, we develop a framework to obtain approximate solutions to the interdiction problem using exact/approximate solutions to the optimization problem. Before stating the result formally (Theorem 32), we set up some basic notation. Let

$$x \backslash x' = \left( \begin{array}{c} \max(0, x_1 - x'_1) \\ \vdots \\ \max(0, x_n - x'_n) \end{array} \right).$$ (4.3)

Note that $x \backslash x'$ is feasible if $x$ and $x'$ are feasible.

Let $x^*$ be an optimal solution to the interdiction problem, and let $y^*$ be an optimal solution to the residual LP w.r.t. $x^*$. Now, consider a solution $x$ that is feasible for LP (4.1). Ideally, we would like to claim that if $x$ is an approximately optimal solution for the optimization problem, then it is also an approximately optimal for the interdiction problem. Unfortunately, this may not be true in general. However, we can show this connection between optimization and interdiction if $x$ satisfies the following stronger condition:

$$c^\top(x^* \backslash x) \leq \alpha \cdot c^\top(x \backslash x^*) \text{ for some approximation factor } \alpha \geq 1.$$ (4.4)

\(^1\) Covering problems are minimization problems where the constraints are $Ax \geq b$, with the same non-negativity restrictions.
This condition says that after removing any overlap between the interdiction and optimization solutions, the approximation ratio must be $\alpha$. For example, consider an optimal interdiction solution $M^*$ to the maximum matching problem, and another matching $M$. After removing edges that appear in both $M$ and $M^*$, the number of remaining edges in $M$ must be within a factor $\alpha$ of the number of remaining edges in $M^*$. In particular, when $x$ is an optimal solution to the optimization problem, condition (4.4) holds with $\alpha = 1$ for any maximization problem. Now, we formally state and prove the theorem that establishes the relationship between optimization and interdiction.

**Theorem 32.** Let $x^*$ be an optimal solution to the interdiction problem, and let $y^*$ be an optimal solution to the residual LP w.r.t. $x^*$. Suppose $x$ is a feasible solution satisfying condition (4.4), i.e., $c^T(x^* \setminus x) \leq \alpha \cdot c^T(x \setminus x^*)$. Then, $x$ is a $(1 + \alpha)$-approximation to the corresponding interdiction problem. That is, if $y$ is an optimal solution to the residual LP of $x$, then $c^T y \leq (1 + \alpha) \cdot c^T y^*$.

**Proof.** We define the intersection $x \cap x'$ to be

$$x \cap x' = \left( \begin{array}{c} \min(x_1, x'_1) \\ \vdots \\ \min(x_n, x'_n) \end{array} \right).$$

Observe that $c^T \cdot y = c^T \cdot (y \setminus (1 - x^*)) + c^T \cdot (y \cap (1 - x^*))$. We upper bound each summand of this equation. We will need to use the fact that $x \setminus x^*$ is a feasible solution to the residual LP of $x^*$. This follows from two observations: (1) $x$ satisfies the constraint $Ax \leq b$ which implies $x \setminus x^*$ does too, and (2) $x \leq 1$ implies $x \setminus x^* \leq 1 - x^*$. 

69
We first upper bound the left summand:
\[
\mathbf{c}^\top \mathbf{y} (1 - \mathbf{x}^*) \leq \mathbf{c}^\top ((1 - \mathbf{x}) (1 - \mathbf{x}^*) )
\]
(since \( \mathbf{y} \) is feasible for the residual LP of \( \mathbf{x} \))
\[
\leq \mathbf{c}^\top (\mathbf{x}^* \mathbf{x})
\]
\[
\leq \alpha \cdot \mathbf{c}^\top (\mathbf{x}^* \mathbf{x})
\]
(by assumption that \( \mathbf{x} \) satisfies (4.4))
\[
\leq \alpha \cdot \mathbf{c}^\top \cdot \mathbf{y}^*
\]
(since \( \mathbf{x}^* \mathbf{x}^* \) is feasible for the residual LP of \( \mathbf{x}^* \), shown above)

Next we bound the right summand. Note that \( \mathbf{y} \cap (1 - \mathbf{x}^*) \) is feasible for the residual LP of \( \mathbf{x}^* \) since \( \mathbf{y} \cap (1 - \mathbf{x}^*) \leq (1 - \mathbf{x}^*) \). Therefore, since \( \mathbf{y}^* \) is optimal for the residual LP of \( \mathbf{x}^* \), we have \( \mathbf{c}^\top \cdot (\mathbf{y} \cap (1 - \mathbf{x}^*)) \leq \mathbf{c}^\top \cdot \mathbf{y}^* \). Putting together the bounds on the left and right summands, we get
\[
\mathbf{c}^\top \cdot \mathbf{y} = \mathbf{c}^\top \cdot (\mathbf{y} (1 - \mathbf{x}^*)) + \mathbf{c}^\top \cdot (\mathbf{y} \cap (1 - \mathbf{x}^*)) \leq \alpha \cdot \mathbf{c}^\top \cdot \mathbf{y}^* + \mathbf{c}^\top \cdot \mathbf{y}^* = (1 + \alpha) \cdot \mathbf{c}^\top \cdot \mathbf{y}^*.
\]

**Corollary 33.** Any optimal solution \( \mathbf{\hat{x}} \) to the optimization problem, is a 2-approximation to the corresponding symmetric interdiction problem.

**Proof.** Note that \( \mathbf{\hat{x}} = (\mathbf{\hat{x}} \mathbf{x}) + (\mathbf{\hat{x}} \mathbf{x}) \). Similarly, \( \mathbf{x}^* = (\mathbf{x}^* \mathbf{x}) + (\mathbf{x}^* \mathbf{x}) \). Since \( \mathbf{\hat{x}} \) is an optimal solution to the optimization problem, \( \mathbf{c}^\top \mathbf{x}^* \leq \mathbf{c}^\top \mathbf{\hat{x}} \). Therefore, \( \mathbf{c}^\top (\mathbf{x}^* \mathbf{x}) \leq \mathbf{c}^\top (\mathbf{\hat{x}} \mathbf{x}) \).

### 4.3 Symmetric Matching Interdiction: A 3/2 Approximation

Let \( G = (V, E) \) be a graph. Then the symmetric matching interdiction (SMI) problem is to find some matching \( M^* \) such that the maximum matching in the graph \( (V, E \backslash M^*) \) is minimized.

From Corollary 33, we get that any maximum matching is a 2-approximation for the SMI problem. In fact, any maximal matching is also a 2-approximation.

**Lemma 34.** Any maximal matching is a 2-approximation for the symmetric matching interdiction problem.
Proof. For a graph $G$, let $M$ be a maximal matching and $L$ be the maximum matching on $G \setminus M$. Each component of $M \cup L$ is a path or a cycle of alternating edges of $M$ and $L$. Any edge that appears by itself in a component of $M \cup L$ must be in $M$, by the maximality of $M$.

Let $C$ be a component of $M \cup L$ that contains at least one edge of $L$. We show that for any matching $M^*$ on $C$, the maximum matching on $C \setminus M^*$ has at least $|L \cap C|/2$ edges, which will complete the proof. Let $j$ be the number of edges of $C$. Then, $|L| = \frac{j}{2}$ if $j$ is even, and $|L| \leq \frac{j+1}{2}$ if $j$ is odd. That is, $|L| \leq \left\lceil \frac{j}{2} \right\rceil$.

We will show later by a case analysis in Lemma 35 that the maximum matching on $C \setminus M^*$ has at least $\left\lceil \frac{j-1}{3} \right\rceil$ edges for any $M^*$. Since $\frac{j}{2}/\left\lceil \frac{j-1}{3} \right\rceil \leq 2$ for integers $j \geq 2$, the lemma follows.

This is better than the 3-approximation guarantee for maximal matchings that we get from Theorem 32. In fact, the approximation factor of 2 is the best achievable, if we were to choose an arbitrary maximum or maximal matching. Consider a length-4 path. The optimal interdiction solution contains the edges at the two ends, leaving a matching of size 1. On the other hand, the first and third edges form a maximum matching, but leaves behind a matching of size 2.

But, what if we choose the best maximum matching instead of an arbitrary one? In the previous example, the optimal interdiction solution also turned out to be a maximum matching. Our first result in this section is to show that there always exists a maximum matching that is a $3/2$-approximation to the optimal interdiction matching. In the second part of this section, we make this result constructive, i.e., give a polynomial-time algorithm for finding such a maximum matching. Before describing our result, we note that the approximation factor of $3/2$ is the best we can hope for from a maximum matching, even the best one. Consider a cycle of length 6. The optimal interdiction solution contains any pair of opposite edges,
leaving behind two disjoint length-2 paths containing a matching of size 2. On the other hand, any maximum matching contains 3 edges, which leaves behind 3 disjoint components forming a matching of size 3.

4.3.1 Approximating the smi problem with maximum matchings

We show that the maximum matching with the largest intersection with any fixed optimal solution to the smi problem is a 3/2 approximation to the smi problem. In this section, $M^*$ denotes an optimal solution to smi, i.e., a matching that minimizes the size of the maximum matching $L^*$ in the remaining graph $(V, E\setminus M^*)$. $M$ denotes a maximum matching on $G$, and $L$ denotes a maximum matching in the remaining graph $(V, E\setminus M)$. All matchings and connected components that we refer to in this section are defined as sets of edges; hence, set operations are only on the edges and do not affect vertices.

For any $M$ and $L$, the size of a matching on $(M \cup L)\setminus M^*$ serves as a lower bound on the size of $L^*$, since $(M \cup L) \subseteq E$. So, our goal will be to show that the size of $L$ is at most 3/2 times the size of a matching that we construct in $(M \cup L)\setminus M^*$. We will show this individually for every component of $M \cup L$. Let $C$ be a component of $M \cup L$. We say $M$ is locally $3/2$-competitive on $C$ with respect to $M^*$ if $C\setminus M^*$ contains a matching of at least $2/3$ times the size of $C \cap L$. If $M$ is locally $3/2$-competitive for each component, then that implies an approximation factor of $3/2$ overall.

For some fixed $M^*$, there are only certain types of components of $M \cup L$ that may not be locally competitive. We call these components critical, and define their structure below. Note that $M \cup L$ is a set of vertex disjoint paths and even-length cycles, since it is composed of two matchings.

**Definition 2.** We call component $C$ critical w.r.t. matching $M^*$ if all the following hold:

72
1. C is an even-length path,
2. the edges at the two ends of C are in $M^*$, and
3. $C \setminus M^*$ is a set of length-2 paths.

We will show in Lemma 35 that critical components, as defined in Definition 2, are the only ones that may not be locally competitive. From Definition 2, for a component to be critical, it must be a path with $\ell$ edges, where $\ell \equiv 4 \mod 6$ edges. We call these components bad:

**Definition 3.** Let $C$ be a component of $M \cup L$. Call $C$ bad if $C$ is a path and $|C| \equiv 4 \mod 6$, where $|C|$ denotes the number of edges in $C$.

Note that all critical components are bad, but not vice-versa, since criticality also depends on the structure of $M^*$.

We next show that $M$ is locally $3/2$-competitive on all components that are not critical. In fact, the lemma gives tighter bounds, which will be helpful in developing an algorithm later. Note that, till now, the only assumption we have made about $M$ is that it is a maximum matching, i.e., the next lemma holds for all maximum matchings.

**Lemma 35.** Fix $M$, $L$, $M^*$, and $L^*$. Let $C$ be a component of $M \cup L$. Let $\ell^*$ denote the size of a maximum matching on $C \setminus M^*$, and $c$ denote the number of edges in $C$. (Note that $\ell^*$, summed over all components $C$, lower bounds the size of $L^*$.) Then,

1. If $C$ is not bad and $c$ is odd, $\ell^* \geq \frac{c-1}{3}$.
2. If $C$ is not bad and $c$ is even, $\ell^* \geq \frac{c}{3}$.
3. If $C$ is bad but not critical, $\ell^* \geq \frac{c+2}{3}$.
4. If $C$ is bad and critical, $\ell^* \geq \frac{c-1}{3}$.
Proof. We find these lower bounds on \( \ell^* \) by constructing a matching \( \hat{L} \) on \( C \setminus M^* \). Note that \( C \setminus M^* \) is either an even cycle or a set of vertex-disjoint paths. In the former case, we pick every alternate edge on the cycle in \( \hat{L} \). In the latter case, for each path, we pick every alternate edge in \( \hat{L} \), including the two edges at the ends for odd length paths. Let \( m^* \) denote \( |M^* \cap C| \). \( \hat{L} \) has the following properties:

1. \( \hat{L} \) contains at least \( \left\lceil \frac{c-m^*}{2} \right\rceil \) edges.

2. For each component of \( C \setminus M^* \), \( \hat{L} \) contains at least one edge.

Next, we show that these two properties are sufficient to prove that for each of the 4 cases in the statement of the lemma, the corresponding inequality holds.

Case (1). Note that \( C \) must be a path, since all cycles have even length in the union of two matchings. Therefore, property 2 ensures that \( \hat{L} \) has at least \( m^* - 1 \) edges. Along with property 1, this implies \( \ell^* \geq |\hat{L}| \geq \min(m^* - 1, \frac{c-m^*}{2}) \). Optimizing over the possible values of \( m^* \) then gives us \( \ell^* \geq \frac{c-1}{3} \).

Case (2). \( C \) is either a path or a cycle. We treat these cases differently.

1. When \( C \) is a cycle, property 2 implies that \( \hat{L} \) has at least \( m^* \) edges. Along with property 1, this implies \( \ell^* \geq |\hat{L}| \geq \min(m^*, \frac{c-m^*}{2}) \). Optimizing over the possible values of \( m^* \) gives \( \ell^* \geq \frac{c}{3} \).

2. When \( C \) is a path, property 2 implies that \( \hat{L} \) has at least \( m^* - 1 \) edges. Identical to case (1) above, we can now infer that \( \ell^* \geq \frac{c-1}{3} \). Since \( \ell^* \) is integral, we can claim that \( \ell^* \geq \lceil \frac{c-1}{3} \rceil \). We also know that \( c \) is even and \( c \not\equiv 4 \mod 6 \). Together, this shows that \( \lceil \frac{c-1}{3} \rceil \geq \frac{c}{3} \), which implies that \( \ell^* \geq \frac{c}{3} \).

Case (3). We subdivide into two cases based on the size of \( M^* \). Note that \( c \equiv 4 \mod 6 \); hence, \( \frac{c+2}{3} \) is an integer.
1. Suppose $m^* \neq \frac{c+2}{3}$. If $m^* \geq \frac{c+2}{3} + 1$, then property (2) implies $\hat{L} \geq \frac{c+2}{3}$. On the other hand, if $m^* \leq \frac{c+2}{3} - 1$, then property (1) ensures that $|\hat{L}| \geq \left\lfloor \frac{c+1/2}{3} \right\rfloor = \frac{c+2}{3}$. In either case, $\ell^* \geq |\hat{L}| \geq \frac{c+2}{3}$.

2. Suppose $m^* = \frac{c+2}{3}$. If $M^*$ does not contain at least one end edge of path $C$, then $C \setminus M^*$ has $m^*$ components, and therefore, property (2) ensures that $\hat{L}$ has at least $m^*$ edges. Now, consider the case where $M^*$ contains both end edges of path $C$. In this case, the number of components in $C \setminus M^*$ is $m^* - 1 = \frac{c-1}{3}$. But, the total number of edges in $C \setminus M^*$ is $c - m^* = \frac{2c-2}{3}$. Therefore, the average number of edges in each component of $C \setminus M^*$ is 2. Since $C$ is not critical w.r.t. $M^*$, every component in $C \setminus M^*$ cannot have exactly 2 edges. As a consequence, there must be at least one component $\alpha$ in $C \setminus M^*$ that contains at least 3 edges. By property (2), $\hat{L}$ contains at least $m^* - 1$ edges, but this matching can be augmented by picking a second edge from component $\alpha$ to produce a matching of size $m^*$ in $C \setminus M^*$. Therefore, $\ell^* \geq m^* = \frac{c+2}{3}$.

Case (4). The proof is identical to the proof of case (1).

We claim that the above lemma implies that $M$ is locally 3/2-locally competitive on all non-critical components. Let $\ell$ denote the number of edges of $L$ in $C$. In case (1), $\ell = \frac{c-1}{2}$, and in cases (2) and (3), $\ell = \frac{c}{2}$. Only case (4) is not locally 3/2-competitive, since $\ell = \frac{c}{2}$.

We now show that there is a maximum matching that has no critical components with respect to a fixed optimal $M^*$; this proves the existence of a 3/2-approximate maximum matching.

Lemma 36. A maximum matching with the largest intersection with some optimal solution $M^*$ is a 3/2-approximation to the optimal interdiction solution, i.e., $|L| \leq \frac{3}{2}|L^*|$. 

75
Proof. Let $M$ be a maximum matching with the largest intersection with $M^*$ and let $L$ and $L^*$ be arbitrary maximum matchings in the respective remaining graphs. Let $C$ be a critical component in $M \cup L$. Since $|C|$ is even, one of its end edges must be in $L$. Call this edge $e$, and let $f$ denote its adjacent edge in $C$ (note that $f$ is in $M$). Since $C$ is critical, we have $e \in M^*$. Then $(M\{f\}) \cup \{e\}$ is also a maximum matching. Since $e \in M^*$ and $f \notin M^*$, this contradicts the fact that we chose $M$ as the maximum matching that maximizes $|M \cap M^*|$. 

4.3.2 A 3/2-Approximation algorithm

In this section, we make the results of the previous section constructive. If we knew $M^*$, we could give an algorithm that performed swaps of the kind used in the proof of Lemma 36. These swaps would each increase the size of $M \cap M^*$, and we would eventually obtain a solution with no critical components. Unfortunately, we don’t know $M^*$. We show, however, that sometimes we can perform sets of swaps such that the overlap of $M$ with every optimal solution $M^*$ is increased. If such a set of swaps does not exist, we argue that our solution is already a 3/2-approximation.

The formal algorithm is given in Algorithm 3. We outline the steps here. We start with an arbitrary maximum matching $M$, and a maximum matching in $G \setminus M$. We then repeatedly perform swaps of the form given above on the set of all bad components for a total of $|E| + 1$ iterations. Finally, we output the best matching found over all these iterations. We argue that while a 3/2-approximate solution has not been obtained, each iteration of swaps increases the overlap of $M$ with every optimal solution. Such an increase cannot happen more than $|E|$ times, and therefore a 3/2-approximate solution is found in some iteration of the algorithm.

Lemma 37. Let $M$ be a maximum matching and $L$ be a maximum matching in $G \setminus M$. Suppose there exists an optimal interdiction solution $M^*$ such that $M^*$ is critical on at most half the bad paths in $M \cup L$. Then, $|L| \leq \frac{3}{2}|L^*|$. 

76
Before proving Lemma 37, we show that this implies correctness of the algorithm. Suppose that for some iteration of the algorithm, the condition from Lemma 37 does not hold, i.e., every optimal solution $M^*$ is critical on strictly more than half the bad paths in $M \cup L$. After the for loop beginning on line 10, the size of the intersection between $M$ and every optimal solution will have increased. This is because for every $M^*$, every $e \to f$ swap on a critical path increases the size of the overlap between $M^*$ and $M$ by 1, while every $e \to f$ swap on a non-critical bad path decreases the overlap by at most 1. This increase in overlap can happen at most $|E|$ times, so after $|E| + 1$ iterations, we must have produced a solution $M$ with $|L| \leq \frac{3}{2}|L^*|$ as desired.

We now prove Lemma 37 using Lemma 35. Although critical components have a local approximation ratio slightly worse than $3/2$, non-critical bad paths offset this with a ratio better than $3/2$.

**Proof of Lemma 37.** Let $C_1, C_2, C_3, C_4$ denote the sets of components of type (1), (2), (3), and (4) respectively from Lemma 35. Let $\ell^*_C$ denote a maximum matching on component $C \setminus M^*$. Also, let $E(C)$ denote the edges of component $C$ and $E(C_i) =$
$\bigcup_{C \in \mathcal{C}_i} E(C)$. Then,

$$|L^*| \geq \sum_{C \in \mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{C}_3 \cup \mathcal{C}_4} \ell^*_C$$

$$\geq \sum_{C \in \mathcal{C}_1} \frac{|E(C)|}{3} + \sum_{C \in \mathcal{C}_2} \frac{|E(C)|}{3} + \sum_{C \in \mathcal{C}_3} \frac{|E(C)|}{3} + \sum_{C \in \mathcal{C}_4} \frac{|E(C)| - 1}{3}$$

(from Lemma 35)

$$= \frac{|E(C_1)| - |C_1|}{3} + \frac{|E(C_2)|}{3} + \frac{|E(C_3)| + 2|C_3|}{3} + \frac{|E(C_4)| - |C_4|}{3}$$

(since $|C_3| \geq |C_4|$, i.e., at most half of all bad paths are critical)

$$\geq \frac{|E(C_1)| - |C_1|}{3} + \frac{|E(C_2)|}{3} + \frac{|E(C_3)| + |C_3|}{3} + \frac{|E(C_4)|}{3}$$

$$(\text{since } |L \cap C| = |C|/2 \text{ for } C \in \mathcal{C}_2 \cup \mathcal{C}_3 \cup \mathcal{C}_4 \text{ and})$$

$$|L \cap C| = (|C| - 1)/2 \text{ for } C \in \mathcal{C}_1)$$

\[\Box\]

**Algorithm 3** A 3/2-approximation algorithm for the smi problem.

1. $M \leftarrow$ arbitrary maximum matching in $G$
2. $L \leftarrow$ arbitrary maximum matching in $G \setminus M$
3. $l_{\text{min}} \leftarrow |L|$
4. $M_{\text{min}} \leftarrow M$
5. for $j = 1 \rightarrow |E| + 1$ do
6. \hspace{1em} if $|L| < l_{\text{min}}$ then
7. \hspace{2em} $l_{\text{min}} \leftarrow |L|$
8. \hspace{2em} $M_{\text{min}} \leftarrow M$
9. \hspace{1em} end if
10. \hspace{1em} for bad path $C$ in $M \cup L$ do
11. \hspace{2em} $M \leftarrow M \setminus \{e\} \cup \{f\}$ \hspace{1em} $\Rightarrow$ Let $e$ be the edge at the end of $C$ that is in $L$, $f \in M$ is the adjacent edge in $C$.
12. \hspace{1em} end for
13. \hspace{1em} $L \leftarrow$ arbitrary maximum matching in $G \setminus M$.
14. end for
15. return $M_{\text{min}}$
4.4 Symmetric Matching Interdiction: Hardness of Approximation

In this section, we show that the symmetric matching interdiction problem is APX-hard which rules out the possibility of a PTAS for the problem. We give an approximation-preserving reduction from a variant of MAX-SAT called 3-OCC-MAX-2-SAT that we define below.

**Definition 4.** Let \( \phi \) be a set of clauses, where each clause is a conjunction of at most 2 literals. Additionally, each variable appears in at most 3 literals in \( \phi \). Let \( k \) be an integer. \((\phi, k)\) is said to be in 3-OCC-MAX-2-SAT if there is a setting of the variables such that at least \( k \) clauses are satisfied.

3-OCC-MAX-2-SAT is known to be APX-hard [20]. To show the hardness of the smi problem, we give an approximation preserving reduction from 3-OCC-MAX-2-SAT to the smi problem. For the purposes of the reduction, we construct an instance graph \( G \) of the smi problem from an instance of the 3-OCC-MAX-2-SAT problem \((\phi, k)\) as follows. For each variable \( x_i \), we have a cycle in \( G \) containing \( 6z_i \) edges, where \( z_i \leq 3 \) is the number of times \( x_i \) appears as a literal in \( \phi \). We partition each cycle into \( z_i \) paths of length 6 each, which we call literal paths, such that each path is associated with one of the literals containing \( x_i \). We order the edges of each path, denoting the first edge with ‘*’ so that we can refer to the first, second, etc. edge on a literal path without ambiguity. The construction up until now is illustrated in Figure 4.1.

We call all edges in such cycles cycle edges. Next, we add one edge to \( G \) for each clause in \( \phi \) (we call these clause edges). Each clause contains either one or two literals. For a clause containing two literals, the clause edge connects the two literal paths corresponding to those literals. For a clause containing one literal, the clause edge connects that literal’s path to a new vertex. Clause edges are adjacent to the second vertex on the literal path corresponding to a positive literal, and the third
vertex on the literal path corresponding to a negative literal. Figure 4.2 gives an example of the addition of clause edges.

This completes the construction of $G$. The following is our main technical lemma of the reduction.

**Lemma 38.** There is a setting of the variables that satisfies at least $k$ clauses in $\phi$ if and only if there is a matching $M$ such that in $G \setminus M$, the size of the maximum matching is at most $2\ell + m - k$, where $m$ is the number of clauses in $\phi$ and $\ell$ is the number of literals.

Before proving this lemma, we show that it is sufficient to prove APX-hardness of the SMI problem.
Theorem 39. Symmetric matching interdiction is APX-hard.

Proof. Suppose we have an \((1+\epsilon)\)-approximation to the SMI problem, i.e., a matching \(M\) in \(G\) such the maximum matching in \(G\backslash M\) has size at most

\[
(1 + \epsilon) \cdot (2\ell + m - k) = 2\ell + m - \left[1 - \epsilon \left(\frac{2\ell + m}{k} - 1\right)\right] k.
\]

By Lemma 38, we can find a formula \(\phi\) and an assignment \(x\) in the 3-OCC-MAX-2-SAT instance such that \(x\) satisfies at least \(1 - \epsilon \left(\frac{2\ell + m}{k} - 1\right)\) \(k\) clauses of \(\phi\). Note that \(\ell \leq 2m\) since each clause contains at most two literals; therefore, \(2\ell + m \leq 5m\). If each variable is set i.i.d. to T/F with equal probability, then each clause is satisfied with probability \(1/2\) if it contains a single literal, and with probability \(3/4\) if it contains 2 literals. Therefore, the expected number of clauses satisfied by a 2-SAT formula under this random assignment is at least \(m/2\). By the probabilistic method, it follows that the maximum number of satisfiable clauses \(k \geq m/2\). Therefore, \(m/k \leq 2\), which implies

\[
1 - \epsilon\left(\frac{2\ell + m}{k} - 1\right) \geq 1 - \epsilon\left(\frac{5m}{k} - 1\right) \geq 1 - 9\epsilon.
\]

Therefore, this gives a \((1 - 9\epsilon)\)-approximate solution to 3-OCC-MAX-2-SAT.

We spend the rest of the section proving Lemma 38. We first give a high level overview of the proof, and then give the technical details. We give a mapping from a setting of variables, \(x\) in \(\phi\), to a matching \(M_x\) in \(G\). We argue that \(x\) satisfies \(k\) clauses of \(\phi\) if and only if the maximum matching in \(G\backslash M_x\) contains \(2\ell + m - k\) edges (Lemma 40). Then, we argue that for graph \(G\) produced by the reduction from a formula \(\phi\), there is a setting of variables \(x\) in \(\phi\) such that \(M_x\) is the optimal solution to the SMI problem in \(G\) (Lemmas 41, 42, 43). Together, these lemmas prove Lemma 38.
Figure 4.3: For the cycle in $G$ corresponding to variable $x_i$, we show the two possibilities for the edges in $G \setminus M_x$: $M_x$ is true on the left cycle, and false on the right. Solid edges denote the edges of $G \setminus M_x$, and dotted edges denote the edges of $M_x$.

For an assignment $x$ to the variables of $\phi$, we construct matching $M_x$ as follows: $M_x$ does not contain any clause edge. $M_x$ contains every third edge on each variable cycle. For the cycle corresponding to variable $x_i$, these edges are chosen in the following way: If $x_i$ set to true, $M_x$ contains the third and sixth edges of each literal path. We call $M_x$ true on such a path. If $x_i$ is set to false, $M_x$ contains the first and fourth edges of each literal path. We call $M_x$ false on such a path. Figure 4.3 illustrates how to construct graph $G \setminus M_x$.

Lemma 40. An assignment $x$ satisfies $k$ clauses if and only if the maximum matching in $G \setminus M_x$ has size $2\ell + m - k$.

Proof. For each clause, we show that a maximum matching on the cycle and clause edges corresponding to that clause after removing $M_x$ contains two edges for each literal in the clause, along with an additional edge if the clause is not satisfied by $x$. This shows that the maximum matching on $G \setminus M_x$ has size at most $2\ell + m - k$. Moreover, in each case there is a maximum matching that does not use the last edge on each literal path. Therefore, they can be combined into a single matching with $2\ell + m - k$ edges. To prove this, we enumerate over all types of clauses in Figures 4.4, and 4.5. This completes the proof.
Figure 4.4: Enumeration of all possible satisfied clauses. Edges in $M_x$ are drawn as dotted lines. It is easily verified that each instance has the desired structure.

Figure 4.5: Enumeration of all possible unsatisfied clauses. Edges in $M_x$ are drawn as dotted lines. It is easily verified that each instance has the desired structure.

Lemma 40 implies the forward direction of Lemma 38. To show that the reduction holds in the other direction, we show that given any optimal solution $M$ to the SMI problem, we can transform it to a matching $M_x$ that is also optimal and corresponds to an assignment $x$ of $\phi$. We call such matchings that correspond to assignments in $\phi$ consistent matchings. The following are necessary and sufficient conditions for a matching to be consistent:

(a) On each variable cycle, the matching is either true or false (i.e. it contains
either the third and sixth edges of each literal path, or the first and fourth edges), and

(b) the matching does not contain any clause edge.

We show that $M$ can be transformed into a consistent matching as follows.

- If property (a) is violated, iteratively identify a cycle $C$ on which $M$ violates (a) and locally replace $M$ with $M_x$, which is defined below.
- Once only property (b) is violated, remove all remaining clause edges.

We show that neither of these steps increases the size of the maximum matching in $G \setminus M$; therefore, the eventual consistent matching is also optimal for the SMI problem.

First, we consider violations of property (a). Let assignment $x$ be defined as follows: for each variable $x_i \in x$, $x_i = true$ if $x_i$ appears as at most one negative literal in $\phi$ and $x_i = false$ if $x_i$ appears as at most one positive literal in $\phi$. If $M$ is not consistent, we will show that we can iteratively replace variable cycles of matching $M$ with the corresponding variable cycles of $M_x$.

$M$ must violate property (a) on cycle $C$ in one of the following two ways:

1. $M$ does not contain every third edge of $C$.

2. $M$ contains every third edge of $C$, but is neither true nor false on $C$ (i.e. it contains the second and fifth edges of each literal path).

Let $C_{\text{clause}}$ denote the set of clause edges adjacent to $C$. We will replace $M \cap (C \cup C_{\text{clause}})$ with $M_x \cap C$ for violation (1), and $M \cap C$ with $M_x \cap C$ for violation (2). We show in Lemmas 41 and 42 respectively that both these replacements result in valid matchings, and neither increases the size of the maximum matching in $G \setminus M$. Let $\xi_G(M)$ denote the size of the maximum matching in $G \setminus M$, and $\overline{C}$ denote $G \setminus (C \cup C_{\text{clause}})$. 

84
Figure 4.6: On the left, the variable \( x_C \) corresponding to the clause \( C \) appears as three true literals. On the right, it appears as two true literals and a false literal. The maximum matching in the first graph has size \( j/3 = 6 \), and matching in the second has size \( j/3 + 1 = 7 \).

**Lemma 41.** Consider a variable cycle \( C \) such that \( M \) does not contain every third edge of \( C \). Then replacing \( M \) with \( M' = (M \cap \overline{C}) \cup (M_x \cap C) \) produces a matching, and does not increase the size of the maximum matching in \( G \setminus M \).

*Proof.* First, note that \( M' \) is a valid matching, since edges in \( M \cap \overline{C} \) share no vertices with edges in \( M_x \cap C \). To complete the proof, we will show that \( \xi_G(M') \leq \xi_G(M) \).

First, we claim that \( \xi_C(M') \leq 3j + 1 \leq \xi_C(M) \). The proof that \( \xi_C(M) \geq 3j + 1 \) is very similar to the proof of case (3) of Lemma 35 and is not repeated here. The proof that \( \xi_{C \cup C_{\text{clause}}}(M_x) \leq 3j + 1 \) is by enumeration over all possible structures of \( (C \cup C_{\text{clause}}) \cap M_x \). We show two cases in Figure 4.6. It is straightforward to verify the other cases.

The rest of the proofs follows:

\[
\xi_G(M) \geq \xi_\overline{C}(M) + \xi_C(M) \quad \text{(vertex sets of } \overline{C} \text{ and } C \text{ are disjoint)}
\]

\[
\geq \xi_\overline{C}(M) + \xi_{C \cup C_{\text{clause}}}(M_x)
\]

\[
\geq \xi_G(M').
\]

\[\square\]

**Lemma 42.** Consider a variable cycle \( C \) such that \( M \) contains every third edge of \( C \), but is neither true nor false on \( C \) (i.e. \( M \) contains the second and fifth edge of each literal path). Then, replacing \( M \) with \( M' = (M \cap (\overline{C} \cup C_{\text{clause}})) \cup (M_x \cap C) \)
produces a matching, and does not increase the size of the maximum matching in $G \setminus M$.

**Proof.** It is not immediately clear that $M'$ is a valid matching. To show that it is, it is sufficient to show that $M$ does not contain any edges of $C_{\text{clause}}$. This follows from the fact that every edge of $C_{\text{clause}}$ is adjacent to an edge of $M \cap C$ since $M$ contains the second and fifth edges of each literal path of $C$. To complete the proof, we will show that $\xi_G(M') \leq \xi_G(M)$.

First, we claim $\xi_G(M) \geq \xi_{C \cup C_{\text{clause}}}(M) + \xi_C(M)$. For this, it is enough to show that there is a matching on $C \setminus M$ of size $\xi_C(M)$ that leaves every vertex adjacent to a clause edge unmatched. The proof is by enumeration. We show one case in below, it is straightforward to show the others.

We can now complete the proof:

$$\xi_G(M) \geq \xi_{C \cup C_{\text{clause}}}(M) + \xi_C(M)$$

$$= \xi_{C \cup C_{\text{clause}}}(M) + \xi_C(M)$$

($M \setminus C$ and $M_x \setminus C$ are the same up to a rotation of cycle $C$)

$$\geq \xi_G(M').$$

So, we can always replace $M$ locally with $M_x$ in a way that does not increase the size of the maximum matching in $G \setminus M$. By iteratively performing these replace-
ments, we obtain a matching \( M \) which violates only property (b) of the consistency conditions. We now show that if matching \( M \) only violates property (b), any clause edge of \( M \) can be removed without changing the size of the maximum matching in \( G \setminus M \).

**Lemma 43.** Suppose \( M \) is either true or false on all cycle edges, but \( M \) contains one or more clause edges. Then, removing the clause edges from \( M \) does not increase the size of the maximum matching in \( G \setminus M \).

**Proof.** \( G \setminus M \) consists of a set of connected components, each of which is either a pair of cycle edges, or two pairs of cycle edges connected by a clause edge (either a path, a barbell, or a T) as shown in Figure 4.8:

![Figure 4.8: Illustration of types of connected components of \( G \setminus M \). Components are either a path, a “barbell”, or a “T”.](image)

Removing a clause edge from \( M \) transforms a pair of two-edge paths into a barbell in \( G \setminus M \), which does not increase the size of the maximum matching. \( \square \)

### 4.5 Randomized Symmetric Matching Interdiction

We now consider a randomized version of the symmetric matching interdiction problem. Rather than selecting matchings deterministically, the interdictor and the optimizer select random matchings \( M \) and \( L \) in \( G \); the goal for the optimizer is to select \( M \) so as to minimize the maximum expected size of \( L \setminus M \), the maximum taken over all choices of the random matching \( L \). Note that unlike in the standard (deterministic)
SMI model, the randomly chosen matchings \( M \) and \( L \) need not be disjoint. It is easy to see that \( L \) can be a (deterministically chosen) best response matching since the support of a randomized best response must consist only of best response matchings. Thus, formally, the randomized SMI problem is to find a probability distribution \( \mathcal{M} \) over matchings that minimizes

\[
\max_{\text{matching } L} \mathbb{E}[|L\setminus M|],
\]

where \( M \) is a random matching drawn from \( \mathcal{M} \). Any distribution (convex combination) over integral matchings, \( M \), can be viewed as a fractional matching, i.e., as a point in the matching polytope [107]. Let \( \bar{x} = \langle x_e \rangle \) denote a point in the matching polytope with \( x_e \) the probability (equivalently the fractional weight) of choosing edge \( e \). The expected size of matching \( L \) in \( G \setminus M \) is \( \sum_{e \in L} (1 - x_e) \). Minimizing Eqn. 4.5 is therefore equivalent to minimizing over the matching polytope, the maximum over all matchings \( L \), \( \sum_{e \in L} (1 - x_e) \). This gives rise to the following LP, where, \( E(S) \) denotes the set of edges with both endpoints in \( S \), and \( \delta(v) \) denotes the set of edges adjacent to \( v \).

\[
\min_{y} \quad y
\]

s.t. \( y \geq \sum_{e \in L} (1 - x_e) \quad \forall \text{ matchings } L \in G \)

\[
\sum_{e \in E(S)} x_e \leq \frac{|S| - 1}{2} \quad \forall S \subset V, \ S \text{ odd}
\]

\[
\sum_{e \in \delta(v)} x_e \leq 1 \quad \forall v \in G
\]

\[
0 \leq x_e \leq 1 \quad \forall e \in G
\]

The constraints on the \( x_e \) variables ensure that \( \bar{x} = \langle x_e \rangle \) lies in the matching polytope [107]. This LP has exponentially many constraints (the first two sets of
constraints - matching constraints and odd set constraints), so we give a separation oracle, enabling it to be solved using the ellipsoid algorithm [65]. For the matching constraints let $z$ be the value of the maximum matching in graph $G$ with edge weights of $(1 - x_e)$. If $y \geq z$, then the solution is feasible. Otherwise, the constraint corresponding to the matching with value $z$ is violated. And for the odd set constraints we use the Gomory-Hu based separation oracle given by Padberg and Rao [97].

Thus, by solving the above LP, we can obtain the point, $\bar{x}$, in the matching polytope. However, we need a representation of this point as a convex combination of (or, distribution over) integral matchings in order to determine the (polynomial-time) strategy of the interdictor. Such a representation is guaranteed by the following known lemma (e.g. see [65]). For completeness, we give a proof in Appendix A.

**Lemma 44.** Let $\mathbf{x}$ be a fractional matching. $\mathbf{x}$ can be written as the convex combination of polynomially many integral matchings, and these matchings and their weights can be found in polynomial time.

Finally, we note that there can be a gap of 2 between the optimal randomized and deterministic matchings. Consider a length 2 path. The optimal deterministic matching is either edge, and this matching has value 1 (since it leaves a matching of size one). On the other hand, the randomized matching that assigns probability $1/2$ to each edge has value $1/2$: Regardless of which edge is chosen to be the second matching, the expected size is $1/2$.

### 4.6 Other Problems: Acyclic Subgraph Interdiction

As discussed in Section 4.2, our symmetric interdiction framework can be applied to a diverse set of combinatorial optimization problems. For example, consider any downward closed set system such as acyclic forests, independent vectors in a vector space, and more generally matroids; we can ask how much the interdictor can re-
duce some measure of the residual set system (e.g., rank) by removing a subset and its elements from the family (we can pose similar questions for families of upward closed sets). We illustrate this idea with the symmetric acyclic subgraph interdiction problem. The goal is to determine an acyclic subgraph $T$ of a given graph $G$ so as to minimize the maximum-size acyclic subgraph of $G \setminus T$. Our general framework implies a 2-approximation for this interdiction problem.

**Lemma 45.** An arbitrary spanning tree on $G$ is a 2-approximation to symmetric acyclic subgraph interdiction, and this bound is tight.

The above lemma follows directly from Corollary 33. We provide a different, more direct proof of this lemma below. This proof enables us to derive an example for which the bound is tight; i.e., there exists a graph $G$ and a spanning tree of $G$ that is at least a 2-approximate solution for $G$.

**Proof of Lemma 45.** We start with an alternate proof that an arbitrary spanning tree is at most a 2-approximation. Let $T^*$ be a minimal optimal solution, and $T$ be an arbitrary spanning tree. (If $G$ is not connected, we argue on each component separately.) Note that for $S \subseteq G$, the size of the largest set of acyclic edges in $G \setminus S$ is $n - c$, where $c$ is the number of components in $G \setminus S$.

Let $c^*$ be the number of components in $G \setminus T^*$ and $c$ be the number of components in $G \setminus T$. We consider two cases.

**Case 1:** $c^* \leq n/2$. Then since $c \geq 1$, $(n - c)/(n - c^*) \leq 2$.

**Case 2:** $c^* = n/2 + k$. The $c^*$ components of $G \setminus T^*$ form a partition of $G$, where all of the edges of $T^*$ cross the partition (by minimality of $T^*$). $T$ must span the components of $G \setminus T^*$, and therefore are exactly those that isaac 2017

$$|T^* \cap T| \geq c^* - 1 = n/2 + k - 1.$$
Additionally, $|T^*| \leq n - 1$, so we have

$$\left| T^* \setminus T \right| \leq n - 1 - n/2 - k + 1 = n/2 - k.$$  

Starting from $G \setminus T^*$, adding back each edge of $T^* \setminus T$ can decrease the number of components by at most one. Therefore, the number of components of $(G \setminus T^*) \cup (T^* \setminus T) = G \setminus (T \cap T^*)$ is at least $(n/2 + k) - (n/2 - k) = 2k$. Therefore, the edges of $T$ partition $G$ into at least $2k$ components, i.e. $c \geq 2k$. Then we have

$$\frac{n - c}{n - c^*} \leq \frac{n - 2k}{n/2 - k} = 2.$$  

Next, we show that the bound is tight. Consider a graph with $n$ vertices, such that $n/2$ vertices form a complete graph and $n/2$ vertices form a line. Additionally, there is an edge between the $i$th vertex on the line, and the $i$th vertex in the complete graph (vertices in the complete graph have arbitrary order). The optimal spanning tree is the line and all connecting edges, which leaves behind $\frac{n}{2} + 1$ components. A spanning tree that does not contain any edges of the line leaves just 2 components. Figure 4.9 illustrates this construction.
5

Price of Stability of Undirected Multicast Games

5.1 Introduction

In cost sharing network design games, we are given a graph/network $G = (V, E)$ with edge costs and a set of users (agents/players) who want to send traffic from their respective source vertices to sink vertices. Every agent must choose a path along which to route traffic, and the cost of every edge is shared equally among all agents having the edge in their chosen path, i.e., using the edge to route traffic. This creates a congestion game since the players benefit from other players choosing the same resources. A Nash equilibrium (NE) is attained in this game when no agent has incentive to unilaterally deviate from her current routing path. The social cost of such a game is the sum of costs of edges being used in at least one routing path, and efficiency of the game is measured by the ratio of the social cost in an equilibrium state to that in an optimal state. (The optimal state is defined as one where the social cost is minimized, but the agents need not be in equilibrium.) The maximum value of this ratio (i.e., for the most expensive equilibrium state) is called the price of anarchy of the game, while the minimum value (i.e., for the least
expensive equilibrium state) is called its *price of stability*. It is well known that even for the most restricted settings, the price of anarchy can be $\Omega(n)$ for $n$ agents (see Figure 5.1 for a simple example). Therefore, the main question of research interest has been to bound the price of stability (PoS) of this class of congestion games.

![Figure 5.1: An example with a price of anarchy of $\Omega(n)$.](image)

Each black vertex is an agent, and the white vertex is the root (i.e. the common sink). There is a NE where every agent routes through the edge of weight $n$. Each agent has a cost of 1 in such a configuration. On the other hand, the optimal configuration has a total cost of $1 + \epsilon$ where every agent routes through the edge of cost $(1 + \epsilon)$.

Anshelevich *et al.* [7] introduced network design games and obtained a bound of $O(\log n)$ on the PoS in directed networks with arbitrary source-sink pairs. While this is tight for directed networks, they left determining tighter bounds on the PoS in undirected networks as an open question. Subsequent work has focused on the case of all agents sharing a common sink (called *multicast games*) and its restricted subclass where every vertex has an agent residing at it (called *broadcast games*). These problems are natural analogs of the Steiner tree and minimum spanning tree (MST) problems in a game-theoretic setting. For broadcast games, Fiat *et al.* [56] improved the PoS bound to $O(\log \log n)$, which was subsequently improved to $O(\log \log \log n)$ by Lee and Ligett [88], and ultimately to $O(1)$ by Bilò, Flammini, and Moscardelli [23].

For multicast games, however, progress has been much slower, and the only improvement over the $O(\log n)$ result of Anshelevich *et al.* is a bound of $O(\log n / \log \log n)$ due to Li [90]. In contrast, the best known lower bounds on the PoS of both broadcast and multicast games are small constants [22]. Determining the PoS of multicast
games has become one of the most compelling open questions in the area of network games.

In the multicast setting, a vertex is said to be a terminal if it has an agent on it, else it is called a nonterminal. Note that in the broadcast problem, there are no nonterminals and all the edges are between terminal vertices. Here, we consider multicast games in quasi-bipartite graphs: all edges are either between two terminals, or between a nonterminal and a terminal. (That is, there is no edge with both nonterminal endpoints.) This represents a natural setting of intermediate generality between broadcast and multicast games. Moreover, quasi-bipartite graphs have been widely studied for the Steiner tree problem (see, e.g., [99, 101, 33, 31]) and has provided insights for the problem on general graphs.

5.1.1 Our Results

Our main result is an $O(1)$ bound on the PoS of multicast games in quasi-bipartite graphs.

**Theorem 46.** The price of stability of multicast games in quasi-bipartite graphs is a constant.

In Table 5.1, we summarize the known and new results for single sink network design and the corresponding cost sharing games.

<table>
<thead>
<tr>
<th>Optimization (Approximation factor)</th>
<th>Cost sharing game (Price of Stability)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MST Poly-time solvable</td>
<td>Broadcast $O(1)$ [23]</td>
</tr>
<tr>
<td>Quasi-bipartite Steiner 1.22 [31]</td>
<td>Quasi-bipartite Multicast $O(1)$ [This paper]</td>
</tr>
<tr>
<td>Steiner Tree 1.39 [31]</td>
<td>Multicast $O\left(\frac{\log n}{\log \log n}\right)$ [90]</td>
</tr>
</tbody>
</table>
5.1.2 Related Work

Recall that the upper bounds for the PoS are a (large) constant and $O\left(\frac{\log n}{\log \log n}\right)$ for broadcast and multicast games, respectively. The corresponding best known lower bounds are 1.818 and 1.862 respectively by Bilò et al. [22], leaving a significant gap, even for broadcast games. Moreover, Lee and Ligett [88] show that obtaining superconstant lower bounds, even for multicast games where they might exist, is beyond current techniques. While this lends credence to the belief that the PoS of multicast games is $O(1)$, Kawase and Makino [83] have shown that the potential function approach of Anshelevich et al. [7] cannot yield a constant bound on the PoS, even for broadcast games. In fact, Bilò et al. [23] used a different approach for broadcast games, as do we for multicast games on quasi-bipartite graphs.

Various special cases of network design games have also been considered. For small instances ($n = 2, 3, 4$), both upper [38] and lower [21] bounds have been studied. [38] show upper bounds of 1.65 and 4/3 for two and three players respectively. For weighted players, Anshelevich et al. [7] showed that pure Nash equilibria exist for $n = 2$, but the possibility of a corresponding result for $n \geq 3$ was refuted by Chen and Roughgarden [35], who also provided a logarithmic upper bound on the PoS. An almost matching lower bound was later given by Albers [3]. Recently, Fanelli et al. [52], showed that the PoS of network design games on undirected rings is 3/2.

Network design games have also been studied for specific dynamics. In particular, starting with an empty graph, suppose agents arrive online and choose their best response paths. After all arrivals, agents make improving moves until an NE is reached. The worst-case inefficiency of this process was determined to be poly-logarithmic by Charikar et al. [34], who also posed the question of bounding the inefficiency if the arrivals and moves are arbitrarily interleaved. This question remains open. Upper and lower bounds for the strong PoA of undirected network design games have also
been investigated [3, 48]. They show that the price of anarchy in this setting is $\Theta(\log n)$.

5.1.3 Chapter Roadmap

The content in the chapter is technically challenging, and therefore we begin by providing a lengthy overview of our techniques in Section 5.2 In Section 5.3, we introduce notation, definitions, and claims that will be used throughout the rest of the chapter. In Section 5.4 we give an algorithm that constructs a sequence of moves that result in a NE. Finally in Section 5.5, we show that this algorithm does indeed terminate, and we analyze the cost of the NE that is reached.

5.2 Overview of Techniques

Our techniques overcome some of the fundamental difficulties of analyzing the PoS of general multicast games, and therefore represent a promising step toward resolving this important open problem. To illustrate this point, we outline the salient features of our analysis below.

The previous PoS bounds for multicast games [7, 90] are based on analyzing a potential function $\phi_e$ defined on each edge $e$ as its cost scaled by the harmonic of the number of agents using the edge, i.e., $\phi_e = \text{cost}(e) \cdot (1 + 1/2 + 1/3 + \cdots + 1/j)$ where $j$ is the number of terminals using $e$. The overall potential is $\phi = \sum_e \phi_e$. When an agent changes her routing path (called a move), this potential exactly tracks the change in her shared cost. If the move is an improving one, then the shared cost of the agent decreases and so too does the potential. As a consequence, for an arbitrary sequence of improving moves starting with the optimal Steiner tree, the potential decreases in each move until a NE is reached. This immediately yields a PoS bound of $H(n) = O(\log n)$ [7]. To see this, note that the potential of any configuration is bounded below by its cost, and above by its cost times $H(n)$. Then, letting $S_{NE}$ be
the Nash equilibrium state reached, and \( T^* \) be the optimal routing tree, we have

\[
c(S_{NE}) \leq \phi(S_{NE}) \leq \phi(T^*) \leq H(n)c(T^*).
\]

This bound was later improved to \( O(\log n / \log \log n) \) by Li [90] with a similar but more careful accounting argument.

The previous PoS bounds for broadcast games [56, 88, 23] use a different strategy. As in the case of multicast games, these results analyze a game dynamics that starts with an optimal solution (MST) and ends in an NE. However, the sequence of moves is carefully constructed — the moves are not arbitrary improving moves. At a high level, the sequence follows the same pattern in all the previous results for broadcast games:

1. **Perform a critical move:** Allow some terminal \( v \) to switch its path to introduce a single new edge into the solution, that is not in the optimal routing tree and is adjacent to \( v \). This edge is associated with \( v \) and denoted \( e_v \). Any edge introduced by the algorithm in any move other than a critical move uses only edges in the current routing tree, and edges in the optimal routing tree. Therefore, we only need to account for edges added by critical moves.

2. **Perform a sequence of moves to ensure that the routing tree is homogenous.** That is, the difference in costs of a pair of terminals is bounded by a function of the length of the path between them on the optimal routing tree. For example, suppose two terminals \( w \) and \( w' \) differ in cost by more than the length of the path between them in the optimal routing tree. Then the terminal with larger cost has an improving move that uses this path, and then the other terminal’s path to the root. Such a move introduces only edges in the optimal routing tree.

3. **Absorb** a set of terminals around \( v \) in the shortest path metric defined on the
optimal tree: terminals \( w \) replace their current strategy with the path in the optimal routing tree to \( v \), and then \( v \)'s path to the root. If \( w \) had an associated edge \( e_w \), introduced via a previous critical move, it is removed from the solution in this step.

The absorbing step allows us to account for the cost of edges added via critical moves, by arguing that vertices associated with critical edges of similar length must be well-separated on the optimal routing tree. If edges \( e_u \) and \( e_v \) are not far apart, the second edge to be added would be removed from the solution via the absorbing step.

Homogeneity facilitates absorption: Suppose \( v \) has performed a critical move adding edge \( e_v \), and let \( w \) be some other terminal. While \( v \) pays \( c(e_v) \) to use edge \( e_v \), \( w \) would only pay \( c(e_v)/2 \) to use \( e_v \), since it would split the cost with \( v \). That is, if \( w \) bought a path to \( v \) and then used \( v \)'s path to the root, it would save at least \( c(e_v)/2 \) over \( v \)'s current cost. If the current costs paid by \( v \) and \( w \) are not too different, and the distance between \( v \) and \( w \) not too large, then such a move is improving for \( w \).

The previous results differ in how well they can homogenize: the tighter the bound on the difference in costs of a pair of terminals as a function of the length of the path between them in the optimal routing tree, the larger the radius in the absorb step. In turn, a larger radius of absorption establishes a larger separation between edges with similar cost, which yields a smaller (tighter) bound on the PoS.

This homogenization-absorption framework has not previously been extended to multicast games. The main difficulty is that there can be nonterminals that are in the routing tree at equilibrium but are not in the optimal tree. No edge incident on these vertices is in the optimal tree metric, and therefore these vertices cannot be included in the homogenization process. So, any critical edge incident on such a vertex cannot be charged via absorption. This creates the following basic problem: what metric can
we use for the homogenization-absorption framework that will satisfy the following two properties?

1. The metric is feasible – the sum of all edge costs in (a spanning tree of) the metric is bounded by the cost of the optimal routing tree. These edges can therefore be added or removed at will, without need to perform another set of moves to pay for them (in contrast to critical edges). This allows us to homogenize using these edges.

2. The metric either includes all vertices (as is the case with the optimal tree metric for broadcast games), or if there are vertices not included in the metric, critical edges adjacent to these vertices can be accounted for separately, outside the homogenization-absorption framework.

We create such a metric for quasi-bipartite graphs, allowing us to extend the homogenization-absorption framework to multicast games. Our metric is based on a dynamic tree containing all the terminals and a dynamic set of nonterminals. We show that under certain conditions, we can include the shortest edge incident on a nonterminal vertex, even if it is not in the optimal routing tree, in this dynamic tree. These edges are added and removed throughout the course of the algorithm. Our new metric is now defined by shortest path distances on this dynamic tree: the optimal routing tree extended with these special edges. We ensure homogeneity not on the optimal routing tree, but on this dynamic metric. Likewise, absorption happens on this new metric. We define the metric in such a way that the following hold:

1. The metric is feasible. That is, the total cost of all edges in the dynamic tree is within a constant factor of the cost of the optimal tree.

2. Consider some critical edge $e_v$ such that the corresponding vertex $v$ is not in the metric. That is, it was not possible to add the shortest edge adjacent to $v$ to the
dynamic tree while keeping it feasible. Therefore, \( v \) is at infinite distance from every other vertex in this metric, ruling out homogenization. Then, \( e_v \) can be accounted for separately, outside the homogenization-absorption framework.

For the remaining edges \( e_v \) such that \( v \) is in the metric, we account for them by using the homogenization-absorption framework. Our main technical contribution is in creating this feasible dynamic metric, going beyond the use of static optimal metrics in broadcast games. While the proof of feasibility currently relies on the quasi-bipartiteness of the underlying graph, we believe that this new idea of a feasible dynamic metric is a promising ingredient for multicast games in general graphs.

5.3 Preliminaries

Let \( G = (V, E) \) be an undirected edge-weighted graph and let \( c(e) \) denote the cost of edge \( e \). Let \( U \subseteq V \) be a set of terminals and \( r \in U \). In an instance of a network design game, each terminal \( u \) is associated with a player, or agent, that must select a path from \( u \) to \( r \). We consider instances in which \( G \) is quasi-bipartite, that is no edge \( e \) has two nonterminal end points.

A solution, or state, is a set of paths connecting each player to the root. Let \( \mathcal{S} \) be the set of all possible solutions. For a solution \( S \), a terminal \( u \), and some subset \( E' \) of the edges in the graph, let \( c_u'(S) = \sum_{e \in E'} c(e)/n_e(S) \) be the cost paid by \( u \) for using edges in \( E' \), where \( n_e(S) \) is the number of players using edge \( e \) in state \( S \). Let \( p_u(S) \) be the set of edges used by \( u \) to connect to the root in \( S \) and let \( c_u(S) = c_{p_u(S)}(S) \) be the total cost paid by \( u \) to use those edges. For a nonterminal \( v \), if every terminal \( u \) with \( v \in p_u(S) \) uses the same path from \( v \) to the root then define \( p_v(S) \) to be this path from \( v \) to \( r \), and \( c_v(S) = c_{p_v(S)}(S) \). Additionally, we will sometimes refer to the cost a vertex \( v \) pays, even if \( v \) is a nonterminal. By this we mean \( c_v(S) \). For any vertex \( v \in S \), let \( e_v \) be the edge in \( p_v(S) \) with \( v \) as an endpoint.
Let $\Phi : \mathcal{S} \to \mathbb{R}_+$ be the potential function introduced by Rosenthal [103], defined by

$$
\Phi(S) = \sum_{e \in E} c(e) H_{n_e(S)} = c(e) \left( 1 + \frac{1}{2} + \cdots + \frac{1}{n_e(S)} \right).
$$

Let $u \in U$ and suppose $S$ and $S'$ are states for which $p_v(S) = p_v(S')$ for all players $v \neq u$. Then $\Phi(S') - \Phi(S) = c_u(S') - c_u(S)$. In particular, if a single player changes their path to a path of lower cost, the potential decreases.

The goal of each player is to find a path of minimum cost. A solution where no player can benefit by unilaterally changing their path is called a Nash Equilibrium. Let $T^*$ be a solution that minimizes the total cost paid. Note that $T^*$ is a minimum Steiner tree for $G$. The price of stability (PoS) is the ratio between the minimum cost of a Nash equilibrium and the cost of $T^*$.

Let $p_{T^*}(u, v)$ be the path in $T^*$ between $u$ and $v$. Let $v_1, \ldots, v_n$ be the vertices of $T^*$ in the order they appear in a depth first search of $T^*$. Let $MC$, the “main cycle”, be the concatenation of $p_{T^*}(v_1, v_2), p_{T^*}(v_2, v_3), \ldots, p_{T^*}(v_{n-1}, v_n), p_{T^*}(v_n, v_1)$.

Note that each edge in $T^*$ appears exactly twice in $MC$. The following property will be helpful:

**Fact 47.** Any $x$ to $y$ path in $MC$ completely contains $p_{T^*}(x, y)$.

Define the class of edge $e$, $\text{class}(e)$, as $\alpha$ if $256^\alpha \leq c(e) < 256^{\alpha+1}$. Without loss of generality, we assume that $c(e) \geq 1$ for all $e \in E$, so the minimum possible edge class is 0. For simplicity, define $[c(e)] = 256^{\text{class}(e)}$, a lower bound for $c(e)$, and $[c(e)] = 256^{\text{class}(e)+1}$, an upper bound for $c(e)$.

For each nonterminal $v$, let $\sigma_v$ be the minimum cost edge adjacent to $v$ in $G$. Let $t_v$ be the terminal adjacent to $\sigma_v$. Let $T^+$ be the extended optimal metric: $T^* \cup \{\sigma_v\}_{v \in V}$. We maintain a dynamic set of nonterminals

$$
\mathcal{Z}_S = \{w \notin T^* : c(\sigma_w) \leq [c(e_w)]/64\}.
$$
That is, \( Z_S \) are those nonterminals \( w \) in solution \( S \) whose first edge \( e_w \) has cost within a constant factor of the cost of \( \sigma_w \). For any \( w \in S \), if \( \sigma_w \) is added to \( S \) while \( w \in Z_S \), then we will show that we will be able to pay for \( \sigma_w \) if it remains in the final solution. We prove this fact in Section 5.5.2. In the description of the algorithm, we denote the current state by \( S_{\text{curr}} \). For ease of notation, we define \( Z = Z_{S_{\text{curr}}} \).

The remaining definitions are modifications of key definitions from [23]. The interval around vertex \( v \in T^* \) with budget \( y \), \( I_{v,y} \), is the concatenation of its right and left intervals, \( I_{v,y}^+ \) and \( I_{v,y}^- \), where \( I_{v,y}^+ \) is the maximal contiguous interval in \( MC \) with \( v \) a left endpoint such that

\[
2 \sum_{\alpha \geq 0} 256^{\alpha+1} H_{n_{I v,y}^+, \alpha}^2 \leq y,
\]

where \( n_{I v,y}^+, \alpha \) is the number of edges of class \( \alpha \) in \( I_{v,y}^+ \) (repeated edges are counted every time they appear). We define \( I_{v,y}^- \) similarly.

The *neighborhood of \( v \) in state \( S \), \( N_S(v) \)* is an interval around \( v \) as well as certain \( w \notin T^* \) with \( t_w \) in the interval. Formally,

\[
N_S(v) = \begin{cases} 
I_{v, [c(e_v) \frac{56}{64}]} \cup \left\{ w \in Z_S \left| t_w \in I_{v, [c(e_v) \frac{56}{64}]} \right. \text{ and } c(\sigma_w) \leq \frac{|c(e_v)|}{64} \right\} & \text{if } v \in T^*, \\
I_{v, [c(e_v) \frac{56}{64}]} \cup \left\{ w \in Z_S \left| t_w \in I_{v, [c(e_v) \frac{56}{64}]} \right. \text{ and } c(\sigma_w) \leq \frac{|c(e_v)|}{64} \right\} & \text{otherwise.}
\end{cases}
\]

\( N_S^+(v) \) and \( N_S^-(v) \) are the right and left intervals of the neighborhood respectively (that is, the portions of \( N_S(v) \) to the right and left of \( v \) or \( t_v \) respectively). We denote \( N_{S_{\text{curr}}}(v) \) as \( N(v) \). Roughly speaking, we are going to charge the cost of edges in the final solution not in \( T^* \) to the interval portions of non-overlapping right neighborhoods.

Observe that every edge in \( N(v) \cap T^* \) has class at most \( \text{class}(e_v) - 2 \). If this were false,

\[
2 \sum_{\alpha \geq 0} 256^{\alpha+1} H_{n_{I v,y}^+, \alpha}^2 \geq 256^{|\text{class}(e_v)|} \frac{|c(e_v)|}{56},
\]

102
which contradicts the definition of \( N(v) \). A path \( X = p_{T^*}(x, y) \) is homogenous if

\[
|c_x(S) - c_y(S)| \leq 4 \sum_{\alpha \geq 0} 256^{\alpha+1} H_{nX,\alpha}^2.
\]

If \( X = p_{T^*}(x, y) \subseteq N(v) \cap T^* \) is a homogenous path then

\[
|c_x(S) - c_y(S)| \leq 4 \sum_{\alpha \geq 0} 256^{\alpha+1} H_{nX,\alpha}^2 \leq 8 \sum_{\alpha \geq 0} 256^{\alpha+1} H_{nN+(v),\alpha}^2 \leq |c(e)|/14.
\]

\( N(v) \) is homogenous if the following holds: For all \( x, y \in N(v) \) with \( x, y \neq u_v \), a special vertex to be defined later, such that the path in \( T^+ \) from \( x \) to \( y \) does not contain \( v \), \( |c_x(S_{curr}) - c_y(S_{curr})| \leq \frac{23|c(e)|}{112} \). Homogenous neighborhoods allow us to bound the difference in cost between any two vertices in \( N(v) \) which will be useful when arguing that players have improving strategy changes.

5.4 Algorithm

The initial state of the algorithm is the minimum cost tree \( T^* \) connecting all the terminals to the root. The algorithm carefully schedules a series of potential-reducing moves. (Recall the potential function \( \Phi(S) = \sum_{e \in E} c(e) H_{n\varepsilon(S)} \) introduced in Section 5.3). Since there are finitely many states possible, such a series of moves must always be finite. Since any improving move reduces potential, we must be at a Nash equilibrium if there is no potential reducing move. These moves are scheduled such that if any edge outside of \( T^* \) is introduced, it is subsequently accounted for by charging to some part of \( T^* \). In particular, we will show that at any point in the process, and therefore in the equilibrium state at the end, the total cost of these edges is bounded by \( O(1) \cdot c(T^*) \).

The algorithm is a series of loops, which we run repeatedly until we reach a Nash equilibrium. Each loop begins with a terminal, \( a \), performing either a safe improving move, or a critical improving move. In both cases, \( a \) switches strategy to
Figure 5.2: Types of critical improving moves. Dotted edges represent the new edges being added.

follow a new path to the root. Let $S$ be the state before the start of the loop. A safe improving move is one which results in some state $S' \subseteq T^* \cup S$, i.e., the new path of $a$ contains edges currently in $S$ and edges in the optimal tree $T^*$. A safe improving move requires no additional accounting on our part. A critical improving move on the other hand introduces one or two new edges that must be accounted for (see Figure 5.2). We will show later that in any non-equilibrium state, a safe or critical improving move always exists (see Lemma 56).

The algorithm will use a sequence of (potential-reducing) moves to account for the new edges introduced by a critical move. At a high level, each of these edges is accounted for in the following way. Let $e_v$ be the edge in question, and $v$ be the first vertex using $e_v$ on its path to the root.

1. In some neighborhood around $v$, perform a sequence of moves to ensure that for every pair of vertices (excluding $v$ and at most one other special vertex), the difference in shared costs of these vertices is not too large. (Recall that the while nonterminals do not pay anything, the shared cost of a nonterminal $u$ is defined to be $c_u(S)$, the cost that a terminal using $u$ pays on its subpath from $u$ to the root). This sequence of moves must be potential-reducing, and cannot add any edges outside of $T^* \cup S$ to the solution.

2. For every vertex $y$ in the neighborhood around $v$, $v$ has an alternative path to the root consisting of the path in $T^+$ to $y$, and $y$’s path to the root. (Recall from Section 5.3 that $T^+$ is the optimal tree, $T^*$, augmented with minimum
cost edges incident on nonterminals \( \{\sigma_w : w \text{ is a nonterminal}\} \).)

(a) If there is a \( y \) for which this alternative path is an improving path for \( v \), then \( v \) can switch to this new path and \( e_v \) will be removed from the solution.

(b) If every path is \textit{not} improving for \( v \), then we show that every vertex in the neighborhood of \( v \) has an improving move that uses \( e_v \).

These steps ensure that we either remove \( e_v \) from the solution, or else for any vertex \( y \) in the neighborhood we remove edge \( e_y \notin T^* \) from the solution. We elaborate on the steps above, referencing the subroutines described in Algorithm 5 – \textsc{Homogenize}, \textsc{Absorb}, and \textsc{MakeTree}:

\textit{Step 1:} This is accomplished in two ways. For any path in \( T^* \), the \textsc{Homogenize} subroutine ensures that a path in \( T^* \) is homogenous. Recall that this gives a bound (relative to the cost of \( e_v \)) on the difference in shared costs of the endpoints of the path. Additionally, for any pair of adjacent vertices, if the difference in the shared costs is more than the cost of the edge between them, then one vertex must have an improving move through this edge. This move adds no edges outside of \( T^* \). The second way of bounding differences in shared cost is much weaker, but we will use it only a small number of times. Overall, the path between any two vertices in the neighborhood will comprise homogenous segments connected by edges whose cost is bounded by the second method above. Adding up the cost bounds for each of these segments gives us the total bound. Lemma 50 gives the technical details.

\textit{Step 2(a):} The purpose of this step is to establish that either the shared cost of \( v \) is not much larger than the shared cost of every other vertex in its neighborhood, or that we can otherwise remove \( e_v \) from the solution. If the shared cost of \( v \) is much larger than some other vertex in the neighborhood, then it is also much larger than the shared cost of an adjacent vertex (call it \( q \)) in \( T^+ \). This is because every pair of
vertices in the neighborhood have a similar shared cost (by Step 1). Then, \( v \) has a lower cost path to the root consisting of the \((v, q)\) edge, combined with \( q \)'s current path to the root. Such a move would remove \( e_v \) from the solution.

**Step 2(b):** If we reach this step, we need to account for the cost of \( e_v \) by making every other vertex in the neighborhood give up its first edge, if that edge is not in \( T^+ \). This ensures that at the end, the edges in the solution that are not in \( T^+ \) will be very far apart. This is accomplished via the Absorb function: \( v \) is currently paying the entire cost of \( e_v \), while any vertex that would switch to using \( v \)’s path to the root would only pay at most half the cost of \( e_v \). Furthermore, if vertices close to \( v \) in \( T^+ \) switch first, vertices farther from \( v \) (who must pay a higher cost to buy a path to \( v \)) will reap the benefits of more sharing, and therefore a further reduction in shared cost. This is formalized in the definition of Absorb.

There are some other details which we mention here before moving on to a more formal description of the algorithm:

- **If** \( v \) **is** a nonterminal, **let** \( u_v \) **be** the terminal that added \( v \) **as part** the critical move. We avoid including \( u_v \) in any path provided to the HOMOGENIZE subroutine. This is because HOMOGENIZE switches the strategies of terminals to follow the strategy of some terminal on input path. If terminals were switched to follow \( u_v \)'s path, this would increase the sharing on \( e_v \), when it is required at the beginning of Step (2b) that only one terminal is using \( e_v \). When \( v \) is a terminal, then \( u_v \) is undefined and this problem does not exist. We define two versions of a loop of the algorithm, defined as MAINLOOP in Algorithm 4, to account for this difference.

- **We** have only described **how** to account for a single edge, but sometimes a critical move adds two new edges that must be accounted for. **Suppose** \( e_a \) **and** \( e_b \) **are** the new edges added by \( a \) (\( a \) **is** a terminal and \( b \) **is** a nonterminal). Then
we run MAINLOOP($e_b$) first, and then MAINLOOP($e_a$). The first loop does not increase sharing on $e_a$, so the second loop is still valid.

- We assume the existence of a function MAKE TREE. This function takes as input a set of strategies. Its output is a new set of strategies such that (1) the new set of strategies has lower potential than the old set, (2) the edge set of the new strategies is a subset of the old edge set, and (3) the edge set of the new strategies is a tree. In particular, MAKE TREE($S_{curr}\setminus \{p_{uv}(S_{curr}), p_v(S_{curr})\}$), used on line 12 does not increase sharing on $e_v$, since $v$ and $u_v$ are the only two vertices using $e_v$ on their path to the root. MAKE TREE($S_{curr}\setminus \{p_{uv}(S_{curr}), p_v(S_{curr})\}$) will also not increase sharing on $e_u$, if this edge has just been added (and therefore $u_v$ is the only vertex using the edge). We will not go into more detail about this function, since an identical function was used in both [23] and [56].

- We assume that all edges in $E$ with $c(e) > c(T^*)$ have been removed from the graph. This is without loss of generality: if the final state $S_f$ is a Nash equilibrium, then $S_f$ is still an equilibrium after reintroducing $e$ with $c(e) > c(T^*)$. This is because any vertex with an improving move that adds such an edge $e$ also has a path to the root (the path in $T^*$) with total cost less than $c(e)$.

We walk through the pseudocode next: We execute the MAINLOOP function given in Algorithm 4 either once or twice, once for each edge not in $T^* \cup S$ that is added by a critical move. If two edges have been added, we execute in the order MAINLOOP($e_b$) then MAINLOOP($e_a$) (where $a$ is the terminal and $b$ is the nonterminal). We define two versions of MAINLOOP($e_v$), one when $v$ is a terminal, and one when $v$ is a nonterminal, appearing on lines 25 and 1 respectively. When $v$ is a nonterminal, we denote the terminal which added $e_v$ to the solution as part of the initial improving
move as $u_v$. For brevity, we define $u_v$ as “empty” when $v$ is a terminal. Thus if $v$ is a terminal, define $N(v) \setminus \{u_v\} = N(v)$.

**Algorithm 4** Main loop to be executed for each edge added to the solution as part of a critical move.

1: function MAINLOOP($e_v$) ⇐ $v$ is a nonterminal and $u_v$ the terminal which added $e_v$ as part of a critical move.
2: while any of the following if conditions are true do
3:    if $\exists X = p_{T^*}(x, y) \in N(v) \cap T^*$ with $u_v, v \notin X$ and $X$ not homogenous
4:        then HOMOGENIZE($X$)
5:        if $\exists x, y \in N(v) \setminus \{v\}$ adjacent to $u_v$ with $c_x(S_{curr}) - c_y(S_{curr}) > c(x,u_v) + c(u_v, y)$ then
6:            Replace $x$’s strategy with $(x, u_v) \cup (u_v, y) \cup p_y(S_{curr}).$
7:        end if
8:        if $\exists w \in N(v) \setminus T^*$ such that $t_w \neq v, u_v$ with $|c_w(S_{curr}) - c_{t_w}(S_{curr})| > c(\sigma_w)$
9:            Assuming WLOG $c_{t_w}(S_{curr}) > c_w(S_{curr})$, replace $t_w$’s strategy with $\sigma_w \cup p_w(S_{curr}).$
10:       end if
11:     if $S_{curr} \setminus \{p_{u_v}(S_{curr}), p_v(S_{curr})\}$ is not a tree then
12:         MAKETREE($S_{curr} \setminus \{p_{u_v}(S_{curr}), p_v(S_{curr})\}$)
13:     end if
14: end while
15: for $q \in N(v) \setminus \{v, u_v\}$ adjacent in $T^+$ to either $v$ or $u_v$ do
16:     if $c(v, q) + c_q(S_{curr}) < c_v(S_{curr})$ then
17:        $v$ changes strategy to $(v, q) \cup p_q(S_{curr}).$
18:     return
19: end if
20: Repeat the previous 3 lines substituting $u_v$ for $v.$
21: ⇐ Note that $u_v$ changing strategy will remove $v$ from the solution.
22: end for
23: ABSORB($v$)
24: end function ⇐ (Continued on next page.)
25: function MainLoop($e_v$)  
26:     while any of the following if conditions are true do
27:         if $\exists X = p_{T^*}(x, y) \in N(v) \cap T^*$ with $v \notin X$ and $X$ is not homogenous then Homogenize($X$)  
28:             end if  
29:         if $\exists w \in N(v) \setminus T^*$ such that $t_w \neq v$ with $|c_w(S_{curr}) - c_{t_w}(S_{curr})| > c(\sigma_w)$ then  
30:             Assuming WLOG $c_{t_w}(S_{curr}) > c_w(S_{curr})$, replace $t_w$'s strategy with $\sigma_w \cup p_w(S_{curr})$.  
31:         end if  
32:         if $S_{curr} \setminus \{p_v(S_{curr})\}$ is not a tree then MakeTree($S_{curr} \setminus \{p_v(S_{curr})\}$)  
33:             end if  
34:     end while  
35:     for $q \in N(v)$ adjacent in $T^+$ to $v$ do  
36:         if $c(v, q) + c_q(S_{curr}) < c_v(S_{curr})$ then  
37:             $v$ changes strategy to $(v, q) \cup p_q(S_{curr})$.  
38:             return  
39:         end if  
40:     end for  
41:     Absorb($v$)  
42: end function

The while loops at lines 2 and 26 terminate with $N(v)$ being homogenous. For any violated if statement within the while loop, we perform a move that reduces potential, and does not increase sharing on $e_v$, or on $e_u$ if it was added along with $e_v$ as part of $u_v$’s critical move. In Lemma 50 we show that if none of these if conditions hold, $N(v)$ is homogenous. Therefore, this while loop eventually terminates in a homogenous state.

We next use the cost bound given by Lemma 50 to ensure that the cost that $v$ pays is similar to the cost every other vertex in $N(v)$ pays. If these costs are not close, we show in Lemma 51 that the condition at line 16/36 will be true, and $e_v$ will be deleted from the solution.

If $e_v$ is still present at this point, we finally call the Absorb function. Lemma 51 ensures that the precondition of the Absorb function is met. We use this condition to show that the switches made by all the vertices in $N(v)$ in the Absorb function
Algorithm 5 Helper functions for Algorithm 4.

25: function Homogenize($X = p_{T^*}(x, y)$)
26: Let $X = (x = x_1, x_2, \ldots, x_k, x_{k+1} = y)$
27: Let $S'$ be the current state.
28: for $i \leftarrow 1$ to $k$ do
29: for $j \leftarrow i$ down to 1 do
30: Change $x_j$’s strategy to $p_{T^*}(x_j, x_{i+1}) \cup p_i(S)$.
31: end for
32: if $\Phi(S_{curr}) < \Phi(S')$ then return
33: else Reset state to $S'$
34: end if
35: end for
36: end function

Require: $c_q(S) \geq c_v(S) - \frac{2|c(e_v)|}{q}$ \quad $\forall q \in N(v) \setminus \{u_v\}$ \quad $\Rightarrow$ See Lemma 51

37: function Absorb($v$) \quad $\Rightarrow v$ absorbs $N(v) \setminus \{u_v\}$
38: for $q \in N(v) \cap T^* \setminus \{u_v\}$ in breadth-first order from $r$ according to $T^*$ do
39: if $v \not\in T^*$ then Change $q$’s strategy along with its descendants to $p_{T^*}(q, t_v) \cup \sigma_v \cup p_v(S)$.
40: else Change $q$’s strategy along with its descendants to $p_{T^*}(q, v) \cup p_v(S)$.
41: end if
42: end for
43: Let $S'$ be the current state.
44: for $q \in N(v) \setminus T^*$, in reverse breadth-first order from $r$ according to $S'$ do
45: Change $q$’s strategy along with its descendants to $\sigma_q \cup p_v(S')$.
46: end for
47: end function

are improving, and therefore reduce potential.

Note that although we do not make this explicit, if at any point $S_{curr}$ contains edges that are not part of $p_u(S_{curr})$ for any terminal $u$, these edges are deleted immediately. This ensures that any nonterminal in $S_{curr}$ is always used as part of some terminal’s path to $r$.

5.5 Analysis

In this section, we first prove some properties about the algorithm. Then we analyze the cost of the final Nash equilibrium.
5.5.1 Termination

We first show that all parts of the algorithm reduce potential, guaranteeing that the algorithm terminates (by the definition of the potential function, the minimum decrease in potential is bounded away from 0).

Most steps in the algorithm involve single terminals making improving moves, and therefore these steps reduce potential. There are two parts of the algorithm for which it is not immediately obvious that potential is reduced: the Homogenize function and the Absorb function. We first show that the Homogenize function reduces potential.

**Theorem 48.** Suppose there is a path $X = p_T^*(x, y) \in N(v)$ which is not homogenous. Let $(x = x_1, x_2, \ldots, x_k, x_{k+1} = y)$ be the sequence of vertices in $X$. Then there exists a prefix of $X$, $(x_1, \ldots, x_i)$, such that the sequence of moves in which each $x_j$, $j \in \{1, \ldots, i\}$, switches its strategy to $p_T^*(x_j, x_{i+1}) \cup p_{x_{i+1}}(S)$ reduces potential.

Note that the order in which the vertices move does not affect the change in potential of the entire sequence of moves. However, to help prove the theorem, we will assume that the vertices execute these moves in the order $x_i, x_{i-1}, \ldots, x_1$, the order given in Algorithm 5. Let $e_j = (x_j, x_{j+1})$. Let $S$ be the state before the prefix move starts, and let $S_j$ be the state just after $x_j$ switches its strategy. Note that $S_{j+1}$ is the state just before $x_j$ switches.

**Lemma 49.** The prefix move given in Theorem 48 for prefix $(x_1, \ldots, x_i)$ does not reduce potential only if

$$\sum_{j=1}^{i} c_{x_j}(S) - c_{x_{i+1}}(S) \leq 2 \sum_{j=1}^{i} 2H_j c(e_j).$$
Proof. The change in potential caused by \( x_j \)'s switch is \( c_{x_j}(S_j) - c_{x_j}(S_{j+1}) \). Partition the edges of \( x_j \)'s strategy in \( S \) into 3 sets: edges in \( p_T^*(x_j, x_{i+1}) \), edges in \( p_{x_{i+1}}(S) \), and all other edges, called \( E_{1,j} \), \( E_{2,j} \), and \( E_{3,j} \) respectively. Additionally, let \( E_{4,j} \) be the remaining edges in \( x_{i+1} \)'s strategy: those not in \( E_{2,j} \). For edge \( e \), let \( n_e(S) \) be the number of players using edge \( e \) in state \( S \).

- \( c_{x_j}^{E_{1,j}}(S_{j+1}) \geq 0 = c_{x_j}^{E_{1,j}}(S) - \sum_{e \in E_{1,j}} \frac{1}{n_e(S)} c(e) \), since \( c_{x_j}^{E_{1,j}}(S) = \sum_{e \in E_{1,j}} \frac{1}{n_e(S)} c(e) \).

- \( c_{x_j}^{E_{2,j}}(S_{j+1}) - c_{x_{i+1}}(S_{j+1}) = c_{x_j}^{E_{2,j}}(S) - c_{x_{i+1}}(S) = 0 \) since all edges in \( E_2 \) are shared.

- \( c_{x_j}^{E_{3,j}}(S_{j+1}) \geq c_{x_j}^{E_{3,j}}(S) \) since \( x_j \)'s cost on this edge set has not decreased, since no sharing has been added.

- \( c_{x_{i+1}}^{E_{4,j}}(S_{j+1}) \leq c_{x_{i+1}}^{E_{4,j}}(S) \) since sharing on these edges only increases.

These facts give us

\[
2c_{x_{i+1}}(S_{j+1}) - c_{x_j}(S_{j+1}) = c_{x_{i+1}}^{E_{4,j}}(S_{j+1}) + c_{x_{i+1}}^{E_{2,j}}(S_{j+1}) \\
- c_{x_j}^{E_{1,j}}(S_{j+1}) - c_{x_j}^{E_{2,j}}(S_{j+1}) - c_{x_j}^{E_{3,j}}(S_{j+1}) \\
\leq c_{x_{i+1}}^{E_{4,j}}(S) + c_{x_{i+1}}^{E_{2,j}}(S) - c_{x_j}^{E_{2,j}}(S) - c_{x_j}^{E_{1,j}}(S) + \sum_{e \in E_{1,j}} \frac{1}{n_e(S)} c(e) - c_{x_j}^{E_{3,j}}(S) \\
= c_{x_{i+1}}(S) - c_{x_j}(S) + \sum_{e \in E_{1,j}} \frac{1}{n_e(S)} c(e).
\]

Then,

\[
c_{x_j}(S_j) - c_{x_j}(S_{j+1}) \leq \sum_{h=j}^{i} 2 \frac{1}{h - j + 1} c(e_h) + c_{x_{i+1}}(S_{j+1}) - c_{x_j}(S_{j+1}) \\
\leq c_{x_{i+1}}(S) - c_{x_j}(S) + \sum_{e \in E_{1,j}} \frac{1}{n_e(S)} c(e) + \sum_{h=j}^{i} 2 \frac{1}{h - j + 1} c(e_h).
\]

112
Summing over all \( j \) gives

\[
\sum_{j=1}^{i} (c_{x_j}(S_j) - c_{x_j}(S_{j+1})) \geq 0 \implies
\]

\[
\sum_{j=1}^{i} (c_{x_j}(S) - c_{x_{i+1}}(S)) \leq \sum_{j=1}^{i} 2 \sum_{h=j}^{i} \frac{1}{h-j+1} c(e_h) + \sum_{j=1}^{i} \sum_{e \in E_{1,j}} \frac{1}{n_e(S)} c(e)
\]

\[
\leq 2 \sum_{j=1}^{i} H_j c(e_j) + \sum_{j=1}^{i} c(e_j) \leq 4 \sum_{j=1}^{i} H_j c(e_j).
\]

Assume that for all \( i, 1 \leq i \leq k \), the prefix move given in Theorem 48 for prefix \((x_1, \ldots, x_i)\) does not reduce potential. Then we show that \( p_T(x, y) \) is homogenous. Let

\[
g_j = \begin{cases} 
\frac{1}{j^2} & \text{for } j < k, \text{ and} \\
\frac{1}{j} & \text{for } j = k.
\end{cases}
\]

Note that \( \sum_{j=1}^{k} g_j = 1/i \) for all \( i \leq k \). From Lemma 49 we have

\[
\sum_{j=1}^{i} c_{x_j}(S) - c_{x_{i+1}}(S) \leq 2 \sum_{j=1}^{i} H_j c(e_j) \quad \forall i, 1 \leq i \leq k
\]

\[
\implies \sum_{i=1}^{k} \left[ g_i \sum_{j=1}^{i} c_{x_j}(S) - c_{x_{i+1}}(S) \right] \leq \sum_{i=1}^{k} 2g_i \sum_{j=1}^{i} 2H_j c(e_j).
\]

Rewriting \( \sum_{j=1}^{i} c_{x_j}(S) - c_{x_{i+1}}(S) \) as \( \sum_{j=1}^{i} \sum_{h=j}^{i} c_{x_h}(S) - c_{x_{h+1}}(S) \) and rearranging, we obtain

\[
\sum_{i=1}^{k} \left[ g_i \sum_{j=1}^{i} j (c_{x_j}(S) - c_{x_{j+1}}(S)) \right] \leq \sum_{i=1}^{k} 2g_i \sum_{j=1}^{i} 2H_j c(e_j).
\]
Rearranging the sums, we get
\[
\sum_{j=1}^{k} \left[ j \left( c_{x_j}(S) - c_{x_{j+1}}(S) \right) \sum_{i=j}^{k} g_i \right] \leq 4 \sum_{j=1}^{k} H_j c(e_j) \sum_{i=j}^{k} g_i
\]
\[
\Rightarrow \sum_{j=1}^{k} c_{x_j}(S) - c_{x_{j+1}}(S) \leq 4 \sum_{j=1}^{k} \frac{H_j}{j} c(e_j)
\]
\[
\Rightarrow c_x(S) - c_y(S) \leq 4 \sum_{\alpha \geq 0} \left[ 256^{\alpha+1} \sum_{j=1}^{\frac{n_{\alpha,X}}{j}} H_j \right] \leq 4 \sum_{\alpha \geq 0} 256^{\alpha+1} H_{n_{\alpha,X}}^2.
\]

This completes the proof of Theorem 48.

Before we can analyze the ABSORB function, we need to show that the precondition for the ABSORB function (see just before line 33) is satisfied. We state and prove the precondition in Lemma 51. The proof of Lemma 51 requires a homogenous solution, so we first prove homogeneity in the following lemma (Lemma 50).

**Lemma 50.** When the while loops on lines 2 and 26 terminate, \(N(v)\) is homogenous.

*Proof.* Let \(S = S_{curr}\). Let \(x, y \in N(v)\) such that the path in \(T^+\) from \(x\) to \(y\) does not contain \(v\). Let \(P\) be this path. In the simplest case, both \(x\) and \(y\) are in \(T^*\), and \(u_v\), if it exists, does not lie in \(P\). Then, we can bound the cost difference between \(x\) and \(y\) using the homogenous property, guaranteed by line 3/27 of the MAINLOOP function. That is, \(|c_x(S) - c_y(S)| \leq \frac{|e_v|}{14}\). Next let us consider the general case, in which both \(x\) and \(y\) are nonterminals, and \(u_v \in P\). Let \(P = (x, t_x, \ldots, u_t, u_v, u_r, \ldots, t_y, y)\). We
have the following bounds:

\[ |c_x(S) - c_{x_k}(S)| \leq \frac{|c(e_v)|}{64}, \]
\[ |c_{x_k}(S) - c_{u_l}(S)| \leq \frac{|c(e_v)|}{14}, \]
\[ |c_{u_l}(S) - c_{u_r}(S)| \leq \frac{2|c(e_v)|}{256}, \]
\[ |c_{u_r}(S) - c_{u_l}(S)| \leq \frac{|c(e_v)|}{14}, \]
\[ |c_y(S) - c_{x_k}(S)| \leq \frac{|c(e_v)|}{64}. \]

Combining these ensures \( |c_x(S) - c_y(S)| \leq \frac{23|c(e_v)|}{112}. \)

We use Lemma 50 to show the precondition for the ABSORB function.

**Lemma 51.** If the for loops on line 15/35 terminate without the algorithm returning, then

\[ c_p(S_{curr}) \geq c_v(S_{curr}) - \frac{2 \cdot |c(e_v)|}{7} \]

for all \( p \in N(v), p \neq u_v. \)

**Proof.** Let \( S = S_{curr}. \) Again, we suppose that \( u_v \) is not adjacent to \( v \) in \( T^*. \) Suppose the claim does not hold, that is, \( c_v(S) - c_p(S) > \frac{2 \cdot |c(e_v)|}{7} \) for some \( p \in N(v) \) Let \( P = (p, \ldots, q, v) \) be the path from \( p \) to \( v \) in \( T^+ \) and suppose \( q \neq u_v. \) By Lemma 50,

\[ |c_p(S) - c_q(S)| \leq \frac{23|c(e_v)|}{112} \implies c_v(S) - c_q(S) > \frac{9|c(e_v)|}{112}. \]

But, the cost of edge \((v, q)\) is at most \( \frac{|c(e_v)|}{256} < \frac{9|c(e_v)|}{112}, \) so \( v \) must have had an improving move in line 16/36.
Now suppose \( u_v \) is adjacent to \( v \) in \( T^* \) and lies in \( P \). Then \( P = (p, \ldots, q, u_v, v) \).

By Lemma 50,

\[
|c_p(S) - c_q(S)| \leq \frac{23|c(e_v)|}{112} \implies c_v(S) - c_q(S) > \frac{9|c(e_v)|}{112},
\]

But, the cost of edge \((v, u_v)\) is at most \( \frac{|c(e_v)|}{256} \) (since this edge is in \( T^* \)), and the same is true of the edge \((u_v, q)\). Therefore \( v \) can switch its path to \((v, u_v) \cup (u_v, q) \cup p_q(S)\) and pay cost at most \( 2\frac{|c(e_v)|}{256} + c_q(S) < \frac{9|c(e_v)|}{112} + c_q(S) < c_v(S) \). Note that this improving move for \( v \) implies an improving move for \( u_v \) in line 16 where \( p_{u_v}(S) \) no longer includes \( v \) but instead consists of \((u_v, q) \cup p_q(S)\).

Now that we have shown the precondition is satisfied, we must show that the \textsc{Absorb} function reduces potential.

**Theorem 52.** If \( c_q(S_{\text{curr}}) \geq c_v(S_{\text{curr}}) - \frac{2|c(e_v)|}{7} \) for all \( q \neq u_v \in N(v) \), then every strategy change in \textsc{Absorb} reduces potential.

The proof follows from the following three lemmas.

**Lemma 53.** At the beginning of the \textsc{Absorb}(\( v \)) function, \( v \) and \( u_v \) (if \( v \) is a non-terminal) are the only vertices using \( e_v \).

**Proof.** First consider the case where \( v \) is a nonterminal. We show that no edge added to the solution by the critical move (this is at least \( e_v \), and possibly \( e_{u_v} \) as well) is used by any terminal other than \( u_v \) as a result of lines 1 to 22. At the start of \textsc{MainLoop} on line 1, \( u_v \) is the only terminal using \( e_v \) and \( e_{u_v} \), by the definition of a critical move.

Note that \textsc{Homogenize}(\( X \)) only increases sharing on edges in \( X \) and the path \( p_x(S) \) for some \( x \in X \). However, since in line 3 we only consider segments not containing \( v \), no edge adjacent to \( v \) is in \( X \) either. And since \( v \) is the only vertex
using $e_v$ and $v \notin X$, there is no $x$ for which $e_v \in p_v(S)$. Therefore line 3 does not increase sharing on $e_v$. An identical argument shows that this line does not increase sharing on $e_{uv}$, if $v$ is a nonterminal.

Next consider line 5 of Algorithm 4. The only edges on which sharing can increase are $(x, u_v)$, $(u_v, y)$, and edges in $p_y(S)$. By our simplifying assumption, $u_v$ is not adjacent to $v$ in $T^*$ and therefore $x, y \neq v$. Therefore neither $(x, u_v)$ nor $(u_v, y)$ is equal to $e_{uv}$ or $e_v$. Additionally, $e_{uv}, e_v \notin p_y(S)$. Therefore sharing does not increase on either $e_{uv}$ or $e_v$.

Line 9 increases sharing on exactly one of $p_w(S)$ and $p_{tw}(S)$, as well as $\sigma_w$, but only for $w \neq v,u_v$. We know that $u_v$ is the only terminal using $e_{uv}$ or $e_v$ in $S$, so neither of these outcomes increases sharing on $e_v$. Line 12 uses only the MAKE_TREE function, which, by definition, does not increase the sharing on $e_{uv}$ or $e_v$.

Lastly consider the for loop beginning at line 15. By definition, $\text{ABSORB}(v)$ is only run if the for loop does not result in a change to $S_{curr}$, so this can not increase sharing on $e_v/ e_{uv}$.

If the condition in line 16 is ever true, then the algorithm does not begin the $\text{ABSORB}$ function. Therefore we only need to consider what happens when the condition in line 16 is never true, in which case the entire for loop has no effect.

Now consider the case where $v$ is a terminal. Note that before the start of the $\text{MAINLOOP}$ function beginning at Line 25, $v$ is the only vertex using $e_v$ (either because $v$ only added a single new edge for the critical move, or $v$ added two edges but the execution of $\text{MAINLOOP}$ on the second edge did not increase sharing on $e_v$ by the argument above). Using the same argument as for the case of $v$ being a nonterminal, we can prove that this function does not increase sharing on $e_v$, and we omit the details to avoid repetition.

\begin{lemma}
Let $\{q_1, \ldots, q_k\}$ denote all vertices in $N(v)$ sorted by breadth first order
\end{lemma}
from \( v \) according to \( T^* \). Then changing \( q_i \)'s strategy as in Lines 39 and 40 is potential decreasing for all \( i \in \{1 \ldots k\} \).

**Proof.** Suppose that \( e_v = \{v, w\} \). Let \( S \) be the solution before \textsc{Absorb}(\( v \)) is called and let \( S^i \) be the solution after \( q_i \) has changed strategy. Therefore \( c_{q_i}(S^{i-1}) \) is the cost paid by \( q_i \) directly before switching, and \( c_{q_i}(S^i) \) is the cost paid by \( q_i \) directly after switching. \( c_{q_i}(S^{i-1}) \) is exactly equal to the cost paid by \( q_i \) before any players switched, minus the reduction in \( q_i \)'s cost due to sharing on edges in \( p_{q_i}(S) \) from \( q_1, \ldots, q_{i-1} \) changing their strategies. Since we know that \( e_v \notin p_{q_i}(S) \), we can divide this reduction into two components: the reduction due to sharing on edges in \( N(v) \cap T^* \cap p_{q_i}(S) \), and the reduction due to sharing on edges in \( p_w(S) \cap p_{q_i}(S) \). Denote the latter quantity by \( \text{dec}_w \). The former quantity is upper bounded by the maximum cost \( q_i \) could pay on \( N(v) \cap T^* \) in \( S \) (remembering that \( u_v \) may not be using any edges in \( N(v) \) and thus not contributing to sharing), \( 2 \cdot \frac{|c(e_v)|}{56} + 2 \cdot \frac{|c(e_v)|}{256} \). So we can upper bound the total decrease in \( q_i \)'s cost due to the players’ switching by \( \frac{|c(e_v)|}{28} + \text{dec}_w + \frac{|c(e_v)|}{128} \). Therefore

\[
c_{q_i}(S^{i-1}) \geq c_v(S) - \frac{2}{7} \cdot \frac{|c(e_v)|}{28} - \text{dec}_w - \frac{|c(e_v)|}{128}.
\]

Suppose that \( v \) is a terminal and consider the cost paid by \( q_i \) directly after switching, \( c_{q_i}(S^i) \). \( q_i \) is sharing edge \( e_v \) with at least one other player (namely \( v \)), so on the edges shared with \( v \), \( q_i \) is paying at most \( c_v(S) - \frac{|c(e_v)|}{2} - \text{dec}_w \). And on the edges not shared with \( v \), namely the edges in \( p_{T^*}(q_i, v) \), \( q_i \) pays at most

\[
2 \sum_{\alpha > 0} 256^{\alpha + 1} H_{n, N^+(v), \alpha} + 2 \cdot \frac{|c(e_v)|}{256} \leq \frac{|c(e_v)|}{56} + \frac{|c(e_v)|}{128}.
\]

So

\[
c_{q_i}(S^i) \leq c_v(S) - \frac{|c(e_v)|}{2} - \text{dec}_w + \frac{|c(e_v)|}{56} + \frac{|c(e_v)|}{128} < c_v(S) - \text{dec}_w - \frac{6}{14} \cdot \frac{|c(e_v)|}{14} < c_{q_i}(S^{i-1}).
\]

(5.1)

Therefore, it is an improving move for \( q_i \) to switch.

Now suppose that \( v \) is a nonterminal. Then by definition, there is some terminal
using the edges \( \{u, v\} \) and \( e_v \). Suppose that \( q_i \neq q_1 \). Then when \( q_i \) switches in the absorbing process, it shares the cost of \( e_v \) with \( q_1 \). For all \( i \geq 2 \), \( q_i \) shares the cost of edge \( e_v \) with (at least) \( q_1 \). Therefore, Equation 5.1 holds for all \( i \).

The last case is when \( v \) is a nonterminal and \( q_1 = q_i \). But if this is the case then edge \( \{u, v\} \in T^* \). If this were not true, then there is some path \( p^*_T(u, v) = \{u, x_1, x_2, \ldots, x_n, x_{n+1} = v\} \). Since the network is quasi-bipartite, \( x_n \) must be a terminal, and \( x_n \) comes before \( q_i \) in a breadth first traversal of \( T^* \) rooted at \( v \), contradicting that \( q_i = q_1 \). Therefore \( q_i \) is already using strategy \( \{u, v\} \cup p_v(S) \), so there is no switch to be done by \( q_1 \) in step 1 of the \textsc{Absorb} function. For \( i \geq 2 \), \( q_i \) pays at most half the cost of edge \( e_v \) (since it is shared with \( u \)), so Equation 5.1 holds.

\[ \text{Lemma 55.} \quad \text{Let } \{s_1, \ldots, s_k\} \text{ denote all nonterminals in } N(v) \setminus T^* \text{ with class}(e_{s_i}) \leq \text{class}(e_v), \text{ sorted in breadth-first order from } r \text{ according to } S. \text{ Then for all } i \in \{1 \ldots k\}, \text{ it is an improving move for } s_i \text{ to switch as in Line 45. Moreover, after } \textsc{Absorb}(v) \text{ is completed, } e_{s_i} \text{ is no longer in the solution and is replaced by } \sigma_{s_i} \text{ (that is, } \sigma_{s_i} \text{ is the first edge on } s_i \text{'s path to the root, } p_{s_i}. \)

\textit{Proof.} Let \( S \) be the solution before \( \textsc{Absorb}(v) \) is called. Let \( s_i \in N(v) \setminus T^* \). Consider first the case where \( e_{s_i} = \sigma_{s_i} \). In this case \( s_i \) is already taking strategy \( e_{s_i} \cup p_{s_i}(S') \), since descendants are absorbed in lines 39 and 40, and there is no change to make in line 45. Suppose for the rest of the proof that \( e_{s_i} \neq \sigma_{s_i} \).

Let \( S' \) be the solution after \( s_i \) has switched strategy. In particular, \( c_{s_i}(S') \) is the cost paid by \( s_i \) directly before switching, and \( c_{s_i}(S') \) is the cost paid by \( s_i \) directly after switching. \( c_{s_i}(S^i+1) \) is the cost paid by \( s_i \) before the absorbing process began, \( c_{s_i}(S) \), minus the cost reduction due to sharing on \( p_{s_i}(S) \) due to other vertices switching as part of the absorbing process. Since \( \sigma_{s_i} \notin p_{s_i}(S) \), there is no contribution to the latter term due to sharing on \( \sigma_{s_i} \). The only other edges in \( p_{s_i}(S) \) that can
have increased sharing as a result of previous moves in the absorbing process are edges in $N(v) \cap T^* \cap p_{s_i}(S)$ and edges in $p_w(S) \cap p_{s_i}(S)$. Therefore we have the same bound on this term as in Lemma 54, $\frac{|c(e_v)|}{28} + \frac{|c(e_w)|}{128} + dec_w$. We also know, from the precondition for $\text{ABSORB}$, that

$$c_{s_i}(S) \geq c_v(S) - \frac{2 \cdot |c(e_v)|}{7}.$$ 

Therefore the cost that $u$ pays immediately before switching, $c_{s_i}(S^{i+1})$, satisfies

$$c_{s_i}(S^{i+1}) \geq c_v(S) - \frac{2 \cdot |c(e_v)|}{7} - \frac{|c(e_v)|}{28} - \frac{|c(e_v)|}{128} - dec_w = c_v(S) - dec_w - \frac{10 \cdot |c(e_v)|}{28}.$$ 

Now consider $s_i$’s cost immediately after switching (before any descendants switch), $c_{s_i}(S^i)$. $s_i$ pays at most the entire cost of $\sigma_{s_i}$. As in the proof of Lemma 54, $s_i$ pays at most $c_v(S) - \frac{|c(e_v)|}{2} - dec_w$ on edges shared with $v$, and at most $\frac{|c(e_v)|}{56} + \frac{|c(e_v)|}{128}$ on edges between $t_{s_i}$ and $v$. So $c_{s_i}(S^i)$ satisfies

$$c_{s_i}(S^i) \leq c(\sigma_{s_i}) + c_v(S) - \frac{|c(e_v)|}{2} - dec_w + \frac{|c(e_v)|}{28} + \frac{|c(e_v)|}{128}$$

$$\leq \frac{|c(e_v)|}{64} + c_v(S) - \frac{|c(e_v)|}{2} - dec_w + \frac{|c(e_v)|}{14}$$

$$< c_v(S) - dec_w - \frac{11 \cdot |c(e_v)|}{28} < c_{s_i}(S^{i+1}),$$

which proves the first part of the lemma. The second part follows from the definition of the move.  

\[\Box\]

Theorems 48 and 52 prove that the entire main loop is potential reducing. Since the minimum decrease in potential is bounded away from zero, and the potential is always at least zero, the algorithm necessarily terminates.

However, termination alone does not guarantee that the final state is a Nash equilibrium. Since we have restricted the set of moves that the algorithm can perform,
we must show that whenever an improving move is available to some terminal, there is also an improving move that is either a safe or critical move.

Lemma 56. The final state reached by the algorithm, $S_f$, is a Nash equilibrium.

Proof. Suppose for contradiction that $S_f$ is not a Nash equilibrium, that there is an improving deviation for some player $q$. Consider the most improving deviation (that with lowest cost) and denote this lowest cost path to the root as $p_q(S_f') = \{q = q_1, q_2, \ldots, q_k = r\}$. Of all vertices $q_i$, consider the terminal with highest index such that $p_q(S_f') = \{q_i, q_{i+1}, \ldots, q_k = r\}$. Since $p_q(S_f')$ is of lower cost to $q$ than the path $\{q = q_1, q_2, \ldots, q_{i-1}, q_i\} \cup p_q(S_f)$, it must also be the case that $\{q_i, q_{i+1}, \ldots, q_k = r\}$ is of lower cost to $q_i$ than $p_{q_i}(S_f)$. And by the maximality of index $i$, $q_i$ has an improving move where she can add edge $\{q_i, q_{i+1}\}$ (if $q_{i+1}$ is a terminal), or edges $\{q_i, q_{i+1}\}, \{q_{i+1}, q_{i+2}\}$ (if $q_{i+1}$ is not a terminal). This is necessarily either a safe or critical move.

Therefore, if an improving move exists for any player at state $S_f$, then a safe or critical move exists for some player, contradicting termination of the algorithm. \qed

5.5.2 Cost Analysis

Our goal for this section is to show our main result, Theorem 46. We will show that $c(S_f) = O(c(T*))$. That is, we will show that the cost of the final Nash equilibrium reached by the algorithm, $S_f$, is within a constant factor of the cost of the optimal tree, $T^*$.

To establish the theorem, it is sufficient to show that $c(S_f \setminus T^*) = O(c(T*))$. We devise a charging scheme that distributes the cost of edges in $S_f \setminus T^*$ among edges in $T^*$. Each $e \in S_f \setminus T^*$ must be an $e_v$ edge for some vertex $v$. Furthermore, these $e_v$ edges were not later removed as the result of an absorbing process initiated from another $e_{v'}$. At a high level, this allows us to distribute the cost of each $e_v$ to the
edges in the neighborhood \( N(v) \cap T^* \), since the \texttt{Absorb}(v) function removes many other \( e_{v'} \) edges where \( v' \in N(v) \) from the solution. When \( v \) is a terminal, this is the same argument used in [23]; however, we will need to take special care when distributing cost for \( e_v \) when \( v \) is a nonterminal as well as for some \( \sigma_v \) edges when \( v \notin T^* \).

We first consider a set of edges that we will not charge to their neighborhood. Define

\[
E_\sigma = \{ e_v \in S_f | v \text{ is a nonterminal, } \frac{|c(e_v)|}{64} \leq c(\sigma_v) \}.
\]

We bound the cost of \( E_\sigma \) by the cost of edges in \( S_f \setminus E_\sigma \).

**Lemma 57.** \( c(E_\sigma) = O(c(S_f \setminus E_\sigma)) \).

**Proof.** Let \( e_v = (v, w) \in E_\sigma \), where \( v \) is a nonterminal. First observe that since \( (v, w) \in S_f \), \( v \) is not a leaf. Let \( e_u = (u, v) \), where \( u \) is necessarily a terminal. Since \( c(e_v) \leq 64 \cdot c(\sigma_v) \), we have that \( c(e_v) \leq 64 \cdot c(e_u) \) from the definition of \( \sigma_v \). Thus, we charge \( c(e_v) \) to \( c(e_u) \). Observe that no edge in \( S_f \setminus E_\sigma \) is charged more than once since every nonterminal has a unique parent in \( S_f \), and edges in \( S_f \setminus E_\sigma \) are only charged the cost of the first edge used by their parent (if at all). \( \square \)

Our goal now is to find a set of edges \( e_v \) such that the right neighborhoods associated with edges of the same class are not overlapping. In the absence of non-terminals, this is simple: For every edge in \( S_f \setminus T^* \), the right neighborhoods of vertices corresponding to edges of the same class being overlapping implies that each edge is contained in the other’s neighborhood. Therefore, we argue that the second edge to arrive would have deleted the first through the \texttt{Absorb} function, which gives a contradiction. With nonterminals, the same property does not hold. When edge \( e_v \) is added for some nonterminal \( v \), \( e_{uv} \) will not be deleted from the solution, even if \( u_v \) falls in \( v \)'s neighborhood. The presence of \( \sigma_v \) for which no \texttt{MainLoop}(\( \sigma_v \))
was run (added, e.g., in line 39) further complicates things. To show that no right neighborhoods overlap, we will therefore remove some edges from $S_f \setminus (T^* \cup E_\sigma)$.

For nonterminal $v$, if $v$ is adjacent to at least two edges in $S_f \setminus (T^* \cup E_\sigma)$ and $\sigma_v$ is one such edge, remove $\sigma_v$ and charge it to one of the remaining edges adjacent to $v$. Next, for any pair of edges $e_u$ and $e_v$ in $S_f \setminus (T^* \cup E_\sigma)$ such that $u$ was the terminal which added $e_v$, we delete the smaller of $e_u$ and $e_v$ and charge it to the remaining edge. We are left with a set of edges which we denote $E^*$, each of which has been charged by at most two edges that were removed (and each edge removed is charged to some edge in $E^*$).

Our argument will charge to each edge in $T^*$ at most one edge in $E^*$ of each class. To make the argument simpler, it is desirable to charge those $\sigma_v$’s for which \textsc{MainLoop}$(\sigma_v)$ was never run to higher classes than their actual classes. To this end, we increase the cost of each such $\sigma_v$ to $c(e_{\sigma_v})$, the cost of the first edge on $v$’s path in the state just before $\sigma_v$ was added.

**Lemma 58.** For edges $e_u, e_v \in E^*$, if $\text{class}(e_v) = \text{class}(e_u)$, then $N^+(v)$ and $N^+(u)$ are disjoint.

**Proof.** If $\text{class}(e_v) = \text{class}(e_u)$ and $N^+(v)$ and $N^+(u)$ overlap, then the vertex corresponding to the edge arriving first is in the neighborhood of the other. Suppose $N^+(v)$ and $N^+(u)$ overlap, $e_u$ preceded $e_v$, and $u \notin N(v)$. If $u \notin T^*$, $c(\sigma_u) \leq \frac{|c(e_u)|}{64} = \frac{|c(e_v)|}{64}$, and thus $t_u \in N(v)$ implies $u \in N(v)$. Then $t_u$ (or $u$ if $u \in T^*$) must lie to the left of $N(v)$ in MC, which means that $N^+(u)$ strictly contains $N^-(v)$, a contradiction given that $\text{class}(e_u) = \text{class}(e_v)$.

Suppose $e_u, e_v \in E^*$ with $\text{class}(e_v) = \text{class}(e_u)$. Suppose $e_u$ preceded $e_v$ and $N^+(v)$ and $N^+(u)$ overlap, which implies $u \in N(v)$ as shown above. We consider two cases for edge $e_v$, and derive contradictions in all cases. First, suppose \textsc{MainLoop}$(e_v)$ was run when $e_v$ was added. Then $e_u$ cannot exist after the completion of
If the MAINLOOP($e_v$) was not run, then $e_v = \sigma_w$ for some $w$, and $e_v$ was introduced because $w$ was in $Z$. Let $e_{\sigma_w}$ be the first edge on $w$'s path to the root in the state just after $\sigma_w$ was added, which we denote $S_{\sigma_w}$ (this may be different from $w$'s current first edge, $e_w$). Then MAINLOOP($e_{\sigma_w}$) caused the deletion of all edges $e_q$ such that $q \in N_{S_{\sigma_w}}(w)$. We consider two possible times when $e_u$ was added: If $e_u$ was added before $e_{\sigma_w}$, then MAINLOOP($e_{\sigma_w}$) deleted $e_u$ since $u \in N_{S_{\sigma_w}}(w)$ because $\text{class}(e_u) = \text{class}(e_{\sigma_w})$. If $e_u$ was added after $e_{\sigma_w}$, $\text{class}(e_u) = \text{class}(e_{\sigma_w})$ implies that $w \in N(u)$. But then $e_{\sigma_w}$ was removed from the solution and $w$ was removed from $Z$, so the addition of $\sigma_w = e_v$ was not possible.

Given Lemma 58, the scheme from [23] for distributing the cost of each $e_v$ to its neighborhood can be applied directly. This gives us the following lemma, which along with Lemma 57 establishes Theorem 46.

**Lemma 59.** The cost of each $e_v \in E^*$ can be distributed to the edges in $N^+(v)$ (and its boundary) such that the total charge on any edge $e' \in T^*$ is $O(c(e'))$.

**Proof.** Let $e_v \in E^*$. Let $\alpha = \text{class}(e_v)$. Throughout this proof we will be interested only in edges $N^+(v) \cap T^*$. For simplicity, we will simply write $N^+(v)$ instead of $N^+(v) \cap T^*$.

We first consider the case where $N^+(v) = MC = N(v)$. If $v$ is a terminal, then $e_v$ is the only edge not in $S_f \setminus E_\sigma$, since all other terminals are following their path in $T^*$ to $v$. But all edges added by a critical move must have $c(e) \leq c(T^*)$ by definition. So in this case, $c(S_f \setminus E_\sigma) \leq c(T^*)$. If $v$ is a Steiner vertex, then $S_f \setminus E_\sigma$ can also include $\sigma_v$, since all terminals $u$ are using strategy $p_{T^*}(u, t_v) \cup \sigma_v \cup p_v(S)$. Since $c(\sigma_v) \leq c(e_v)$, we have $c(S_f \setminus E_\sigma) \leq 2c(T^*)$.

Now we consider the case where $N^+(v) \neq MC$. Recall that every edge in $N^+(v)$ has class at most $\alpha - 2$. Let $a \notin N^+(v)$ be the first edge to the right of $N^+(v)$ in $T^*$,
and let \( \mu = \text{class}(a) \). There are two cases.

**Case 1:** \( \mu \geq \alpha - 1 \). In this case we can charge \( c(e_v) \) to \( a \). We show that only one edge of each class will get charged to \( a \). Suppose that this is not the case, that there is some \( e_{v'} \) with \( \text{class}(e_{v'}) = \alpha \), such that \( a \) is the first edge to the right of \( N^+(v') \) in \( T^* \). Then the edges of \( N^+(v) \) have non-empty intersection with those of \( N^+(v') \), contradicting Lemma 58.

So the total cost charged to edge \( a \) in this fashion is no greater than \( \sum_{\gamma=0}^{\mu+1} 256^\gamma + 1 < 256^\mu + 3 \leq 256^3 c(e) \). Since each \( e \in T^* \) appears at most twice in \( MC \), \( e \) is charged a total cost of at most \( 2 \cdot 256^3 c(e) \).

**Case 2:** \( \mu \leq \alpha - 2 \). In this case we charge \( c(e_v) \) to a subset of the edges in \( N^+(v) \).

We first prove a technical claim.

**Claim 60.** There exists some class \( 1 \leq \beta \leq \alpha - 2 \) such that

\[
\frac{256^\beta + 1 H^2_{n^+(v), \beta}}{256^\alpha - 1} \geq \frac{1}{256^{\frac{\alpha-\beta}{2}}}. 
\]

**Proof.** Assume, for contradiction, that there exists no such \( \beta \). That is, \( \frac{256^\gamma + 1 H^2_{n^+(v), \gamma}}{256^\alpha - 1} < \frac{1}{256^{\frac{\alpha-\gamma}{2}}} \) for all \( 0 \leq \gamma \leq \alpha - 2 \). Then we can sum over all classes:

\[
\sum_{\gamma=0}^{\alpha-2} 256^\gamma + 1 H^2_{n^+(v), \gamma} = \sum_{\gamma=0}^{\alpha-2} \frac{256^\gamma + 1 H^2_{n^+(v), \gamma}}{256^\alpha - 1} 256^\alpha - 1 
\]

\[
< 256^{\alpha - 1} \sum_{\gamma=0}^{\alpha-2} \frac{1}{256^{\frac{\alpha-\gamma}{2}}} 
\]

\[
< 256^{\alpha - 1}. \quad (5.2)
\]

However, we also know from maximality of \( N^+(v) \) that

\[
2 \left( \sum_{\gamma=0}^{\alpha-2} 256^\gamma + 1 H^2_{n^+(v), \gamma} + 256^\mu + 1 H^2_{n^+(v), \mu+1} - 256^\mu + 1 H^2_{n^+(v), \mu} \right) \geq \frac{256^\alpha}{56},
\]

125
which implies that, using the fact that $H_{i+1}^2 - H_i^2 \leq \frac{3}{4}$ for any $i \geq 0$,

$$2 \sum_{\gamma=0}^{a-2} 256^{\gamma+1} H_{n_{N^+(v),\gamma}}^2 \geq \frac{256^a}{56} - 2 \cdot 256^{\mu+1}(H_{n_{N^+(v),\mu+1}}^2 - H_{n_{N^+(v),\mu}}^2)$$

$$\geq \frac{256^a}{56} - \frac{10}{4} 256^{a-1} > 256^{a-1},$$

contradicting Equation 5.2.

Consider all edges of class $\beta$. By using the inequality $H_i \leq 1 + \ln i$ and rearranging the equation from Claim 60, we get that

$$n_{N^+(v),\beta} \geq e^{\sqrt{256^\frac{a-\beta+1}{2}}} - 1.$$

We charge $c(e_v) \leq 256^{a+1}$ equally across all edges of class $\beta$. Therefore each edge of class $\beta$ is charged at most

$$256^{a+1} \frac{1}{e^{\sqrt{256^\frac{a-\beta+1}{2}}} - 1} = 256^\beta \frac{256^{a-\beta+1}}{e^{\sqrt{256^\frac{a-\beta+1}{2}}} - 1}.$$

Suppose that any other edge $e_{v'}$ of class $\alpha$ is (partially) charged to some edge $a$ that $e_v$ has also been partially charged to. Then $a \in N^+(v)$ and $a \in N^+(v')$ overlap, a contradiction to Lemma 58. Therefore the total amount charged to $a$ is at most

$$\sum_{\gamma \geq \beta+2} 256^\beta \frac{256^{a-\beta+1}}{e^{\sqrt{256^\frac{a-\beta+1}{2}}} - 1} = 256^\beta \sum_{z \geq 0} e^{\sqrt{256^\frac{a+3}{2}}} - 1$$

$$= O(256^\beta)$$

$$= O(c(a)).$$

Since each edge $e$ appears in $MC$ at most twice, the total cost of $E^*$ from this type of charging is $O(c(T^*))$. This proves the lemma.
Conclusions

In this dissertation, we gave algorithms for solving problems in networks in uncertain settings. We proposed a generalization of the $k$-server problem called all-or-one $k$-server, and gave algorithms for solving this problem. Motivated by deploying distributed applications on a network, we studied the graph retraction problem, giving algorithms for retracting graphs to cycles. Motivated by denial of service of attacks in networks, we proposed the symmetric interdiction model, and gave general algorithms for problems in this model and an algorithm for the symmetric matching interdiction problem. Finally, we made progress towards a better price of stability bound for multicast games, showing a constant upper bound on the price of stability of multicast games for quasi-bipartite graphs.

There are many interesting open questions and areas of research that arise from the results presented in this dissertation.

Our results on all-or-one $k$ server apply only on uniform metrics. It is an open question whether we can obtain similar results for arbitrary metrics.

Our work on retraction also leaves a natural open question: what is the best approximation ratio for retracting arbitrary graphs to cycles? The $\Theta(\sqrt{n})$ gap left
by our results is large. We showed that we can do no better than this the distance-based lower bound. Instead, we need a lower bound that captures Sperner’s Lemma in order to improve. We showed that for planar graphs we can use such a lower bound to give an optimal algorithm, which is evidence that a similar lower bound may be effective for general graphs. As an intermediate step, it would be interesting to consider classes of graphs more general than planar graphs. Finally, another possible direction is to consider retraction to other types of graphs besides cycles.

The symmetric interdiction model we present is very general. While we showed that an optimal solution to an optimization problem is also a good solution for the corresponding interdiction problem, we also showed that it is possible to improve this further by developing a specialized algorithm, as we did for the matching problem. We can consider giving similar specialized algorithms for other optimization problems. We initially proposed the symmetric interdiction model to model denial of service attacks. Another direction is to consider other models for this problem. For example, we assumed that the network operator is completely oblivious to whether requests are legitimate or malicious. Suppose instead we are given a probability that each request is malicious; how would we decide on traffic to route?

Finally, while we made progress towards finding the price of stability for multicast games, the problem remains unsolved. We conjecture that the price of stability for these games is constant. There are other natural questions in the area of network design games. For example, computing equilibria in these games is unsolved.
Appendix A

Representation as a convex combination

We need to show that the representation of fractional matching as a convex combination of integral matchings can be obtained in polynomial-time. We show something more general - namely, that given a feasible point in the polytope defined by a set of linear constraints (even an exponential number in implicit form as a separation oracle [65]) we can find a representation of the point as a convex combination of the vertices of the polytope, in polynomial-time. Our constructive algorithm is folklore, but for completeness, we describe it in its entirety.

Let $PT$ represent the $d$-dimensional polytope (convex bounded polyhedron) of solutions to $LP = \{C\}$, a set of linear constraints with $F_C$ denoting the face, of dimension at most $d - 1$, generated by constraint $C$. Let $\hat{p} \in PT$ be the given point. The set of constraints may be given in explicit form or implicitly with a bounding ball and a separation oracle [65]. The main idea is to take the ray starting at any vertex $v$ of $PT$ through $\hat{p}$ to its intersection $p_i$ with the face opposite, $F_C$, then recursively represent $p_i$ as the convex combination of vertices $V_C$ of $F_C$; now it is an easy matter to see that $\hat{p}$ can be represented as a convex combination of $v \cup V_C$. In fact, it is easy
to see, by strengthening the inductive hypothesis, that \( \hat{p} \) is the convex combination of at most \( d + 1 \) vertices of \( PT \). All that is left to do is to see that a vertex of a polytope, the point of intersection of a ray with a face of the polytope and the representation of the face as a set of constraints (along with bounding ball and separation oracle, in the general case) can all be computed in polynomial-time. These operations are easy to compute when the constraints are given in explicit form and we leave it to the reader as an exercise. In the general setting of the separation oracle, a vertex can be computed using the ellipsoid algorithm [65]; the point of intersection of a ray with a face can be computed using binary search; and the polytope restricted to the face \( F_C \) can be represented by the same separation oracle augmented with the constraint that \( C \) be satisfied with equality, i.e the restricted polytope lies on the hyperplane (the bounding ball stays the same).

Note that the above proof shows that any point in a \( d \)-dimensional polytope lies in a simplex formed by at most \( d + 1 \) vertices of the polytope. This gives an alternate proof of Caratheodory’s theorem [15]. It also shows that the simplex can be chosen to contain any particular vertex of the polytope. As a small digressional note, it is worth pointing out that the above technique does not extend to showing that every polyhedron (not polytope, polyhedrons may be non-convex) can be triangulated (simpliciated); in fact, though every polyhedron in 2 dimensions (i.e. polygon) can be triangulated this is not true in 3 (or higher) dimensions [100].
Bibliography


Biography

Samuel Mitchell Haney is a Ph.D. candidate in computer science at Duke University and is advised by Debmalya Panigrahi. He received his Bachelor of Science degree in computer science from Tufts University in 2013. He was born in Connecticut on April 11, 1991.