Internet Routing and Internet Service Provision

Henry Lin

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2009-105
http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-105.html

July 29, 2009

Internet Routing and Internet Service Provision

by

Henry Chih-Heng Lin

B. S. (Cornell University) 2003

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION
of the
UNIVERSITY of CALIFORNIA at BERKELEY

Committee in charge:

Professor Christos H. Papadimitriou, Chair
Professor Satish Rao
Professor Richard Karp
Professor David Aldous

Fall 2009

The dissertation of Henry Chih-Heng Lin is approved:

Chair Date

Date

Date

Date

University of California at Berkeley

Fall 2009

Internet Routing and Internet Service Provision

Abstract

Internet Routing and Internet Service Provision

by

Henry Chih-Heng Lin
Doctor of Philosophy in Computer Science

University of California at Berkeley

Professor Christos H. Papadimitriou, Chair

The recent development and expansion of the Internet has created many technical challenges in several diverse research areas. In this thesis, we study recent problems arising in the area of Internet routing and Internet service provision. In the area of Internet routing, we analyze properties of the selfish routing model, which is a mathematical model of users selfishly routing traffic in a network without regard to their effect on other users. Additionally, we study the properties of various random graph models which have been used to model the Internet, and utilize the properties of graphs generated by those random models to develop simple compact routing schemes, which can allow network routing without having each node store very much information. In the area of Internet service provision, we study an online bipartite matching problem, in which a set of servers seeks to provide service to arriving clients with as little interruption as possible. The central theme of this thesis is to analyze precise mathematical models of Internet routing and Internet service provision, and in those models, we show certain properties hold or derive algorithms which work with high probability.

The first model we study is the selfish routing model. In the selfish routing model, we analyze the efficiency of users selfishly routing traffic and study a counterintuitive phenomenon known as Braess's Paradox, which states that adding a link to a network with selfish routing may actually increase the latency for all users. We produce tight and nearly tight bounds on the maximum increase in latency that can occur due to Braess's Paradox in single-commodity and multicommodity networks, respectively. We also produce the first nearly tight bounds on the maximum latency that can occur when traffic is routed selfishly

in multicommodity networks, relative to the maximum latency that occurs when traffic is routed optimally.

In the second part of the thesis, we study random graph models which have been used to model the Internet, and look for properties of graphs generated by those models, which can be used to derive simple compact routing schemes. The goal of compact routing is to derive algorithms which minimize the information stored by nodes in the network, while maintaining the ability of all nodes to route packets to each other along relatively short paths. In this research area, we show that graphs generated by several random graph models used to model the Internet (e.g. the preferential attachment model), can be decomposed in a novel manner, which allows compact routing to be achieved easily. Along the way, we also prove that a Polya urns random process has good load balancing properties, which may be of independent interest.

In the last part of the thesis, we study an online bipartite matching problem, which models a problem occurring in the area of Internet service provision. In our online bipartite matching problem, we imagine that we have some servers capable of providing some service, and clients arrive one at a time to request service from a subset of servers capable of servicing their request. The goal of the problem is to assign the arriving clients to servers capable of servicing their requests, all while minimizing the number of times that a client needs to be switched from one server to another server. Although prior worst case analysis for this problem has not yielded interesting results, we show tight bounds on the number of times clients need to be switched under a few natural models.

As we analyze these problems arising in the Internet from a precise mathematical perspective, we also seek to reflect on the process used to solve mathematical problems. Although the thought process can sometimes be difficult to describe, in one case, we attempt to provide a step-by-step account of how the final result was proved, and attempt to describe a high level algorithm, which summarizes the methodology that used to prove it.

Professor Christos H. Papadimitriou
Dissertation Committee Chair

Contents

List of Figures

Acknowledgements

There are many people I would like to acknowledge for their support in writing this thesis. First, I would like to acknowledge my co-advisors Christos Papadimitriou and Satish Rao. Their support, encouragement, and research guidance were invaluable over the years. In addition to their wisdom and academic advice, there were also memorable concerts with Christos's music band *Lady X and the Positive Eigenvalues* and numerous political discussions with Satish, which provided a nice break from research. They also had their own unique sense of humor, which made meetings enjoyable and interesting.

Besides my main advisors, I would like to thank Richard Karp and David Aldous for providing feedback on my thesis and serving on my dissertation committee. I would also like to thank Richard Karp for providing sound guidance, support, and collaboration over the years, and thanks go to David Aldous, Alistair Sinclair, Umesh Vazirani, Luca Trevisan for teaching some insightful courses during my time at Berkeley. In addition to teaching courses, it was also nice to collaborate with Luca Trevisan and Hoeteck Wee on a complexity project. I would also like to thank my other co-authors and collaborators at Berkeley: Jacob Abernathy, Kamalika Chaudhuri, Costis Daskalakis, Robert Kleinberg. It was great to get to know you and collaborate with you at Berkeley. To my fellow Ph.D. graduates this year Alexandra Kolla and Madhur Tulsiani, congratulations and good luck!

Going back, I would also like to thank members of the Cornell University community. I thank Eva Tardos for advising me as an undergraduate at Cornell, and helping to start my academic career. It was a great opportunity and learning experience to work with Eva and Tim Roughgarden on selfish routing. Jon Kleinberg and Eva also taught an inspiring introductory course on algorithms, which convinced me to study theoretical computer science. I would also like to thank my friends, roommates, and fellow occupants in the Class of 28 and Sperry Hall dormitories at Cornell University, who made sure I experienced everything college has to offer, besides the rigorous academic curriculum. Your support and friendship over the years were invaluable.

To my fellow graduate students at UC Berkeley, thank you for sharing this journey with me with all the highs and lows. I would like to thank all those who have helped me with friendship and advice, technical or otherwise, although I am sure to forget some people. Apologies in advance to those whom I may forget to mention. Besides the graduate students already mentioned above, I would like to especially acknowledge Kris Hildrum

and Alex Fabrikant for some useful early guidance, when I was only a first year graduate student. Thanks also go to the theory graduate students roughly my year and above, for useful guidance and support as well: Andrej Bogdanov, Kevin Chen, Omid Etesami, Brighten Godfrey, Omar Khan, Bonnie Kirkpatrick, Mani Narayanan, Lorenzo Orecchia, Daniel Preda, Sam Riesenfeld, Grant Schoenebeck, Alexandre Stauffer, Kunal Talwar, and Boriska Toth. To the younger students, Gregory Valiant, Thomas Vidick, James Cook, Yaron Singer, and the first year students, I wish you luck and it was great to get to know you during the theory retreat(s). I'm sure you'll be very successful in your graduate studies.

I give many thanks to the members of the CSGSA for hosting so many events that kept us well-fed and entertained over the years. In particular, I would like to acknowledge my friends, Ben Rubinstein and Juliet Rubinstein for hosting movie nights, and Erika Chin, Arel Cordero, Adrian Mettler, Barret Rhoden, Isabelle Stanton, and David Zhu for serving on the social committee and lounge committee. Thanks to my friends who were my roommates: Mircea Dinca, Bowei Du, Krish Eswaran, Craig Hashi, Steve Martin, Radu Miheascu, Matt Piotrowski, Jingyi Shao, and Shane Tackett. Thank you to my friends on the EECS softball team, my fellow GSI's, and other great friends I met over the years: Alex Dimakis, Amin Gohari, Pulkit Grover, Steven Houston, Ann Hsieh, Francois Labelle, Garmay Leung, Joe Makin, David Molnar, Bobak Nazer, Blaine Nelson, Raina Spalek, Robert Spalek, Rahul Tandra, David Tung, Falk Unger, and Helen Yin. Also, thanks to Neha Dave for providing excellent food at the theory lunches.

I also want to thank Cynthia Phillips, Robert Carr, William Hart, Erik Lauer, and my other collaborators and coworkers at Sandia National Laboratories for two enriching and enjoyable internship experiences. Going back even further, I would also like to thank my friends and teachers from Newton, MA and Newton North High School in particular, for providing a sound foundation for academics and life in general.

Last but not least, I would especially like to thank my parents for their support and encouragement over the years. It goes without saying that I would not have been able to do well in life without them.

Chapter 1

Introduction

The recent rise of the Internet has brought about many technological innovations and advances for society, but with these new developments come many challenges and open questions. As the technologies that run and operate services on the Internet vary widely, the challenges and open questions that arise from the development of the Internet are numerous and diverse. There are countless open questions regarding both the efficient operation and transfer of data within the Internet, and regarding the efficient operation of applications running on the Internet. In this thesis, we explore various aspects of both problems, studying problems related to the efficient operation of the Internet and related to the efficient operation of applications running on the Internet.

The research described in this thesis seeks to study the aforementioned challenges facing the Internet from a precise mathematical perspective. We seek to study precise mathematical models of Internet routing and Internet service provision, and seek to prove precise guarantees regarding the behavior or performance achievable in those models. In some cases, we seek to provide worst case guarantees, and in others, we follow the recent trend of analyzing randomized models, which utilize randomness to generate a better model of the behavior we expect to see in the real world. In the random models we analyze, we seek to develop and analyze algorithms which have precise performance guarantees, which hold with high probability. With those goals in mind, the work in this thesis covers three main topics: the selfish routing model, compact routing in "Internet-like" random graphs, and an online bipartite matching problem.

Although the focus of this thesis is to explore the challenges and problems facing Internet design, we also reflect on the mathematical techniques we use to analyze and study

mathematical problems of interest. An auxiliary goal of this thesis is determine if there are some high level algorithms, which can summarize the methodologies we use to prove new mathematical results. Although it may be difficult to determine or define the exact high level algorithm or methodology that we use to prove each result in this thesis, in one case, we provide readers with a detailed step by step account of how a new mathematical result was discovered, and describe a high level algorithm, which was used to generate that new result.

In the next three sections, we provide a high level description of the three areas in Internet routing and Internet service provision that we study in this thesis, and then we provide more thoughts on the goal of studying the methodology we use to prove new mathematical results. Then, in the three subsequent chapters, we explore the three areas we mention below in more detail.

1.1 Selfish Routing, Braess's Paradox, and the Price of Anarchy

The first area we study follows the recent trend of studying game theoretic aspects of Internet routing. Internet routing can be viewed as a noncooperative game between selfish agents who individually seek to minimize the latency incurred by the traffic they send over the Internet, while ignoring the extraneous effects that their traffic has on the latencies incurred by other users. One popular model of noncooperative users sending traffic in a network is known as the *selfish routing* model and was studied extensively in [Rou05].

In the selfish routing model, we imagine that we have a large number of users each sending a small amount of traffic in a network, which is represented collectively as a network flow, and we imagine that users selfishly choose to send their traffic or flow along minimum latency paths without regard to their effect on the latency of other user's traffic. In this model, a network flow in which all traffic is traveling along minimum latency paths is said to be at Nash equilibrium, since no users have any reason to deviate their traffic from the current state. Although flows at Nash equilibrium may be stable with users having no incentive to switch their traffic to other paths, a flow at Nash equilibrium may not be socially optimal in terms of minimizing the average latency incurred by all users, nor is it optimal in terms of minimizing the maximum latency incurred by all users.

Moreover, there is a surprising illustration of the inefficiency that can arise in a network with selfish routing known as Braess's Paradox. Breass's Paradox states that adding a link to a network with selfish routing may actually increase latency for all users, instead of improving the overall latency. This counterintuitive phenomenon more or less occurs because adding a link to a network can cause selfish users to change routes and congest the same links, and as a result, increase the latency for all users. Although Roughgarden's work in [Rou] precisely characterized the worst possible severity of Braess's Paradox in single-commodity networks (where all traffic flows from a single source to a single destination) based on the number of nodes in the network, open questions still remained on the worst possible severity of Braess's Paradox.

For example, the work in [Rou] shows that Braess's Paradox can be arbitrarily severe, if the original network is large enough, and if we are allowed to add an arbitrary number of edges to the network. However, it was still an open question whether or not adding a single edge could cause arbitrarily severe instances of Braess's Paradox in single-commodity networks. The examples in [Rou] show that adding k edges to a single-commodity network can increase the overall latency incurred by users by a factor of $k+1$, but it was unclear if those examples were the worst case that could occur. In the first result of this thesis, we show a matching upper bound, which shows that adding k edges to a single-commodity network can increase the overall latency by at most $k+1$.

Additionally, the story of Braess's Paradox in multicommodity networks (where there is traffic between multiple sources and destinations) was also not complete. The work in [Rou] did not provide any worst case upper bound on the severity of Braess's Paradox in multicommodity networks, and only showed that adding k edges to a multicommodity network could increase the overall latency by as much as $k+1$. In the second result of this thesis, we show that Braess's Paradox can be much worse in multicommodity networks. In fact, it can be exponentially worse, as we show that adding a *single edge* to a network with only *two commodities* (two sources and two destinations), can increase the latency by a factor of as much as $2^{\Omega(n)}$, where n is the number of nodes in the network.

Furthermore, we provide an upper bound which almost matches the lower bound, by showing that the overall latency incurred by all users can increase by at most $2^{O(\min\{kn, m\log n\})}$, when any number of edges are added to a network, where k is the number of commodities in the network, and m is the number of edges. When the number of commodities is a constant, we have a tight bound of $2^{\theta(n)}$ on the severity of Braess's Paradox.

Tangentially, our work on Braess's Paradox in multicommodity networks also provides the first close upper and lower bounds, in multicommodity networks, on what is known as the price of anarchy with respect to the maximum latency objective. The price of anarchy with respect to the maximum latency objective is simply the highest ratio that can occur between the maximum latency that any user incurs in a flow at Nash equilibrium, and the maximum latency that any user incurs in an optimal flow. Our work on Braess's Paradox in multicommodity networks, can also be used to show that the price of anarchy with respect to the maximum latency objective is as bad as $2^{\Omega(n)}$, but is at most $2^{O(\min\{kn, m \log n\})}$.

Lastly, as a final consequence of our work on Braess's Paradox, we show that a natural network design problem, motivated by the goal of detecting and avoiding Braess's Paradox, is NP-hard to approximate better than $2^{o(n)}$. The network design problem is simply to take any given network and find the subnetwork with the smallest maximum latency at Nash equilibrium. Using our family of two-commodity networks, which show that Braess's Paradox can be as bad as $2^{\Omega(n)}$, and ideas from the gap reductions of [Rou] for the single-commodity version of this problem, we prove that there is no polynomial-time algorithm for this network design problem with $2^{o(n)}$ approximation ratio (assuming $P \neq NP$). Since our upper bound on the price of anarchy trivially implies that an exponential performance guarantee is achievable, our result provides a rare example of a natural optimization problem with intrinsically exponential approximability.

1.2 Linked Decompositions for Internet Routing and Management

In the second chapter of this thesis, we focus our attention on the recent efforts to redesign the Internet from a clean slate to improve the Internet in terms of various measures, such as scalability, reliability, manageability, and performance. As the Internet continues to grow larger and the challenges of scaling up and managing the current Internet infrastructure continue to accumulate, there are several serious efforts underway to redesign the Internet [FI05; GE05; NA; CCK+06]. It is in this spirit that we study the properties of the various random graphs models used to model the Internet, such as the preferential attachment model [BA99], in order to determine if the properties of these "Internet-like" graphs, can be useful for developing new ideas to redesign the Internet. We discover that

these Internet-like graphs can with high probability be decomposed into roughly equal size components, such that every pair of components intersect in at least one node, and our experiments indicate that the real Internet can also be decomposed in such a manner. We call such a decomposition, a *linked decomposition*.

Our motivation for finding a linked decomposition is that it provides a very simple compact routing scheme. A compact routing scheme is a way to store a minimal amount of information at each node, while maintaining the ability of each node to send and route packets to every other node in the network. Our compact routing scheme based on the idea of linked decompositions may be easier to maintain, since the various components of the linked decomposition may manage themselves somewhat independently. Good compact routing schemes also make sure that packets are sent along relatively short paths. In addition to showing that the concept of linked decompositions provides a simple compact routing scheme, we also have experiments and proofs which show that our routing scheme based on linked decompositions sends packets along relatively short paths in the real Internet and in graphs generated by random graph models for the Internet.

In showing that the graphs generated by the preferential attachment model can be decomposed into a linked decomposition, with high probability, where the components are roughly equal size, we need to analyze a new random process, which is a variation on the classic *Polya urns process*, which may be of independent interest. In this new process, which we call *Polya urns with the power of choice*, we start with n nonempty bins containing $O(n)$ balls total, and each arriving ball is placed in the least loaded of m bins, drawn independently at random *with probability proportional to load*. Our analysis shows that in our new process, with high probability the bin loads become roughly balanced some time before $O(n^{2+\epsilon})$ further balls have arrived and stay roughly balanced, regardless of how the initial $O(n)$ balls were distributed, where $\epsilon > 0$ can be arbitrarily small, provided m is large enough.

1.3 An Online Bipartite Matching Problem

In the last chapter of this thesis, we study an online bipartite matching problem related to efficiently providing service over the Internet. The online bipartite matching problem we study can model problems in many settings, including problems in wireless communication, content delivery, and job scheduling, but in the primary motivating sce-

nario, we imagine that we have some servers capable of providing some service, and clients who arrive online one at a time and request service from one of the servers. As each client arrives, she provides a list of servers capable of servicing her request, and the goal of the algorithm is to maintain a matching between clients who have arrived and servers, which respect the requirements provided by the clients. In order to maintain a maximum matching between clients and servers, as clients arrive the algorithm may need to switch one client from being serviced on one server to being serviced on another server. For many applications, switching a client from one server to another server incurs some cost, and so the goal of our algorithm will be to maintain a maximum matching at all times while the clients arrive, while minimizing the total number of times clients are switched from one server to another server.

More formally, in our problem, we have a bipartite graph G between n clients and n servers, which represents the servers to which each client can connect. We will assume that G has a perfect matching, although our results also hold more generally when a perfect matching does not exist. Although the edges of G are unknown at the start, we learn the graph over time in an online manner, as each client arrives and requests to be matched to a server. As each client arrives, she reveals the servers to which she can connect, and the goal of the algorithm is to maintain a matching at all times between the clients who have arrived so far and the servers, while minimizing the switching cost, the total number of times clients need to switch servers.

Although it has been difficult to produce good bounds on the best possible switching cost achievable in the worst case, a worst case analysis may be too pessimistic for the scenarios we typically expect to see in practice. In many settings, it may make sense to make assumptions about the online bipartite matching problem we expect to encounter in practice. For example, it may be reasonable to assume in many settings that the clients arrive in a uniformly random order, or assume that the bipartite graph between clients and servers is a random bipartite graph. Although there are no known algorithms which are guaranteed to yield switching cost better than the trivial $O(n^2)$ in the worst case, we show that the switching cost can be much lower in three natural settings.

In our first result, we show that for any arbitrary graph G with a perfect matching, if the clients arrive in a random order, then the total switching cost is only $O(n \log n)$ with high probability. This bound is tight, as we show an example where the switching cost is $\Omega(n \log n)$ in expectation under this model. In our second result, we show that if each

client has edges to $\Theta(\log n)$ uniformly random servers, then the total switching cost is even better; in this case, it is only $O(n)$ with high probability, and we also have a lower bound of $\Omega(n/\log n)$ for this model as well. In terms of the number of edges needed for each client, our result is also tight, since $\Omega(\log n)$ edges are needed to guarantee a perfect matching in G with high probability. In our last result, we derive the first algorithm known to yield total cost $O(n \log n)$, given that the underlying graph G is a forest. This is the first result known to match an existing lower bound, which shows that any online algorithm must have switching cost $\Omega(n \log n)$, even when G is restricted to be a forest.

1.4 The Methodology Used to Prove a Bound on Braess's Paradox

In Chapter 2, in addition to studying selfish routing and Braess's Paradox, we also attempt to achieve our secondary goal of describing the thought process and methodology we use to prove one new mathematical result. Ideally, we would like to describe our thought process as a concrete algorithm which takes as input some known results and a target result, and outputs a proof or disproof of the target result, or a related result. Although the algorithm we describe could not possibly always succeed in proving or disproving a target result, as the above problem is undecidable, we may be able to explicitly describe a high level heuristic algorithm that mathematicians can use to generate new results. The main challenge is to discover and explicitly describe the high level heuristic algorithms, which we already use implicitly when proving new mathematical results.

Although it may seem easy to analyze one's own thought process and describe the methodology or high level algorithm one uses to produce a new result from some known results, the exercise of analyzing one's own thought process can be difficult, and it can be hard to pin down and describe the knowledge and methodology we use to solve mathematical problems explicitly. Nevertheless, we attempt to describe as best we can, the steps we use to prove a new mathematical result in the next chapter.

Even though our attempt to describe the methodology that we use to solve one mathematical problem may not be perfect, and may not be the most useful methodology, we hope our attempt starts a broader discussion on the various techniques people use to solve mathematical problems. There are perhaps many ways in which new mathematical

results can be derived, but very few papers in mathematics attempt to describe the thought process or high level algorithm which allow the author(s) to start from known results and end at a new theorem. It is in this light that we attempt to provide a concrete description of the steps and methodology used to prove a bound on Braess's Paradox in Chapter 2.

1.5 Bibliographic Notes

The results on selfish routing presented in Chapter 2 originally appeared in two papers [LRT04; LRTW05], which were joint work with Tim Roughgarden, Eva Tardos, and Asher Walkover. The thought process described to prove the first result in Chapter 2 is new, although it is based on previous joint work with Tim Roughgarden and Eva Tardos. The results on linked decompositions presented in Chapter 3 represent joint work with Christos Amanatidis, Richard Karp, Christos Papadimitriou, and Martha Sideri, and first appeared in [AKL$^+$08]. The results on online bipartite matching first appeared in a paper [CDKL] with Kamalika Chaudhuri, Costis Daskalakis, and Robert Kleinberg.

Chapter 2

Braess's Paradox and Efficiency of Selfish Routing

2.1 Background

A recent trend in theoretical computer science is to analyze the overall social welfare experienced by agents at equilibria in a noncooperative game. In the first chapter of this thesis, we study one of the most popular models for analyzing the social welfare in a noncooperative game, the so-called *selfish routing* model. Selfish routing is a mathematical model of how noncooperative agents route traffic in a network with congestion, and has been well-studied in prior work (see [Rou05; Rou07] for a survey of recent work). Formally, the game takes place in a directed single or multicommodity flow network, where each edge possesses a continuous, nondecreasing latency function that models how the performance of an edge degrades as it becomes increasingly congested. The traffic in the network is assumed to comprise a large number of independent network users, so that each individual has negligible impact on the experience of others. Under this assumption, equilibria—*flows at Nash equilibrium*—are naturally defined as the network flows in which all traffic travels only on minimum-latency paths. The selfish routing model assumes that traffic converges to a flow at Nash equilibrium, and when we study selfish routing, we study the properties of flows at Nash equilibrium. In this chapter, we will analyze a phenomenon known as Braess's Paradox and study the price of anarchy of selfish routing, which we describe in greater detail below.

2.1.1 Braess's Paradox in Single-Commodity Networks

In 1968, Braess [Bra68] demonstrated a remarkable and counterintuitive fact, now known as "Braess's Paradox": in a network in which users selfishly and independently choose minimum-latency paths and the latency of an edge increases with the amount of edge congestion, *adding* edges to the network can *increase* the latency encountered by traffic. Braess's Paradox has since motivated a vast number of follow-up papers; see [Rou07] for a survey. Almost all existing work on the paradox confines attention to close variations on or analogues of Braess's original example in a four-node network. Only recently have larger, more severe versions of Braess's Paradox been discovered. Specifically, Roughgarden [Rou] defined an infinite family of networks, beginning with Braess's original example, that shows that adding $\lfloor n/2 \rfloor - 1$ edges to a single-commodity network with n vertices can increase the latency experienced by all of the traffic (and hence the maximum latency) by a factor of $\lfloor n/2 \rfloor$. Furthermore, Roughgarden also proved that Braess's Paradox can increase the latency by at most $\lfloor n/2 \rfloor$ in a graph with n nodes.

However, the "Braess graphs" of [Rou] differ from Braess's example in that $\lfloor n/2 \rfloor - 1$ edges need to be added in order to increase the latency by a factor of $\lfloor n/2 \rfloor$. Although Roughgarden's prior work provides a complete characterization of the severity of Braess's Paradox, when an arbitrary number of edges can be added, it is natural to ask, for $n \geq 6$, can adding a single edge cause an $\lfloor n/2 \rfloor$ increase in latency? When adding a single edge, are there more severe examples than in Braess's original example, which increase the common latency by more than a factor of 2?

In the first result of this thesis, we introduce a simple but powerful combinatorial theorem that resolves these questions in the negative: for any integer $k \geq 1$, adding k edges to a graph increases the latency experienced by traffic by at most $k + 1$. Since the Braess's graphs of [Rou] show that adding k edges can increase the latency by a factor of $k + 1$, our first result provides a tight bound on the increase in latency that can occur due to Braess's Paradox when a fixed number of edges are added to a single-commodity network.

In describing the proof of our first result, we attempt summarize the methodology we use to prove the result, and describe the step-by-step reasoning we used to come up with the final proof. We hope readers find the discussion of the thought process we used to derive the result interesting as well.

2.1.2 Braess's Paradox and Price of Anarchy in Multicommodity Networks

Although our first result provides a complete characterization of the worst possible severity of Braess's Paradox in single-commodity networks, when a fixed number of edges are added to a graph, our result does not have any implications for Braess's Paradox in multicommodity networks. Thus, the story of Braess's Paradox in multicommodity networks was incomplete. Are the networks of [Rou] the worst examples of Braess's Paradox, or are more severe versions of Braess's Paradox lurking in the richer landscape of multicommodity networks?

In this chapter, we also study the severity of Braess's Paradox in multicommodity networks and in addition, we study a related concept known as the *price of anarchy* of selfish routing. The *price of anarchy* [Pap01] (also called the *coordination ratio* [KP99]) measured relative to an objective function, is defined as the worst-case ratio between the objective function value of a solution at Nash equilibrium and the optimal objective value achievable by any solution. When our objective function measures the maximum latency that any flow experiences, this price of anarchy with respect to the maximum latency objective is very related to the worst case severity of Braess's Paradox. In particular, lower bounds on the severity of Braess's Paradox imply lower bounds on the price of anarchy with respect to the maximum latency objective, and upper bounds on the price of anarchy with respect to the maximum latency objective imply upper bounds on the severity of Braess's Paradox. Given this relationship between the price of anarchy with respect to the maximum latency objective and the severity of Braess's Paradox, we now state the new upper and lower bounds we have for both quantities below.

Our Formal Results

In this paper, we establish nearly matching upper and lower bounds on both the price of anarchy with respect to the maximum latency and on the worst-possible severity of Braess's Paradox in multicommodity networks. Our results resolve both of the conjectures in [Rou04]—one in the affirmative, one in the negative—and also give the first demonstration that Braess's Paradox is provably more severe in multicommodity networks than in single-commodity ones. Specifically, our two main results are the following.

- We give a parameterized construction, based on the Fibonacci numbers, that shows

that adding one edge to a two-commodity network with n vertices can increase the latency of all traffic by a $2^{\Omega(n)}$ factor.

- We prove that the price of anarchy with respect to the maximum latency in networks with k commodities, n vertices, and m edges is $2^{O(\min\{kn, m \log n\})}$.

The construction used to prove the first result has wide implications. In particular, for all existing approximation-type analyses of selfish routing that were only known to hold for single-commodity networks [LRT04; Rou; Rou04], this construction rules out any reasonable extension to multicommodity networks, even those with only two commodities. For example, adding one edge to a single-commodity network can only increase the maximum (or average) latency of a Nash flow by a factor of 2 [LRT04], while our construction shows that adding a single edge can cause an exponential increase in the average and the maximum latency (even with only two commodities). This dichotomy between single- and two-commodity networks is somewhat unexpected, given the negligible role that the number of commodities has played in previous work in this area [CSM04b; Rou02b; RT02].

The first result easily implies a lower bound of $2^{\Omega(n)}$ on the price of anarchy for the maximum latency in multicommodity networks, as an optimal flow has the option of ignoring edges that are causing Braess's Paradox. By the same reasoning, the second result implies that adding any number of edges to a network with k commodities, n vertices, and m edges can only increase the maximum latency by a $2^{O(\min\{kn, m \log n\})}$ factor. Our upper and lower bounds on both the price of anarchy and on the worst-possible severity of Braess's Paradox are thus essentially tight for networks with a constant number of commodities.

Finally, we consider the following network design problem, motivated by the goal of detecting and avoiding Braess's Paradox: given a network, find the subnetwork with the smallest maximum latency. Using our family of two-commodity networks and ideas from the gap reductions of [Rou] that apply to the single-commodity version of the problem, we prove that there is no polynomial-time algorithm for this network design problem with subexponential approximation ratio (assuming $P \neq NP$). Since our upper bound on the price of anarchy trivially implies that an exponential performance guarantee is achievable, this network design problem is a rare example of a natural optimization problem with intrinsically exponential approximability.

Related Work on the Price of Anarchy

The price of anarchy of selfish routing, measured relative to the average latency incurred by all traffic, has been extensively studied. Beginning with Roughgarden and Tardos [RT02] and continuing with studies of ever-increasing generality [CS03; CSM04b; Per04; Rou02b; RT04], exact worst-case bounds on the price of anarchy with respect to the average latency have been established under a wide variety of different assumptions.

Although the price of anarchy relative to the average latency objective has been well-studied, permitting an objective function to average the cost of different users can be problematic from a fairness perspective. Specifically, to attain or approximate a flow that minimizes the average latency, some users may need to be sacrificed to very costly paths, in order to reduce the congestion encountered by others (see e.g. [JMS00; Rou02a]). This unfairness inherent in the average latency measure motivates modifying the objective function to be more attuned to those users on the most costly paths. Arguably, the most obvious way to accomplish this is to aspire toward minimizing the *maximum* latency incurred by any user.

Compared to the average latency objective, considerably less is known about the price of anarchy relative to the maximum latency. The first paper on the topic is by Weitz [Wei01], whose results we will review below. Most relevant for us is a paper by Roughgarden [Rou04], where only the special case of single-commodity networks, networks in which all traffic shares the same source and destination, were considered. The main result of [Rou04] states that, if latency functions are allowed to be arbitrary continuous, nondecreasing functions, then the (worst-case) price of anarchy with respect to the maximum latency objective in single-commodity networks with at most n vertices is precisely $n - 1$.

Roughgarden [Rou04] also made two conjectures about this price of anarchy in multicommodity networks. The *weak conjecture* of [Rou04] asserts that in multicommodity networks, this price of anarchy can be bounded by a function of the number of vertices, edges, and commodities in the network. As a point of contrast, simple examples show that no such bound is possible for the price of anarchy relative to the average latency, unless additional structure is imposed on the network latency functions [RT02]. The *strong conjecture* of [Rou04] states that the price of anarchy with respect to the maximum latency remains $n - 1$ in multicommodity networks. This conjecture was motivated in part by the provable equivalence of single-commodity and multicommodity networks for the price of

anarchy relative to the average latency [CSM04b; Rou02b]. Our work confirms the weak conjecture and disproves the strong conjecture.

As noted above, Weitz [Wei01] was the first to study the price of anarchy of selfish routing under the maximum latency objective. Weitz [Wei01] noted that, for single-commodity networks and classes of restricted latency functions, the price of anarchy for the maximum latency is no more than that for the average latency objective. For example, a theorem of Roughgarden and Tardos [RT02] bounding the price of anarchy with respect to the average latency objective then implies that the price of anarchy for the maximum latency in single-commodity networks with linear latency functions is at most 4/3, and a matching lower bound is furnished by the original form of Braess's Paradox [Bra68; Wei01]. However, upper bounds on the price of anarchy with respect to the maximum latency objective do not imply upper bounds on the price of anarchy with respect to the average latency objective: for example, the price of anarchy for the maximum latency objective is at most $n - 1$ in single-commodity networks with arbitrary latency functions [Rou04], while the price of anarchy for the average latency can be arbitrarily large even in two-node, two-link networks [RT02].

Weitz [Wei01] also gave a family of networks that shows that this price of anarchy is $\Omega(n)$ for multicommodity networks with n vertices and linear latency functions. Concurrently with Roughgarden [Rou04], Correa, Schulz, and Stier Moses [CSM04a] studied the maximum latency objective from several different perspectives. The results of [CSM04a] mostly concern the computational complexity of computing an optimal solution and the extent to which multiple objective functions can be simultaneously optimized, and are disjoint from those in [Rou04] and in the present work.

There have also been numerous price of anarchy analyses in many other settings in the past few years. Study of the original load-balancing model of Koutsoupias and Papadimitriou [KP99] continues unabated; see [Czuar; FGL+03] for surveys. A survey of the selfish routing model studied here, including results on the price of anarchy, can be found in [Rou05]. Other noncooperative games have also been studied recently from a price of anarchy perspective, including facility location games [DGK+04; Vet02], network design games [ADTW03; FLM+03], and resource allocation games [JT04].

2.2 Preliminaries

The Model We now describe our model of selfish routing, following Roughgarden and Tardos [RT02]. We will study a flow network, described by a directed graph $G = (V, E)$ and k source-destination vertex pairs $(s_1, t_1), \ldots, (s_k, t_k)$. If $k = 1$ we refer to the instance as a single-commodity instance, and if $k > 1$ we refer to the instance as a multicommodity instance. In single-commodity instances, we may drop the subscripts and simply refer to the source node as s and the destination node as t. We denote by r_i the amount of traffic that wishes to travel from the source s_i to the destination t_i—the *traffic rate*. The graph G can contain parallel edges, but we can exclude self-loops. We will denote the s_i-t_i paths of G by $\mathcal{P}_i$. We assume that $\mathcal{P}_i$ is non-empty for all i, and define $\mathcal{P} = \cup_{i=1}^{k} \mathcal{P}_i$.

A *flow* is a nonnegative vector indexed by $\mathcal{P}$. By f_e we mean the amount $\sum_{P \in \mathcal{P} : e \in P} f_P$ of flow that traverses edge e. With respect to a network G and a vector r of traffic rates, a flow is *feasible* if $\sum_{P \in \mathcal{P}_i} f_P = r_i$ for all commodities i.

We assume that the network G suffers from congestion effects, and to model this we give edge e a nonnegative, continuous, nondecreasing *latency function* ℓ_e that describes the time needed to traverse the edge as a function of the edge congestion f_e. Given a flow f, the latency ℓ_P of a path P is the sum of the latencies of the edges in the path: $\ell_P(f) = \sum_{e \in P} \ell_e(f_e)$. We will call a triple of the form (G, r, ℓ) an *instance*.

Flows at Nash Equilibrium We next define the flows that we expect to arise from selfish routing. Assuming that all network users have negligible size and want to minimize the latency experienced, we expect all users to travel on paths with minimum-possible latency. We formalize this in the next definition.

Definition 1 *A flow f feasible for (G, r, ℓ) is* at Nash equilibrium*, or is a* Nash flow*, if for every $i \in \{1, 2, \ldots, k\}$ and two paths $P_1, P_2 \in \mathcal{P}_i$ with $f_{P_1} > 0$,*

$$\ell_{P_1}(f) \leq \ell_{P_2}(f).$$

Definition 1 implies that in a Nash flow, all of the traffic of a given commodity experiences a common latency. Happily, Nash flows must always exist, and although Nash flows are not necessarily unique, for any two Nash flows of an instance, the common latency incurred by each commodity will be the same. More formally, if we define $L_i(f)$ to be the common latency of the ith commodity's traffic in a Nash flow for f for (G, r, ℓ), then the following proposition holds:

Proposition 2 *Let (G, r, ℓ) be an instance.*

(a) There is at least one Nash flow for (G, r, ℓ).

(b) If $f, \tilde{f}$ are Nash flows for (G, r, ℓ), then $L_i(f) = L_i(\tilde{f})$ for all commodities i.

Proposition 2 is classical; for example, it follows from arguments of Beckmann, McGuire, and Winsten [BMW56]. Since each Nash flow results in the same common latency for each commodity, we will sometimes use $L_i(G, r, \ell)$ to denote be the common latency of the ith commodity's traffic in a Nash flow for (G, r, ℓ).

The maximum latency objective, which we seek to minimize, is formally defined as $M(f) = \max_{P \in \mathcal{P}:\, f_P > 0} \ell_P(f)$ for a flow f. Note that Proposition 2(b) implies that all Nash flows for an instance (G, r, ℓ) have the same maximum latency objective value. With respect to an instance (G, r, ℓ), a flow that minimizes $M(\cdot)$ over all feasible flows will be called *optimal*. Since the feasible flows of an instance form a compact subset of Euclidean space and $M(\cdot)$ is a continuous function, every instance admits an optimal flow.

To provide a concrete example illustration of the selfish routing model, and concept of flows at Nash equilibrium, consider the networks discovered by Braess in 1968 [Bra68], shown in Figures 2.1 and 2.2. In the first figure without the added edge, we have one unit of flow which seeks to travel from s to t, and at Nash equilibrium, the flow divides evenly between the two paths, incurring 1.5 units of latency along each path. In the second figure with the added edge, all flow must converge to the s-u-v-t path at Nash equilibrium, and all flow experiences 2 units of latency. This illustrates the counterintuitive phenomenon that adding an extra edge to a network can actually increase the latency incurred by all users. Note that this example can also be used to show that adding a single edge can increase the common latency by a factor that is arbitrarily close to 2, if we replace the edges with the $\ell(x) = x$ latency function by edges with the latency function $\ell(x) = x^d$ for some sufficiently high exponent d.

We will also benefit from the following alternative definition of a Nash flow, which was first noted by Smith [Smi79]. It is an easy consequence of Definition 1.

Proposition 3 *A flow f feasible for (G, r, ℓ) is at Nash equilibrium if and only if*

$$\sum_{e \in E} \ell_e(f_e) f_e \leq \sum_{e \in E} \ell_e(f_e) \tilde{f}_e \tag{2.1}$$

for every flow $\tilde{f}$ that is feasible for (G, r, ℓ).

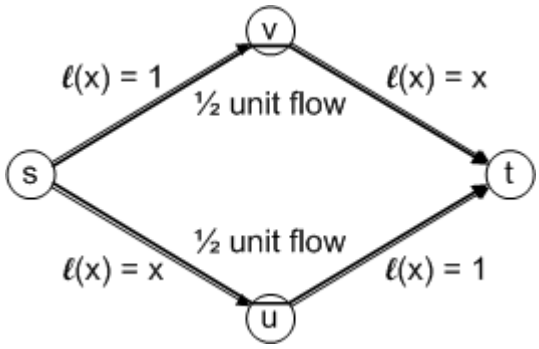

Figure 2.1: Braess's Paradox Example: Traffic experiences 1.5 units of latency at Nash equilibrium initially.

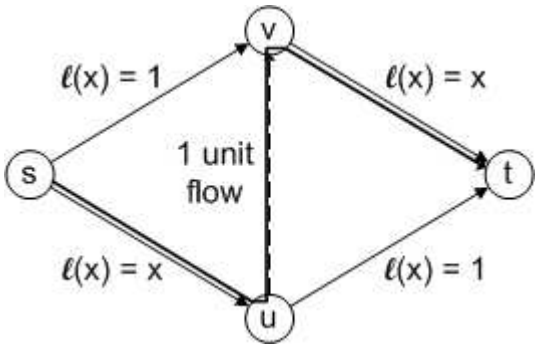

Figure 2.2: Braess's Paradox Example: Traffic experiences 2 units of latency at Nash equilibrium after adding a zero latency edge.

The Price of Anarchy We now formalize what we mean by the price of anarchy. As noted in the introduction, it is the ratio of the objective function values of a flow at Nash equilibrium and an optimal flow. If (G, r, ℓ) is an instance, then the *price of anarchy of* (G, r, ℓ), denoted $\rho(G, r, \ell)$, is the ratio $M(f)/M(f^*)$, where f is a Nash flow and f^* is an optimal flow. Proposition 2 ensures that the price of anarchy of an instance is well defined provided $M(f^*) > 0$. If $M(f^*) = 0$, then f^* is also a flow at Nash equilibrium and we define the price of anarchy of the instance to be 1.

Finally, the price of anarchy $\rho(\mathcal{I})$ of a collection $\mathcal{I}$ of instances is defined in the obvious way:

$$\rho(\mathcal{I}) = \sup_{(G,r,\ell)\in\mathcal{I}} \rho(G, r, \ell).$$

2.3 Bounding Braess's Paradox in Single-Commodity Networks

In this section, we attempt to summarize and explicitly describe the thought process that we use to prove that adding k edges to a single-commodity network increases the common latency by at most a factor of $k + 1$. The process can be summarized by the following high level steps.

1. Read and understand the existing literature

2. Formulate a target theorem to prove or disprove

3. Simplify the target theorem to a statement S that is as specific as possible but is not currently known to be true or false

4. Attempt to prove or disprove S by repeating the following steps:

   (a) Generate specific examples to examine, starting with the simplest examples first (or use examples existing in the previous literature)

   (b) Analyze those examples and attempt to generate observations that summarize the common behavior occurring in the examples generated

   (c) See if any of the analysis used to study the specific examples can be used more generally or inductively to generate an conclusion which covers a larger and/or infinite class of instances

5. If you manage to prove S, repeat step 3 with a more general statement, or if not, attempt restrict the problem in other ways and repeat the process with a different statement S.

Beginning at step 2, let us assume that the target theorem we would like to prove is that adding k edges to a graph increases the latency by at most $k + 1$. Following step 3, let us simplify the target theorem to the more modest goal of proving that adding a single edge with zero latency increases the common s to t latency by at most a factor of two. We will later consider the case when multiple edges are added, and when they are not zero latency edges. As we describe our thought process, we omit the various lines of thought that did not yield any useful conclusions, and for the sake of presentation, we may present some of our discoveries out of order.

2.3.1 Zero Latency Paths

To start step 4 and gain intuition for the problem, let us consider the simplest graph in which Braess's Paradox might occur: a two node network consisting only of two nodes s and t and one edge between them. In this network, we can only add an edge from s to t, and clearly Braess's Paradox cannot occur here, as adding a zero latency edge from s to t automatically reduces the Nash equilibrium latency to zero, the lowest possible latency. Although this example is too trivial for Braess's Paradox to occur, we have already proved something new in showing Braess's Paradox cannot occur in a two node network.

As we continue our analysis, let us consider the next simplest example: a network with three nodes s, u, and t, and two edges (s, u) and (u, t), forming a path from s to t. Here there are three possibilities for placing a new edge. Notice again that if we add an edge from s to t, we will zero out the latency from s to t, and again Braess's Paradox cannot occur. Although this example is also very trivial, it is important to note that we have a common explanation as to why Braess's Paradox cannot occur in both of the first two examples we examined: whenever we add a zero latency edge from s to t, the s to t latency automatically becomes zero, and thus Braess's Paradox cannot occur.

Moreover, it is not hard to see that the observation we made above holds not only for the two examples we examined above, but it also holds for all graphs. Namely, given any graph, Braess's Paradox can never occur if a zero latency edge is added from s to t. Although we have only looked at two examples, we have found a statement that holds for all graphs. Even though our reasoning is trivial at this point, we mention it explicitly to describe the entire thought process and because it illustrates a common theme that occurs frequently in the thought process we use to prove a general bound on Braess's Paradox. To discover a general theorem, we always study finite examples, but we continuously seek to generate observations that can be generalized to say something about an infinitely large number of objects. So far, we have already generated an observation that covers an infinite number of graphs, but it is not general enough to cover all possible instances of Braess's Paradox that may occur. As our thought process proceeds, we will look to find more and more general observations until we have a general statement that covers all possible instances of Braess's Paradox.

Remark

Even though we have not done much at this point, upon reflection one may find it remarkable that we were somehow able to examine a finite number examples, and obtain an observation that holds for an infinite number of objects. It takes some thought to realize that even when we came to the simple conclusion that our observation holds for all graphs, we must have utilized some form of inductive reasoning, even if we may not have realized it. Upon further reflection, one way we may been able to come to our new conclusion was to notice that as a base case our statement on adding an zero latency edge from s to t holds for the smallest example with two nodes, and then after seeing our second example with

three nodes, we implicitly realize by induction that the reasoning of our statement does not change no matter how many more nodes or edges we have in our initial graph.

Although we may be overly pedantic in pointing out the subtle reasoning that we use implicitly to obtain a statement that holds for an infinite number of objects after examining only a finite number of examples, we feel it is important to recognize these subtle steps in describing the complex thought process we utilize in proving mathematical theorems. Recognizing the subtle and implicit steps we use in our reasoning can be the key to understanding the way we prove new mathematical theorems, and can help make the knowledge we use to prove new results explicit and teachable rather than being implicit and tacit. Although we do our best to recognize the subtle steps we use in our thought process, we may fall short of this goal, as the reasoning process is sometimes complex and very nuanced. Nevertheless, let us do our best in describing the thought process and continue analyzing some more examples in the hopes of proving a more general theorem.

2.3.2 Light Paths

If we continue on this path of exploring new examples, starting with the simplest examples first, it would be natural to explore the same three node example with the two edges (s, u) and (u, t) as before, and consider the two other cases of adding a new zero latency edge from s to u or from u to t. It is not too difficult to argue that the overall latency improves in these two cases as well and again Braess's Paradox does not occur. We omit a formal proof, but Braess's Paradox cannot occur here essentially because the latency between s and u and the latency between u and t must improve in both cases, and thus the overall s to t latency must improve as well. For conciseness, when we say that the latency must improve, we mean that the latency must stay the same or decrease. For future reference, we will be precise in saying that the latency *strictly* improves, if we mean to say that the latency *strictly* decreases. It may not immediately be clear whether or not we can draw a more general conclusion from these two examples yet, but let us continue analyzing some more complicated examples.

There is one more three node example with the two edges (s, u) and (s, t) which we could consider here, but to move our exposition forward, let us skip ahead and consider a more complicated example with four nodes s, u, v, and t, and three edges (s, v), (s, u), and (u, t). Now let us suppose that a new zero latency edge is added from v to t. Can

Braess's Paradox occur here as well? Note that there is only one s to t path from s to u to t before our (v, t) edge is added, and after the edge is added, there is only one additional s to t path from s to v to t. If there is no new additional flow along our new $s - v - t$ path after we add the v to t edge, then clearly Braess's Paradox cannot occur as the Nash equilibrium flow remains the same both before and after adding the new edge. Furthermore, if there is additional flow on the $s - v - t$ path after we add the v to t edge, then there must be less flow on our original $s - u - t$ path, and as a result, the latency along the s to u edge and the u to t edge must improve since our latency functions are nondecreasing. Therefore, the overall s to t latency must improve here as well, and Braess's Paradox does not occur in this example also.

Now that we have seen that Braess's Paradox cannot occur in a few more examples, we should see if there is some common explanation that serves to explain why Braess's Paradox does not occur in the previous examples we studied. If we think about our last example, we notice that Braess's Paradox does not occur because no matter what happens, there is always a path from s to t which consists only of edges, which have some flow on them before the new edge is added, and the flow along these edges stay the same or decrease after the new edge is added. As a result, we can conclude that the latency along those edges improve, and the overall latency from s to t must improve as well.

Now to see if this explanation generalizes, for notational purposes, let us call an edge *light* if it has the same amount of flow or less after the new edge is added. For simplicity, let us assume without loss of generality that our instances are preprocessed to remove all edges with zero flow on them both before and after the new edge is added, so that we can assume all light edges have some nonzero amount of flow on them before the new edge is added. Let us also define a *light path* to be a path consisting only of light edges, and let us see if we can state our previous argument more generally. To summarize our previous argument in more formal and general terms, we first observe that there is always a light path from s to t in our example, and then we note that there is flow along the light path before the new edge is added, so that the latency along the light path before the new edge is added is precisely the overall s to t latency experienced by the Nash flow before the new edge is added. After adding the new edge, the latency along the light path must improve, so that we have a path with lower s to t latency than the previous s to t latency experienced at Nash equilibrium. This implies that the new s to t latency at Nash equilibrium must be same or better than the previous s to t latency at Nash equilibrium, and thus Braess's

Paradox cannot occur.

Notice that this explanation is a very general explanation, which can also be applied to the previous examples we studied with two or three nodes to explain why Braess's Paradox does not occur in those examples as well. Also notice that by phrasing our argument in terms of light paths, we have a very general observation which states that whenever there is a light path, Braess's Paradox cannot occur. It is not always the case that our graph will have a light path, but as we continue to study examples, we hope to find a more general observation that covers all instances and bounds the worst case severity of Braess's Paradox in any graph.

2.3.3 Light-Heavy Alternating Paths

As we have proven that the overall latency must decrease if we have a light path from s to t, we now move towards studying more complicated examples, which do not have a light s to t path, in order to generalize our results. Clearly, such examples must exist if Braess's Paradox is to occur, and indeed, if we look back at the Braess's original example of Braess's Paradox, we see that every path from s to t contains an edge with strictly more flow along it. Let us call these edges with strictly more flow on them *heavy* edges, except for the new edge we added, which we will refer to exclusively as an *added* edge. Despite the existence of heavy edges blocking our s to t paths, we might wonder if we can still bound any potential increase in latency. Or better yet, can the existence of heavy edges be used to our advantage in bounding any increase in latency?

Let us study a variant of Braess's original example, which can illustrate the potential use of heavy edges in bounding any increase in latency. Imagine an example with four nodes s, u, v, and t, and five edges (s, u), (s, v), (u, v), (u, t) and (v, t), and let us assume that another edge is added from (v, t), as shown in Figures **??** and **??**. It could be possible that the flows change in this graph, from sending flow along the s-u-t and s-v-t paths to sending flow only on the two s-u-v-t paths. In this case, the (s, u), (u, v), and (v, t) edges all become heavy, and the (s, u) and (u, t) edges become light. As a result, there is no light s to t path, but perhaps we can find some alternative way to bound any potential increase in latency.

First note that in this example, after the new edge is added there is more flow traveling from s to u to v, despite the fact that the latency from s to v improved, as (s, v) is

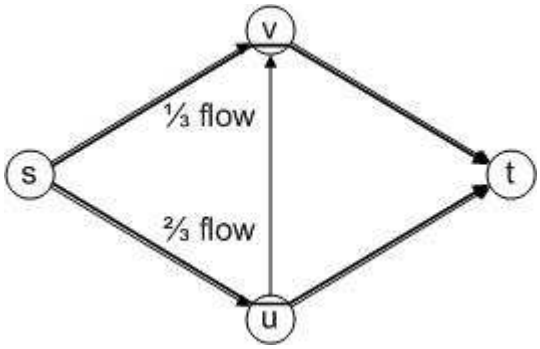

Figure 2.3: A possible Braess's Paradox instance before edge is added.

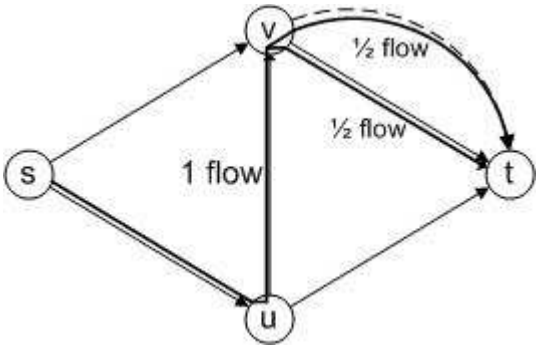

Figure 2.4: A possible Braess's Paradox instance after edge is added.

a light edge. Moreover, the resulting flow is using the s-u-v path despite the fact that (u, v) is a heavy edge and the latency between u and v got worse. Given these two observations, can we make any conclusions regarding the latency along the (s, u) edge? If the latency along (s, v) has improved, while the latency along (u, v) has gotten worse, then the latency from (s, u) must have improved (or stayed the same) or else we would never consider using the s-u-v path instead of the s-v path. Even though (s, u) is a heavy edge, it is like a light edge, in the sense that its latency must improve, or in this case, it must stay exactly the same. As a result, we can think of the path from (s, u) and (u, t) as if it is a light path, and conclude that the overall s to t latency must decrease.

Now, in this example we found that a path from s to t traversing light edges forward and heavy edges backward could be very useful in proving the latency does not increase. Let us see if we can generalize this result to cover any path that traverses light edges forward and heavy edges backward. Let us define a *light-heavy alternating* path to be one that traverses light edges forward and heavy edges backward. We may wonder if an s to t light-heavy alternating path always implies that the s to t latency decreases. It turns out that we can prove such a result, if we borrow some ideas from Roughgarden's previous bound on Braess's Paradox [Rou]. In [Rou], Roughgarden labels each node with its shortest path distance from s with respect to the latency of edges at Nash equilibrium and uses those distances to prove a bound on Braess's Paradox.

Let us see if we can use distance labels to help with our analysis as well. Let us first define $l(u, v)$ to be the latency along edge (u, v) at Nash equilibrium before the new edge is added, and define $l'(u, v)$ to be the latency of the edge at Nash equilibrium after

the new edge is added. (Our notation may be slightly ambiguous when there is more than one edge between u and v, but it should be clear from context, to which edge the latencies $l(u, v)$ and $l'(u, v)$ refer). For each node v, let us define $d(v)$ to be the length of the shortest path from s to v measured with respect to the latencies at Nash equilibrium before the new edge is added, and define $d'(v)$ to be length of the shortest path from s to v measured with respect to the latencies at Nash equilibrium after the edge is added.

Using this idea, let us see if we can come up with a more general explanation which can prove the s-t latency cannot increase even if we have an arbitrarily long light-heavy alternating path. If we follow the analysis we used on our specific example with four nodes, we essentially showed that the latency from s to each node v along the light-heavy alternating path improved, or in other words, $d'(v) \leq d(v)$ for all nodes v in the light-heavy alternating path, until we reached t, and concluded that $d'(t) \leq d(t)$. Now, if we follow our prior analysis, we were able to prove that $d'(v) \leq d(v)$ for each node v along the light-heavy alternating path in sequence, both when our path crossed a light edge forward and a heavy edge backward. If we write things down formally, it is not hard to see that our previous analysis already has essentially everything we need to prove inductively that $d'(v) \leq d(v)$ along the light-heavy alternating path.

To prove our claim, we know that $d'(s) = d(s) = 0$ as a base case, and we need to show that the invariant $d'(v) \leq d(v)$ is preserved along the light-heavy alternating path as we traverse light edges forward and heavy edges backward. Now, let us assume that we are at a node v about to cross a light edge (v, w), and we would like to prove $d'(w) \leq d(w)$. We can assume by induction that $d'(v) \leq d(v)$ and by playing around with some inequalities we know must be true, we can prove the following chain of inequalities holds: $d'(w) \leq d'(v) + l'(v, w) \leq d(v) + l(v, w) = d(w)$. The first inequality holds because d' measures shortest path distances after the new edge is added, and the second inequality holds because (v, w) is a light edge. The last equality holds because there is flow along the (v, w) edge before the new edge is added and at Nash equilibrium, this flow must travel along shortest paths.

Now let us consider the case when we are at a node v and cross a heavy (w, v) edge backward. We can use a similar analysis to prove the following chain of inequalities $d'(w) + l'(w, v) = d'(v) \leq d(v) \leq d(w) + l(w, v)$. The first equality holds because there is flow on (w, v) after the edge is added and flow only travels along shortest paths. The second inequality holds by the inductive hypothesis, and the third inequality holds because d is

defined as the shortest path distance from s to nodes in the graph. Lastly, we can conclude that $d'(w) \leq d(w)$, since (w, v) is a heavy edge. Thus, we have proved the inductive step, and as a result, we know that if there is ever an s to t light-heavy alternating path, then $d'(w) \leq d(w)$ and Braess's Paradox does not occur.

2.3.4 General Alternating Paths with One Added Edge

Now that we have proven that Braess's Paradox cannot occur if there is a light-heavy alternating path from s to t, one might wonder when exactly can Braess's Paradox occur, and can we bound its increase when it does occur? Let us now consider Braess's original example with four nodes, four edges, and one added edge, as shown in Figures 2.1 and 2.2. In this case, even though we do not have a light-heavy alternating path as we have defined it, we do have a path from s to t that traverses only light edges forward, and heavy or added edges backward, along the s-v-u-t path. Let us call a path that only traverses light edges forward, and heavy or added edges backward, a *general alternating path*. Without assuming any other knowledge about our instance, we may wonder if we can prove a bound on the increase in latency that occurs in this four node graph, only based on the fact that there exists a general alternating path along the nodes s-v-u-t, traversing a light edge forward, an added edge backward, and another light edge forward. Let us see if we can use some of the same techniques as before to bound any increase in the s to t latency.

Note that we can still conclude that the distance from s to v decreases as there is a light edge from s to v, but we can no longer conclude that the distance from s to u has decreased because the (u, v) edge is now an added edge rather than a heavy edge. Previously, we could conclude that the distance from s to u decreased because the heavy (u, v) edge is present in the original graph, and therefore $d(v) \leq d(u) + l(u, v)$. We can no longer utilize this inequality when (u, v) is an added edge, but we can still utilize the first steps of our chain of inequalities to conclude that $d'(u) + l'(u, v) \leq d'(v) \leq d(v)$. Now since v is adjacent to a light edge, we know there is flow going to v in the original graph, so that we can conclude $d(v) \leq d(t)$ and then $d'(v) \leq d(t)$. Now continuing along the general alternating path to node t, it is easy to see that the following chain of inequalities holds $d'(t) \leq d'(u) + l'(u, t) \leq d'(u) + l(u, t) \leq d(t) + d(t) \leq 2 \cdot d(t)$, so that the latency increases by at most a factor of two. The first inequality holds because d' measures shortest path

distances, and the second inequality holds because (u, t) is a light edge. The third inequality holds because of our previous reasoning, and because the (u, t) edge previously had flow along it.

Thus, we have proven a fairly general statement that the latency can increase by at most a factor of 2 whenever there is a s-v-u-t general alternating path consisting of a light edge, an added edge, and a light edge. Let us try to generalize our reasoning further to prove the latency cannot increase much even if we have a general alternating path of arbitrary length containing an added edge. For reasons that will become clear later, let us define a *backwards segment* of our general alternating path P to be a maximal subpath of P consisting of heavy or added edges, and let us assume that our added edge lies on a backwards segment that the general alternating path starts crossing at node some node v and ends at some node u. (This backwards segment consists of heavy edges and an added edge, which form a directed path from node u to node v). Note that our previous reasoning can be used to show that $d'(w) \leq d(w)$ for all nodes w on the general alternating path before v occurs.

Now, when we reach v, let us follow our previous reasoning to see if we can bound the increase in latency that can occur for node u. Note we can obtain the same sequence of inequalities $d'(u) + l'(u, v) \leq d'(v) \leq d(v) \leq d(t)$ as before, where the last step holds since v previously had flow traveling to it. (We know v is adjacent to a light edge based on our definition of backwards segment). Now inequalities above imply that $d'(u) \leq d(u) + d(t)$, or putting things another way, we can conclude that the latency from s to u increases by at most $d(t)$. Furthermore, as we cross light edges forward and heavy edges backward, we can more or less use the same analysis as before to prove by induction $d'(w) \leq d(w) + d(t)$ for all nodes w that occur after the backwards segment with the added egde. Moreover, when we reach t, we know that $d'(t) \leq d(t) + d(t) \leq 2 \cdot d(t)$. Therefore, we have just shown that whenever we have a general alternating path consisting of at most one added edge, the latency increases by at most a factor of two.

Now that we have proven a good bound on the increase in latency whenever we have general alternating path, we may wonder if a general alternating path always exists in our instances. If we can prove a general alternating path always exists, then we can conclude Braess's Paradox increases latency by at most a factor of 2 when adding a single edge. Luckily it is not too hard to prove a general alternating path always exists, although the methodology we use to prove it is a bit different from the main methodology described

above and tangential to our main proof. We describe the proof below nonetheless.

2.3.5 Existence of General Alternating Paths

It turns out that a general alternating path from s to t must always exist, since adding a new edge (or multiple new edges) only causes the existing flow to shift around without changing the total amount of flow traveling from s to t. To describe the methodology we use to prove an alternating path always exists, we merely borrow some ideas from an existing proof on the related topic of maximum flows, and see if they can be applied to our new problem. The ideas we borrow are from the proof which shows that the Ford-Fulkerson algorithm always terminates with a maximum flow, by finding a corresponding minimum cut. Recall that the proof first notes that when the Ford-Fulkerson algorithm terminates, there is no s to t path left in the residual graph. The argument then proceeds by looking at the set S of nodes reachable from s in the residual graph, and proving the amount of flow crossing the cut between S and $V - S$ is the same as the value of the cut. Now let us see if any of these ideas here can be used to prove that there must always be a s to t general alternating path.

Similarly, for our problem, let us define S to be the set of nodes reachable from s via paths which only traverse light edges forward and heavy or added edges backward. Is it possible that the set S does not include t? Let us prove that this cannot happen by looking at what happens in the cut between S and $V - S$. Note that all edges leaving S must be heavy or added edges, and all edges entering S must be light edges, or otherwise S would not represent the set of all nodes reachable by an alternating path from s. Moreover, there must be at least one edge present in our original graph leaving S, since there must be at least one s to t path in our original graph in order to have a valid s to t flow. This implies that there must be at least one heavy edge leaving S, and as a result, there must be strictly more flow crossing this s-t cut after adding the new edge. However, it clearly cannot be the case that there is more flow crossing this s-t cut, as it would imply that there is more flow traveling from s to t after the new edge is adding. Therefore, we have a contradiction, and t must be reachable from s via a general alternating path.

2.3.6 General Alternating Paths with Multiple Added Edges

We have now reached our goal of proving that adding a single zero edge can increase the s to t latency by at most a factor of two, but it is tempting to see if we can generalize our results. Following step 5, let us generalize our target statement S, and see if we can prove anything when more than one edge is added. Let us see if we can prove a bound on Braess's Paradox when $k > 1$ zero latency edges are added to our graph. We know that the only way that Braess's Paradox can increase latency by more than a factor of two is if the general alternating path crosses more than one newly added edge, but let us see if we can still bound the increase in latency, even if the general alternating path crosses more than one added edge.

To get an idea about proving a bound when multiple edges are added, let us look at Roughgarden's example in [Rou] which shows that adding two extra edges can increase the s-t latency by 3. We see that in Roughgarden's example, the general s-t alternating path does indeed cross two added edges. Along the alternating path, we observe that the s to v distance does not increase for a node v that occurs before crossing the first added edge, and it increases by $d(t)$ for a node that occurs on the general alternating path after crossing the first added edge. Subsequently, after the crossing the second added edge, the distance for a node v along the general alternating path can increase by $2 \cdot d(t)$. Additionally, when we look at more severe examples of Braess's Paradox in [Rou], we observe that in general, if a node v occurs in a general alternating path after crossing c added edges, its distance from s may increase by $c \cdot d(t)$. Given this information, we may hypothesize that this is the worst case, and attempt to prove it by induction.

Let us try to prove by induction that if a node v occurs in a general alternating path after crossing c added edges, then $d'(v) \leq d(v) + c \cdot d(t)$, although in this statement, let us not include the nodes v which occur within a backwards segment with added edge(s), as these nodes need to be handled differently. If we want to prove this hypothesis by induction along the general alternating path, it is not hard to see that our previous analysis can be used to prove the inductive step along the light edges we cross forward and heavy edges we cross backward, as long the heavy edge is not on a backwards segment with one or more added edges. The only challenge is to show that our inductive hypothesis still holds when crossing a backwards segment with one or more added edges.

Let us focus on this case, and assume that we are crossing a backwards segment

with one or more added edges, which starts at node v and ends at node u. This backwards segment consists of heavy and added edges forming a directed path from node u to node v. We seek to prove that if the general alternating path crosses c added edges previously before reaching v, then $d'(u) \leq d(u) + (c+1) \cdot d(t)$. Now, it is not hard to show the following chain of inequalities holds: $d'(u) \leq d'(v) \leq d(v) + c \cdot d(t) \leq d(t) + c \cdot d(t)$. The first equality holds because we can assume without loss of generality that all added edges have flow on them, and thus, we can conclude there is flow traveling from u to v in our graph after the new edges are added. The second inequality holds by the inductive hypothesis, and the third inequality holds because v must be adjacent to a light edge, which implies that $d(v) \leq d(t)$. As a result, we have that $d'(u) \leq d(u) + (c+1) \cdot d(t)$ and we have proved the inductive step when crossing a backwards segment with an added edge. As our previous analysis can also show that the inductive step holds on all other portions of our general alternating path, we have proved our original hypothesis inductively.

Furthermore, our newly proved hypothesis implies that when k new edges are added to our graph, we can conclude that $d'(t) \leq (k+1) \cdot d(t)$, or in other words, the s-t latency increases by at most a factor of $k+1$. Therefore, we have reached the goal of proving our target statement again in showing that the latency increases by at most $k+1$ when adding k zero latency edges. Lastly, let us see if we can prove our original target theorem, which does not assume that the edges we add are zero latency. Note that none of the previous analysis we used in bounding the distances on nodes along the general alternating path relied on the fact that the edges we added have zero latency. Therefore, we have met our original goal of proving that the latency increases by at most $k+1$ when k edges with arbitrary latency functions are added to our graph.

2.4 Selfish Routing Results in Multicommodity Networks

2.4.1 Braess's Paradox in Multicommodity Networks

In this section, we prove that Braess's Paradox can be much more severe in multicommodity networks than in single-commodity networks. In fact, there will be a "phase transition" of sorts: the worst-case severity of Braess's Paradox is polynomial in single-commodity instances, but exponential in two-commodity instances. The family of instances that we construct in this section will also serve as a starting point for our inapproximability

results in Section 2.4.3.

We will begin this section by formally stating the properties of our construction in Theorem 4 below. Prior to detailing this construction and proving Theorem 4, we will discuss its many consequences for multicommodity networks.

Our family of two-commodity instances is closely related to the *Fibonacci numbers*. Recall that for a nonnegative integer p, the pth Fibonacci number F_p is defined as follows: $F_0 = 0$, $F_1 = 1$, and $F_p = F_{p-2} + F_{p-1}$ for $p \geq 2$. It is well known that $F_p \approx c \cdot \phi^p$ as $p \to \infty$, where $c \approx 0.4472$ and $\phi \approx 1.618$ is the golden ratio.

We can now state the main result of this section.

Theorem 4 *There is an infinite family $\{(G^p, r^p, \ell^p)\}_{p=1}^{\infty}$ of instances with the following properties:*

- *(G^p, r^p, ℓ^p) has two commodities and $O(p)$ vertices and edges as $p \to \infty$;*
- *for p odd, $L_1(G^p, r^p, \ell^p) = F_{p-1} + 1$ and $L_2(G^p, r^p, \ell^p) = F_p$;*
- *for p even, $L_1(G^p, r^p, \ell^p) = F_p + 1$ and $L_2(G^p, r^p, \ell^p) = F_{p-1}$;*
- *for all p, there is a subgraph H^p of G^p with one less edge than G^p that satisfies $L_1(H^p, r^p, \ell^p) = 1$ and $L_2(H^p, r^p, \ell^p) = 0$.*

Theorem 4 has a number of implications. We begin by noting two immediate corollaries of the theorem.

Corollary 5 *Adding a single edge to an n-vertex two-commodity instance can increase the latency of all traffic by a $2^{\Omega(n)}$ factor as $n \to \infty$.*

Corollary 6 *If $\mathcal{I}_n$ is the set of instances with at most n vertices, then $\rho(\mathcal{I}_n) = 2^{\Omega(n)}$ as $n \to \infty$.*

Furthermore, it is trivial to extend Corollary 5 to show that for any $k \geq 2$, adding a single edge to an n-vertex k-commodity instance can increase the latency of *all* traffic by $2^{\Omega(n)}$ as $n \to \infty$.

Theorem 4 and Corollaries 5 and 6 show that a number of previously established properties of single-commodity instances do not carry over to multicommodity networks. In particular, the following statements are known to hold in single-commodity instances.

(1) Adding one edge to a single-commodity instance can only increase the maximum or average latency of a Nash flow by a factor of 2 [LRT04].

(2) Adding any number of edges to an n-vertex single-commodity instance can only increase the maximum or average latency of a Nash flow by a factor of $\lfloor n/2 \rfloor$ [Rou].

(3) The price of anarchy with respect to maximum latency in an n-vertex single-commodity instance is at most $n - 1$ [Rou04].

Theorem 4 and Corollaries 5 and 6 demonstrate that all of these statements utterly fail to extend to multicommodity networks, even to those with only two commodities. This dichotomy stands in contrast to other work on selfish routing, such as bounds on the price of anarchy with respect to the average latency objective function, where there is provably no separation between single-commodity and multicommodity instances [Rou02b; CSM04b].

We now give the construction of the family of instances claimed in Theorem 4. We begin by defining the graph G^p for $p \geq 1$, see Figure 2.5. We will describe the construction only for p odd; the construction for even p is similar. We begin with two paths, which we will call P_1 and P_2. The $(p + 3)$-vertex path P_2, drawn vertically in Figure 2.5, is $s_2 \to w_0 \to w_1 \to \cdots \to w_p \to t_2$. The $(p + 4)$-vertex path P_1, drawn horizontally in Figure 2.5, is $s_1 \to a \to w_1 \to v_1 \to \cdots \to v_p \to t_1$. We also add the following edges between the two paths:

- (a, w_i) for all positive even i;
- (v_i, w_i) for all positive even i;
- (s_2, v_i) for all odd i at most $p - 2$;
- (w_i, v_i) for all odd i.

Finally, we complete G^p by adding what we will call an *extra edge*, defined as the edge (s_1, w_0).

For all p, the traffic rate vector r^p will be $r_1^p = r_2^p = 1$. To complete the construction, we therefore need only describe the edge latency functions. All edges will either possess a constant latency function, or a latency function that approximates a step function. We next introduce notation for the latter function type. For a positive integer i and a positive real number δ, f_δ^i will denote a continuous, nondecreasing function satisfying $f_\delta^i(x) = 0$ for

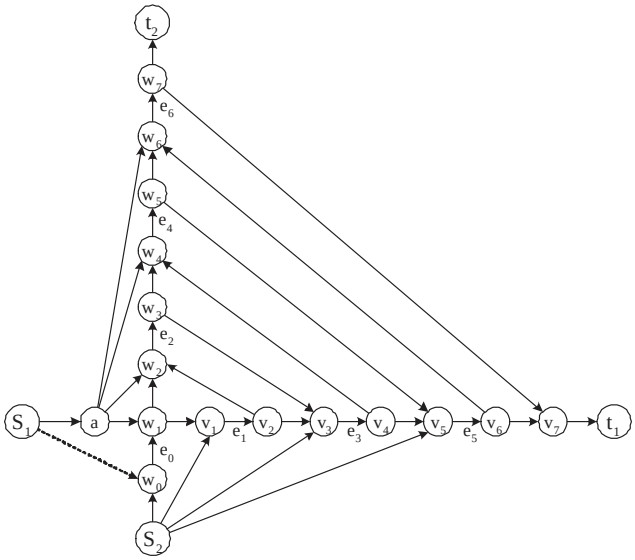

Figure 2.5: Construction of the instance (G^p, r^p, ℓ^p) when $p = 7$. Dotted edge (s_1, w_0) is the "extra edge". Edges with non-constant latency functions are labelled.

$x \leq 1$ and $f^i_\delta(x) = F_i$ for $x \geq 1+\delta$. (The function f^i_δ can be defined arbitrarily on $(1, 1+\delta)$, provided it is continuous and nondecreasing.)

For $i \in \{0, 1, \ldots, p-1\}$, we define the edge e_i to be (w_i, w_{i+1}) if i is even and (v_i, v_{i+1}) if i is odd. (See Figure 2.5.) We now define the latency functions ℓ^p for G^p as follows, where δ is sufficiently small (to be chosen later): for each $i > 0$, edge e_i receives the latency function $\ell^p(x) = f^i_\delta(x)$, edge e_0 receives the latency function $\ell^p(x) = f^1_\delta(x)$, edge (s_1, a) receives the latency function $\ell^p(x) = 1$, and all other edges receive the latency function $\ell^p(x) = 0$.

With the construction in hand, we now turn toward proving Theorem 4 for odd p (the arguments for even p are similar). Part (a) is obvious. Part (d) is easy to see: if H^p is obtained from G^p by removing the extra edge (s_1, w_0) and f is the flow that routes one unit of traffic on both P_1 and P_2, then f is at Nash equilibrium for (G^p, r^p, ℓ^p), showing that $L_1(H^p, r^p, \ell^p) = 1$ and $L_2(H^p, r^p, \ell^p) = 0$. (See Figure 2.6.)

To finish the proof of Theorem 4 (for p odd), we need only prove part (b). We will accomplish this via a sequence of lemmas, the first of which requires some further definitions. First, we will say that a flow f, feasible for (G^p, r^p, ℓ^p), *floods* the instance if $f_{e_i} \geq 1 + \delta$ for all $i \in \{0, 1, \ldots, p-1\}$. Thus if f floods (G^p, r^p, ℓ^p), all edge latencies are at

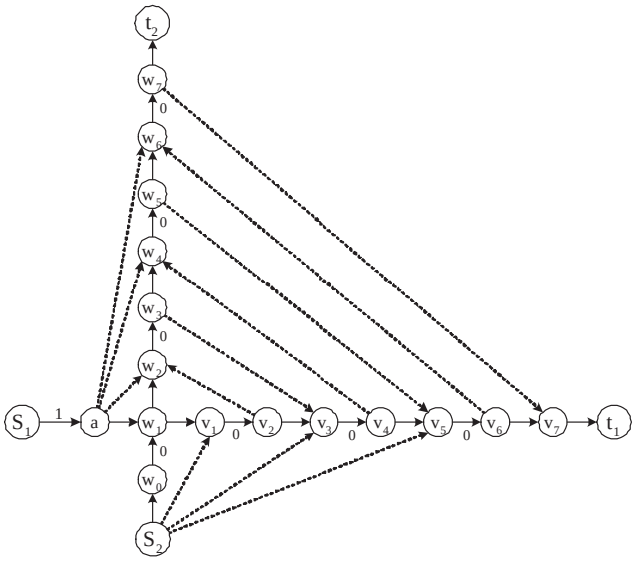

Figure 2.6: Nash flow in (H^p, r^p, ℓ^p), with $p = 7$, where H^p is G^p with the extra edge (s_1, w_0) removed. Solid edges carry flow, dotted edges do not. Edge latencies are with respect to the Nash flow. Unlabelled edges have zero latency.

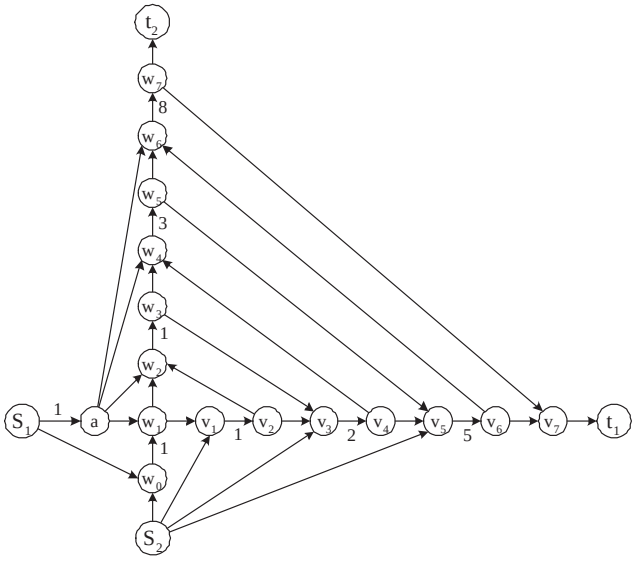

Figure 2.7: Nash flow in (G^p, r^p, ℓ^p), with $p = 7$. Solid edges carry flow, dotted edges do not. Edge latencies are with respect to the Nash flow. Unlabelled edges have zero latency.

their maximum, as in Figure 2.7. Second, we introduce notation for some of the paths of G^p. For i even, Q_i will denote the unique s_1-t_1 path which traverses edge e_i before any odd labelled edges, and includes no other edge of P_2. For i odd, Q_i will denote the unique s_2-t_2 path which traverses edge e_i before any even labelled edges, and includes no other edge of P_1. We will call the paths $Q_0, \ldots, Q_{p-1}$, together with the "axis-aligned" paths P_1 and P_2, the *short paths*. The next lemma justifies this terminology, at least for flows that flood the instance (G^p, r^p, ℓ^p).

Lemma 1 *If f floods (G^p, r^p, ℓ^p) with p odd, then:*

(a) $\ell_P(f) \geq F_{p-1} + 1$ for every s_1-t_1 path P, and equality holds for short paths;

(b) $\ell_P(f) \geq F_p$ for every s_2-t_2 path P, and equality holds for short paths.

We will only prove part (b) of Lemma 1, as the proof of part (a) is similar. In the proof, we will use the following lemma about Fibonacci numbers, which is easy to verify.

Lemma 2 *Let j and p be odd positive integers with $j < p$, and I the even numbers between j and p. Then, $F_j + \sum_{i \in I} F_i = F_p$.*

We now prove Lemma 1.

Proof of Lemma 1: Let P be an s_2-t_2 path. Let j be the largest odd number such that $e_j \in P$, or 0 if there is no such number. We only need to prove the case where $j > 0$, since the $j = 0$ and $j = 1$ cases are the same. If $j > 0$, then P contains e_j and also e_i for all even i between j and p. Since f floods (G^p, r^p, ℓ^p), Lemma 2 implies that $\ell_P(f) \geq F_p$. Moreover, this inequality holds with equality for short paths. ■

Our final lemma states that routing flow on short paths suffices to flood the instance (G^p, r^p, ℓ^p). For the statement of the lemma, recall that the parameter δ controls how rapidly the non-constant latency functions of (G^p, r^p, ℓ^p) increase as the amount of flow on the edge exceeds one.

Lemma 3 *For all p odd and δ sufficiently small, there is flow f, with $f_P > 0$ only for short paths P, that floods (G^p, r^p, ℓ^p).*

Define the flow f as follows. First, for $i = 0, 1, \ldots, p-1$, route $2^{-(i+1)}$ units of flow (of the appropriate commodity) on the short path Q_i. This routes strictly less than one unit of flow of each commodity. The remaining flow is then routed on the short paths P_1 and P_2.

To complete the proof, we need to show that f floods (G^p, r^p, ℓ^p)—that $f_{e_i} \geq 1 + \delta$ for all $p \in \{0, 1, \ldots, p-1\}$ provided δ is sufficiently small. We will prove this inequality only for i odd; the argument for even i is similar.

The second commodity uses edge e_i only in the short path Q_i, on which it routes $2^{-(i+1)}$ units of flow. The first commodity uses edge e_i in all of its flow paths except for

the short paths Q_j for j even and greater than i. The total amount of flow on e_i is thus at least

$$2^{-(i+1)} + 1 - \sum_{j \geq 0} 2^{-(i+2+2j)} = 1 + 2^{-(i+1)} - \frac{4}{3} \cdot 2^{-(i+2)} > 1 + 2^{-(i+3)}.$$

Thus, provided $\delta \leq 2^{-(p+3)}$, f floods (G^p, r^p, ℓ^p), and the proof is complete. ■

Theorem 4(b) now follows immediately from Definition 1, Lemma 1, and Lemma 3.

2.4.2 Upper Bounding the Price of Anarchy in Multicommodity Networks

We now turn toward proving upper bounds on the price of anarchy and, as a consequence, on the worst-possible severity of Braess's Paradox. We will aim for an upper bound that matches the lower bound of Theorem 4, and will largely succeed in this goal.

We begin by proving a very weak bound on the price of anarchy, a bound that depends on parameters other than the size of the network. While not interesting in its own right, this bound will play a crucial role in later proofs in this section.

Lemma 4 *Let f be a Nash flow and f^* a feasible flow for an instance (G, r, ℓ), where G has m edges. For every edge e of G with $f_e > f_e^*$,*

$$\ell_e(f_e) \leq \frac{m \sum_i r_i}{f_e - f_e^*} \cdot \max_e \ell_e(f_e^*). \tag{2.2}$$

Proof of Lemma 4: Let $F \subseteq E$ denote the edges e of G for which $f_e > f_e^*$. Using inequality (2.1) in Proposition 3 and the fact that $\ell_e(f_e^*) \leq M(f^*)$ whenever $f_e^* > 0$, we can derive the following crude bound:

$$\begin{aligned}
\sum_{e \in F} \ell_e(f_e)(f_e - f_e^*) &\leq \sum_{e \in E \setminus F} \ell_e(f_e)(f_e^* - f_e) \\
&\leq \left(\max_e \ell_e(f_e^*)\right) \sum_{e \in E \setminus F} (f_e^* - f_e) \\
&\leq \max_e \ell_e(f_e^*) \cdot m \cdot \sum_i r_i.
\end{aligned}$$

The lemma now follows easily. ■

We next use Lemma 4 as a bootstrap for deriving upper bounds on the price of anarchy that depend only on the size of the network. We will accomplish this as follows.

For an arbitrary instance, we will set up a linear program, with edge latencies as variables, that maximizes the price of anarchy among instances that are "basically equivalent" to the given instance. We will define our notion of equivalence so that Lemma 4 ensures that the linear program has a bounded maximum, and will then analyze the vertices of the feasible region of the linear program to derive the following bound.

Theorem 7 *If (G, r, ℓ) is an instance with n vertices and m edges, then*

$$\rho(G, r, \ell) = 2^{O(m \log n)}.$$

Before implementing the proof approach outlined above, we state a proposition that bounds the maximum size of the optimal value of a linear program with a constraint matrix with entries in $\{-1, 0, 1\}$.

Proposition 8 *Let A be an $m \times n$ matrix with entries in $\{0, \pm 1\}$ and at most α non-zero entries in each row. Let b be a real-valued m-vector, and let the linear program $\max x_i$ subject to $Ax \leq b$ have a finite maximum. Then, this maximum is at most $n\alpha^n \|b\|_\infty$, where $\|b\|_\infty$ denotes $\max_j |b_j|$.*

Proposition 8 can be proved with Cramer's rule and a simple bound on the determinant. We omit further details.

Proof of Theorem 7: Let (G, r, ℓ) be an instance with n vertices and m edges. Let f and f^* be Nash and optimal flows for (G, r, ℓ), respectively. We aim to show that $\rho(G, r, \ell) = 2^{O(m \log n)}$.

We begin by performing some preprocessing on the instance (G, r, ℓ). First, if $f_e = f_e^* = 0$ for some edge e, then that edge can be removed from the instance without affecting its ρ-value. We can therefore assume that $f_e^* > 0$ or $f_e > 0$ for every edge e. Second, we can assume that $\ell_e(0) = 0$ whenever $f_e^* = 0$. To see why, note that replacing the latency function $\ell_e(x)$ of such an edge by the function equal to (e.g.) $\min\{x/f_e, 1\} \cdot \ell_e(x)$ leaves the Nash flow unaffected while only decreasing the maximum latency of f^* and hence increasing the ρ-value of the instance. Combining these two assumptions, we can assume without loss of generality $\ell_e(f_e^*) \leq M(f^*)$ for every edge e of G.

We now set up a linear program that attempts to further transform the latency functions to make the ρ-value of the given instance as large as possible. In the linear

program, the flow amounts $\{f_e\}$ and $\{f_e^*\}$, as well as the latencies $\{\ell_e(f_e^*)\}$ with respect to f^*, will be held fixed. There will be a nonnegative variable $\hat{\ell}_e(f_e)$ representing the latency of edge e with respect to the flow f. So that the new latency functions are nondecreasing, we impose the following linear constraints, which we call *monotonicity constraints*:

- For all edges e with $f_e = f_e^*$, $\hat{\ell}_e(f_e) = \ell_e(f_e^*)$.
- For all edges e with $f_e < f_e^*$, $\hat{\ell}_e(f_e) \leq \ell_e(f_e^*)$.
- For all edges e with $f_e > f_e^*$, $\hat{\ell}_e(f_e) \geq \ell_e(f_e^*)$.

Additionally, we will insist that the (fixed) flow f be at Nash equilibrium with respect to the (variable) latencies $\{\hat{\ell}_e(f_e)\}$. There are several ways that this requirement can be encoded with linear constraints. For this proof, we will be content with the following naive approach: for every commodity i, and every pair of paths $P, \tilde{P} \in \mathcal{P}_i$ for which $f_e^{(i)} > 0$ for all $e \in P$, we insist that $\sum_{e \in P} \hat{\ell}_e(f_e) \leq \sum_{e \in \tilde{P}} \hat{\ell}_e(f_e)$ in our linear program. Since this linear program has a small number of variables, we will not be hampered by its potentially massive number of constraints.

By construction, our constraints ensure the following: for every feasible solution $\{\hat{\ell}(f_e)\}$, there is an instance $(G, r, \hat{\ell})$ with continuous, nondecreasing latency functions $\hat{\ell}$, so that these latency functions interpolate their two prescribed values and f is a Nash flow for $(G, r, \hat{\ell})$. Consider the objective function $\max \hat{\ell}_e(f_e)$ for an edge e. Our key claim is that the resulting linear program is not unbounded. For edges e with $f_e \leq f_e^*$, the claim is obvious from the constraints. For edges e with $f_e > f_e^*$, the claim follows from Lemma 4 and the fact that all parameters on the right-hand side of the bound (2.2) are fixed in the linear program.

Since the maximum of the above linear program is bounded, we can apply Proposition 8. In our linear program, there are a total of m variables, of which each constraint contains at most $2n$. The right-hand side of each constraint is either a 0 or a term of the form $\ell_e(f_e^*)$. By our preprocessing step, $\ell_e(f_e^*) \leq M(f^*)$ for all edges e. Hence, Proposition 8 implies that the maximum of the linear program is at most $mn^{O(m)} \cdot M(f^*)$. Hence, returning to the original instance (G, r, ℓ), we must have $\ell_e(f_e) \leq mn^{O(m)} \cdot M(f^*)$ for all edges e. Since a flow path of f can contain only n edges, we can conclude that $\rho(G, r, \ell) \leq nmn^{O(m)} = 2^{O(m \log n)}$. ■

When the number of commodities is small (e.g., a constant), we can improve the bound to $2^{O(kn)}$ using a different way to encode the constraint that f must be at Nash equilibrium with respect to $\{\hat{\ell}_e(f_e)\}$.

Theorem 9 *If (G, r, ℓ) is an instance with n vertices and k commodities, then*

$$\rho(G, r, \ell) = 2^{O(kn)}.$$

To prove our bound, we start with the linear program described in Theorem 7. We leave the objective and monotonicity constraints the same, but replace the constraints that ensure f is a Nash equilibrium for $\{\hat{\ell}_e(f_e)\}$. To ensure the latencies $\{\hat{\ell}_e(f_e)\}$ define a Nash equilibrium for f, we introduce an auxiliary variable $\hat{d}^i(v)$ for each commodity i and for every vertex $v \in V$ reachable from that commodity's source s_i, which will represent the length of the shortest path from s_i to v, with respect to the latencies $\{\hat{\ell}_e(f_e)\}$. Now, we define the following constraints:

- $\hat{d}_i(s_i) = 0$, for all commodities i.
- $\hat{d}_i(u) + \hat{\ell}_e(f_e) = \hat{d}_i(v)$, for all edges $e = (u, v)$ and commodities i with $f_e^{(i)} > 0$.
- $\hat{d}_i(u) + \hat{\ell}_e(f_e) \leq \hat{d}_i(v)$, for all edges $e = (u, v)$ and commodities i.

To complete the proof, we need to show:

(a) A set of latencies $\{\hat{\ell}_e(f_e)\}$ is feasible for our linear program if and only if it defines a Nash equilibrium for f.

(b) The latency variables can be removed, yielding a linear program with kn variables, that can be bounded by $2^{O(kn)}$.

To prove the forward direction of (a), it is easy to see that with our constraints, for every commodity i, each path $P \in \mathcal{P}_i$, with $f_e^{(i)} > 0$ for all $e \in P$, must have length precisely equal to $\hat{d}_i(t_i)$. Furthermore, no path in $P \in \mathcal{P}_i$ may have length strictly less than

$\hat{d}_i(t_i)$. Therefore, for every commodity i and every $P, \tilde{P} \in \mathcal{P}_i$, with $f_e^{(i)} > 0$ for all $e \in P$, $\sum_{e \in P} \hat{\ell}_e(f_e) \leq \sum_{e \in \tilde{P}} \hat{\ell}_e(f_e)$, and our latencies define a Nash equilibrium for f. To prove the other direction of (a), note that for any set of latencies $\hat{\ell}_e(f_e)$ defining a Nash equilibrium for f, we can define $\hat{d}_i(v)$ to be the length of the shortest path from s_i to v, and we have a feasible solution for our linear program.

With (a) proven, we know that the maximum value of our linear program is precisely the maximum edge length that occurs in any Nash equilibrium for flow f. As before, note that Lemma 4 implies our linear program is bounded, and we can apply Proposition 8 to bound the maximum value of the linear program.

Before applying Proposition 8 however, we first eliminate the latency variables, which allows us to prove a better bound. Note that for any edge $e = (u, v)$ with $f_e^i > 0$ for some commodity i, $\hat{d}_i(u) + \hat{\ell}_e(f_e) = \hat{d}_i(v)$, so we can replace any occurrence of $\hat{\ell}_e(f_e)$ in our linear program with $\hat{d}_i(v) - \hat{d}_i(u)$. Furthermore, for any other edges, $f_e = 0$, and it must be the case that $\hat{\ell}_e(f_e) \leq M(f^*)$. For these edges, an optimal solution must assign some value $x_e \leq M(f^*)$ to the latencies $\hat{\ell}_e(f_e)$, and thus for these edges, we can substitute the $\hat{\ell}_e(f_e)$ variable with the constant x_e value used in the optimal solution. With these substitutions, we have not changed the optimal value of our linear program, and we are only left with $O(kn)$ variables. Moreover, there are still a constant number of variables per constraint, and each entry of b is still bounded by $M(f^*)$. Therefore, applying Proposition 8 bounds the price of anarchy by $2^{O(kn)}$. ■

Corollary 6 shows that Theorem 9 is essentially tight for a constant number of commodities.

2.4.3 Exponential Inapproximability for Multicommodity Network Design

In this section, we will show that a network design problem that is naturally motivated by Braess's Paradox has intrinsically exponential approximability. The problem, which we call Multicommodity Network Design (MCND), is as follows.

> Given a (multicommodity) instance (G, r, ℓ), find a subgraph H of G that minimizes $M(H, r, \ell)$.

By $M(H, r, \ell)$, we mean the maximum latency of a Nash flow for (H, r, ℓ) (well defined by Proposition 2). *MCND* is tantamount to detecting and avoiding Braess's Paradox. For single-commodity instances, this problem was studied in [Rou].

The *trivial algorithm* is defined as the algorithm that always returns the entire graph G—the algorithm that always punts on trying to detect Braess's Paradox. The following was proved in [Rou]: the trivial algorithm is an $\lfloor n/2 \rfloor$-approximation algorithm for the special case of single-commodity instances; and for every $\epsilon > 0$, no $(\lfloor n/2 \rfloor - \epsilon)$-approximation algorithm exists (assuming $P \neq NP$). Here, we will succeed in proving analogues of these results for multicommodity networks, where the best-possible approximation ratio is inherently exponential.

First, we note that since Theorems 7 and 9 imply limits on the largest possible increase in the maximum latency due to Braess's Paradox, they also translate to an upper bound on the trivial algorithm.

Proposition 10 *The trivial algorithm is a* $2^{O(\min\{kn, m \log n\})}$*-approximation algorithm for* MCND.

Much more interesting is the next result, which states that there is *no* polynomial-time algorithm with subexponential approximation ratio (assuming $P \neq NP$).

Theorem 11 *Assuming* $P \neq NP$*, there is no* $2^{o(n)}$*-approximation algorithm for* MCND.

The proof of Theorem 11 combines ideas from the gap reductions of [Rou] for the single-commodity version of MCND with the family of two-commodity instances described in Section 2.4.1. We first provide a high-level overview of the proof below, and then describe the full proof in the next subsection.

Recall that in an instance of the NP-complete problem PARTITION, we are given q positive integers $\{a_1, a_2, \ldots, a_q\}$ and seek a subset $S \subseteq \{1, 2, \ldots, q\}$ such that $\sum_{j \in S} a_j = \frac{1}{2} \sum_{j=1}^{q} a_j$ [GJ79, SP12]. The idea of the reduction is to start with an instance (G^p, r^p, ℓ^p) of the form described in Section 2.4.1, and to replace the extra edge (s_1, w_0) with a collection of parallel edges representing an instance $\mathcal{I} = \{a_1, \ldots, a_p\}$ of PARTITION. We will give these edges latency functions that simulate "capacities", with an edge representing an integer a_j of $\mathcal{I}$ receiving capacity proportional to a_j. The proof then break down into three parts.

First, if too many of these parallel edges are removed from the network, there will be insufficient remaining capacity to send flow cheaply. To implement this, we must also augment the latency functions of the edges $e_0, \ldots, e_{p-1}$ of Section 2.4.1 to have effective capacities. Second, if too few of the parallel edges are removed, the excess of capacity results in a bad flow at Nash equilibrium similar to that of Figure 2.7. Finally, these two cases can be avoided if and only if $\mathcal{I}$ is a "yes" instance of PARTITION, in which case removing the appropriate collection of parallel edges results in a network that admits a good Nash equilibrium similar to that of Figure 2.6.

The Formal Reduction

We now prove Theorem 11 formally by showing the complete reduction.

Proof of Theorem 11: Given an instance $I = \{a_1, a_2, \ldots, a_q\}$ of PARTITION, we will construct a network G with n nodes such that:

(i) If $\mathcal{I}$ is a "yes" instance of PARTITION, then G admits a subgraph H with $M(H, r, \ell) = 1$.

(ii) If $\mathcal{I}$ is a "no" instance, then $M(H, r, \ell) \geq F_p$, where $p = \lfloor \frac{n-6}{2} \rfloor$, for every subgraph H of G.

For our statement we will assume $n \geq 12$, and for simplicity, we will only prove the statement where p is odd and greater than 3. The network we construct will have the same edges as the network described in section 2.4.1, except for the (s_1, w_0) edge. In its place, we will add one edge e^j for each element a_j of the partition instance I for a total of q edges between s_1 and w_0.

We will continue to use the same notation e_i, P_1, P_2, Q_i to refer to the same edges and paths as defined before. However, as we have created some new short paths through e_0, we re-define Q_0 to be the set of all short paths passing through e_0. If we count Q_0 as one path and set $c = \frac{p+1}{2}$, then we have defined $c + 1$ paths from s_1 to t_1 and c paths from s_2 to t_2. We will set our rate for (s_1, t_1) to be $c + 1$, and the rate for (s_2, t_2) to be c. (In the optimal construction, the flow at Nash equilibrium will send one unit of flow along each path we have defined). For clarity, it will also be useful to define the quantity $\gamma(i) = \lfloor \frac{i+5}{2} \rfloor$, which represents the number of paths P_1, P_2, or Q_i that pass through edge e_i.

Before we define the latency functions for our edges, we first define a few constants that will be useful. For some edges, we will want to limit the amount of flow travelling along the edge, so after a certain flow amount, we will set the latency equal to $M_0 = F_p$. Additionally, we will want some latency functions to have a steep transition from say 1 to M_0, so we will define a small value $\delta = 1/(cAq \cdot 2^{p+3})$, and allow our latency function to increase from 1 to M_0 over δ units of flow. For our latency functions, we will not define the latency function over all intervals, but any function that is continuous and non-decreasing over the undefined interval will suffice for our reduction, as it will not affect the latencies experienced by the Nash or optimal flows.

- (A) For edge $e = (s_1, a)$:

$$\ell_e(f_e) = \begin{cases} 1 & \text{if } f_e \leq c \\ M_0 & \text{if } f_e \geq c + \delta \end{cases}$$

- (B) For each edge e of the form (a, w_i) or (s_2, v_i):

$$\ell_e(f_e) = \begin{cases} 0 & \text{if } f_e \leq 1 + \frac{1}{2c} \\ M_0 & \text{if } f_e \geq 1 + \frac{1}{c} \end{cases}$$

- (C) For edge e_0:

$$\ell_{e_0}(f_{e_0}) = \begin{cases} 0 & \text{if } f_{e_i} \leq \gamma(0) \\ 1 & \text{if } \gamma(0) + \delta \leq f_{e_i} \leq \gamma(0) + \frac{1}{2c} \\ M_0 & \text{if } \gamma(0) + \frac{1}{c} \leq f_{e_i} \end{cases}$$

- (D) For each edge e_i for $i \neq 0$:

$$\ell_{e_i}(f_{e_i}) = \begin{cases} 0 & \text{if } f_{e_i} \leq \gamma(i) \\ F_i & \text{if } f_{e_i} \geq \gamma(i) + \delta \end{cases}$$

- (E) For each edge e^j:

$$\ell_{e^j}(f_{e^j}) = \begin{cases} 0 & \text{if } f_{e_i} \leq \frac{2a_j}{A} - \delta \\ 1 & \text{if } f_{e_i} = \frac{2a_j}{A} \\ M_0 & \text{if } f_{e_i} \geq \frac{2a_j}{A} + \delta \end{cases}$$

- (F) Every other unlabelled edge has constant latency 0

To prove claim (i) of the reduction, suppose there is some set $S \subset \{1, 2, ..., q\}$ such that $\sum_{j \in S} a_j = \frac{A}{2}$. We will show that deleting all edges e^j for $j \notin S$ produces a network with latency 1 at Nash equilibrium. Our flow at Nash equilibrium will send one unit of flow along each path P_1, P_2, and Q_i. For Q_0, our unit of flow will be divided among the e^j edges so that there is $\frac{2a_j}{A}$ flow along each edge for a total of $\sum_{j \in S} \frac{2a_j}{A} = 1$ flow along the paths in Q_0. This flow is feasible since we have $c + 1$ flow between s_1 and t_1, and c flow between s_2 and t_2. To show the flow is at Nash equilibrium, it can be verified that the e^j edges and the (s_1, a) have latency 1 in this flow, and all other edges have latency 0. Therefore, all (s_1, t_1) flow paths have 1 unit of latency, and all (s_2, t_2) paths have 0 units of latency. So we have proved that a "yes" instance of partition implies a subgraph with maximum latency 1 at Nash equilibrium.

To complete the proof of correctness for the reduction, we now need to show that a "no" instance of partition implies all subgraphs of G have at least F_p maximum latency at Nash equilibrium (ii). First note, that deleting any edge that is not between s_1 and w_0, will cause at least one edge type of (A), (B), or (C), to have too much flow and latency will increase to M. So now we just need to show that any subgraph that includes all edges in G, except some subset of edges between s_1 and w_0, will also experience exponential latency, provided that we have a "no" instance of partition. First note that if we only included some set of edges $\{e^j : j \in S\}$ such that $S \subset \{1, 2, ..., q\}$ and $\sum_{j \in S} a_j < \frac{A}{2}$, then the maximum amount of flow that can travel between s_1 and w_0 without incurring M latency will be $\frac{2}{A}(\sum_{j \in S} a_j) + q\delta \leq \frac{2}{A}(\frac{A}{2} - 1) + q\delta \leq 1 - \frac{2}{A} + \frac{1}{A} \leq 1 - \frac{1}{A} < 1 - \delta$. However, we cannot send more than $c + \delta$ flow along the (s_1, a) edge without incurring M_0 latency as well. Therefore, if we only include some set of edges e^j whose corresponding a_j values sum to strictly less than $\frac{A}{2}$ then the maximum latency will be at least M_0.

Now we need to analyze the last case, where we include all edges not between s_1 and w_0, and some set of edges e^j such that their corresponding a_j values sum to a value strictly greater than $\frac{A}{2}$. The amount of flow that can pass with 0 latency in this case is $\frac{2}{A}(\frac{A}{2} + 1) - q\delta \geq 1 + \frac{2}{A} - \frac{1}{A} \geq 1 + \frac{1}{A}$. We will show that a Nash flow experiences latency equal to F_p as in the bad case of Braess's Paradox. To show this recall in our original Braess's Paradox graph, some amount of flow was diverted from the P_1 and P_2 paths and $2^{-(i+1)}$ flow was sent along each Q_i path. This was enough to cause the each edge e_i to have flow

in excess of $1 + 2^{-(p+3)}$ causing each edge to increase latency to F_i. A similar process will cause a the maximum latency to become F_p in this case as well. Imagine we start with 1 unit of flow along each path P_1, P_2, and Q_i. Now we shift flow from P_1 and P_2 such that each Q_i path has an additional $1/(cA \cdot 2^{i+1})$ along it. Note that this will be enough flow to cause each e_i edge to have more than $\gamma(i) + \delta$ flow on it to cause latency equal to F_i. Now, we just need to verify that edges of type (A), (B), (C), and (E) have the proper latencies. It is not difficult to verify that edges of type (A), (B), (C) have flows that cause latency 1, 0, and 1 respectively. Lastly since there is only $1 + 1/(2cA)$ flow between s_1 and w_0, edges of type (E) have 0 latency. Therefore, we have shown that if we have a "no" instance of PARTITION, all subgraphs of G have latency at least F_p. ■

Chapter 3

Linked Decompositions for Internet Routing

3.1 Background

A *linked decomposition* of a graph with n nodes is a set of subgraphs covering the n nodes such that all pairs of subgraphs intersect; we seek linked decompositions such that all subgraphs have about $\sqrt{n}$ vertices, logarithmic diameter, and each vertex of the graph belongs to either one or two subgraphs. A linked decomposition enables many control and management functions to be implemented locally, such as resource sharing, maintenance of distributed directory structures, deadlock-free routing, failure recovery and load balancing, without requiring any node to maintain information about the state of the network outside the subgraphs to which it belongs. Linked decompositions also enable efficient routing schemes with small routing tables, which we describe in Section 3.7. Our main contribution is to show that "Internet-like graphs" (e.g. the preferential attachment model proposed by Barabasi et al. [BA99] and other similar models) have linked decompositions with the parameters described above with high probability; moreover, our experiments show that the Internet topology itself can be so decomposed. Our proof proceeds by analyzing a novel process, which we call *Polya urns with the power of choice*, which may be of great independent interest. In this new process, we start with n nonempty bins containing $O(n)$ balls total, and each arriving ball is placed in the least loaded of m bins, drawn independently at random *with probability proportional to load*. Our analysis shows that in our new process,

with high probability the bin loads become roughly balanced some time before $O(n^{2+\epsilon})$ further balls have arrived and stay roughly balanced, regardless of how the initial $O(n)$ balls were distributed, where $\epsilon > 0$ can be arbitrarily small, provided m is large enough.

Previously it was known that throwing balls into bins uniformly at random produces relatively well-balanced occupancies with high probability. In contrast, when each new ball is added to a bin selected with probability proportional to its current occupancy, we get the much-studied model of *Polya urns,* which is known to produce large imbalances: with n urns, the largest urn is expected to have $\log n$ times more balls, and the smallest urn n times fewer balls, than the average one. It is a celebrated result in our field that if, while throwing balls into bins, we choose two random bins and add a ball only to the smallest one, then this *power of two choices* makes the already small imbalances significantly smaller [ABKU94; KLadH92; Mit01]. But what about utilizing the power of choice in the Polya urns model? Does selecting $m \geq 2$ bins according to the Polya urn distribution, and adding a ball to the least loaded bin, result in balanced loads?

In this dissertation, we show that *the power of choice in the Polya urn model does balance bin loads.* In particular, we show (Theorem 14) that if n nonempty bins start with $O(n)$ balls total, then throwing $O(n^{2+\epsilon})$ more balls according to our new Polya urns process with the power of choice, balances all bins within a multiplicative factor of $(1+\epsilon)$ with high probability, where $\epsilon > 0$ can be made arbitrarily small, provided we make the number of choices m large enough. (See Section 3.3 for the exact dependence on m). Moreover, once the bins become roughly balanced, they stay roughly balanced as more balls are added. As m increases, our result becomes essentially tight, since $\Omega(n^2)$ balls are required to balance an initial distribution that starts with $\Omega(n)$ balls in one bin, and 1 ball in every other bin. We can also show the bin loads balance when $m = 2$ choices are used and poly(n) balls are thrown, but it remains an interesting open question whether or not the loads also balance when $O(n^{2+\epsilon})$ new balls are thrown with $m = 2$ choices.

The motivation for this investigation comes from a graph-theoretic problem related to the Internet. Let G be a graph, and a, b, c, d be integer parameters. We say that a graph G has an (a, b, c, d)-*linked decomposition* if there are c connected subgraphs of G, which we call *components*, such that

1. each component has between $\frac{a}{4}$ and a nodes

2. each node belongs to at least one and at most b components

3. the diameter of each component is at most d

4. most importantly, *any two components have a node in common.*

Our notion of linked decomposition is related to the notion of *sparse covers* defined in [AP90], which decomposes the graph into connected subgraphs satisfying properties (2) and (3). Other related concepts of network decomposition are introduced in [AGLP89] and [PS89].

For our purposes, we would like $a, c \approx \sqrt{n}$, $b = 2$, and $d = O(\log n)$, where n is the number of nodes in G. It is not hard to see that a linked decomposition, if it exists for graph G, would yield a routing algorithm with unstructured node names with $O(d)$ delay (worst-case number of hops) and $O(d)$ traffic (total messages sent per packet), and $\tilde{O}(ab + n/a)$ storage per node (see Section 3.7 for details), which is about $\sqrt{n}$ for the parameters discussed. Moreover, using a more relaxed form of linked decomposition and structured node names, we can achieve about $O(n^{1/3})$ storage per node and $\tilde{O}(\log n)$ delay/messages sent as well. In other words, a network with our linked decomposition property can be decomposed into subnetworks of balanced size that can be administered relatively independently, with very little harm to communication. For more details regarding our routing schemes and their implications for network routing, see Section 3.7.

But why should we expect that we can decompose networks in this way? Rather surprisingly, our experiments in Section 3.8) indicate that *the existing Internet graph does have linked decompositions with these approximate parameters* (both at the router and autonomous system level).

A linked decomposition with parameters $a, c \approx \sqrt{n}$, $b = 2$, and $d = O(\log n)$ is a very demanding requirement, and hence an unlikely property of graphs; on the other hand the Internet seems to have it. We suspect that this is not a coincidence, and that "any Internet-like graph" is very likely to have a linked decomposition with these parameters. Towards this goal, we turn to the simplest and perhaps most influential model of Internet-like graphs, namely the *preferential attachment model* $\mathcal{PA}(m)$ proposed by [BA99] in 1999, and since then studied extensively (see [BR02] for a survey). In this model, nodes arrive one after the other, and when a node arrives, it is connected via m edges to previously arrived nodes. In particular, for each new node t, we pick m previous nodes, i.i.d. at random and with replacement, with probability proportional to the degree of the node at the current time, and create an edge from node t to each of the m nodes picked (with parallel edges

possible).

Our main result is Theorem 12 which states that *for any $\epsilon > 0$, there is a $m \geq 2$, such that $\mathcal{PA}(m)$ produces a graph which has a linked decomposition with parameters $a = \Theta(n^{1/2+\epsilon})$, $b = 2$, $c = \Theta(n^{1/2-\epsilon})$, and $d = O(\log n)$, with probability $1 - n^{-\epsilon}$.*

The proof of our main result is based on the Polya urns insight from the beginning of the Introduction. To find our decomposition, the idea is to follow the preferential attachment process, and assign the first $c = \Theta(n^{1/2-\epsilon})$ nodes to their own component. Then as the next $\frac{n}{2} - c$ nodes arrive, we look at the nodes/components to which the m edges of arriving node point, and assign the new node to the component whose total degree is lowest. Now, provided the components are roughly balanced in terms of total degree after these first $\frac{n}{2}$ nodes have arrived, it is not hard to assign the last $\frac{n}{2}$ nodes to two components each in a simple way, such that a coupon collector argument can be used to show each pair of components intersect at some node (i.e. property (4) of linked decomposition holds) with high probability. As we show in Section 3.2, it is not hard to prove the other three properties hold as well.

Our main technical challenge is to prove that the components obtain roughly balanced total degree after $\frac{n}{2}$ nodes arrive (Theorem 13), which is needed to prove property (4) of linked decomposition holds. To prove that the degrees of the components become roughly balanced when running $\mathcal{PA}(m)$, note that if we think of the sum of the degrees of each component as the occupancy of a Polya urn, then our random process is very much like the Polya urns process with the power of choice, defined previously, with c bins and m choices. Although Polya urns with the power of choice is not quite the same process as the one we would like to analyze, the analysis used in the proof of Theorem 14, which shows that Polya urns with the power of choice produces roughly balanced bin loads when $O(c^{2+\epsilon})$ balls are thrown, can be modified to prove that the degrees of the components balance when $O(c^{2+\epsilon})$ new nodes arrive, Theorem 13. Here again, in order to make ϵ arbitrarily small, we need to make m sufficiently large. We expect that the dependence of ϵ on m to be similar for Theorem 13, as the dependence given for Polya urns with the power of choice (see Section 3.3), but some details remain to be checked. By proving Theorem 13, we complete the last step needed to prove that a linked decomposition with the required parameters exists with high probability.

One negative aspect of the proof above is the fact that we need to increase m in order to make ϵ arbitrarily small. However, if we relax the definition of linked decomposi-

tion so that each component may contain at most one special node violating property (2) of the linked decomposition — that is, some nodes may be allowed to belong to more than two components — then we can find a linked decomposition even when $m = 2$ (Theorem 15). Moreover, it is easy to see that routing is not harmed by this exception, and we obtain a slightly lower value for $a = \Theta(\sqrt{n \log n})$. The proof proceeds by using a dynamic programming algorithm to decompose the network formed by the first $\Theta(n/\log n)$ nodes into $c = \Theta(\sqrt{n/\log n})$ components of approximately equal size. Once we have c balanced components, it is not very hard to use a coupon collector argument to show that the remaining $c^2 \log c$ nodes are enough to link together these components as required by property (4) of linked decomposition, and the other three properties of linked decomposition are not difficult to prove as well.

3.2 Linked Decompositions in $\mathcal{P}A(m)$ Graphs

The *preferential attachment model* of random graphs $\mathcal{P}A(m)$ [BA99; BR02] creates a sequence of random graphs $G_1, G_2, \ldots, G_t, \ldots$ with multiple edges, where G_t has t nodes. G_1 is a single node with no edges, and G_2 is a graph with m edges between two nodes. To generate G_{t+1} from G_t for $t \geq 2$, a new node $t+1$ is added, and then m nodes are selected i.i.d. at random with replacement, from among nodes $1, \ldots, t$, where each node $i \leq t$ is selected with probability $\frac{d_t(i)}{2mt}$ and $d_t(i)$ is the degree of node i in G_t. Then edges are added in G_{t+1} from node $t+1$ to the m selected nodes. This is a very influential graph-theoretic model in the context of the Internet (even though it is understood that it does not satisfy all known properties of the Internet graph).

By the phrase "with high probability" (whp) in this chapter of the dissertation we shall mean with probability greater than or equal to $1 - n^{-\alpha}$, for some $\alpha > 0$. In most cases, α can be made arbitrarily large by deteriorating the other parameters (e.g. by increasing m in Theorem 12). Following the definition of linked decomposition described in the Introduction, we can prove one of our main theorems:

Theorem 12 *For any $\epsilon > 0$, there exists a $m \geq 2$, such that a graph G_n generated by the $\mathcal{P}A(m)$ has a linked decomposition (whp) with parameters $a = \Theta(n^{1/2+\epsilon})$, $b = 2$, $c = \Theta(n^{1/2-\epsilon})$, and $d = O(\log n)$.*

To find a linked decomposition in a $\mathcal{P}A(m)$ graph with n nodes, we follow the

preferential attachment process, and as nodes arrive, we use a very simple procedure to assign each node t to one or two components.

(1) For node $t \in \{1, \ldots, n^{1/2-\epsilon}\}$, we assign node t to its own component.

(2) For node $t \in \{n^{1/2-\epsilon}+1, \ldots, \frac{n}{2}\}$, we look at the nodes/components to which the m edges of node t point, and assign node t to the component whose total degree (i.e. the sum of the degrees of its nodes) is lowest so far.

(3) For node $t \in \{\frac{n}{2}+1, \ldots, n\}$, we look at where the first two edges of node t point, say to nodes u and v, and assign node t to a component to which u belongs, and a component to which v belongs. (If u or v belongs to two components, then we assign t to a random component of u or v).

It is easy to see that we have decomposed the graph into connected components, and that we have satisfied property 2 of linked decompositions with $b = 2$. Furthermore, since each node t has a constant probability (dependent on m) of joining the component of a node $v \leq \frac{t}{2}$, it is easy to prove that each component has diameter $O(\log n)$ (whp).

To prove the final two properties of our linked decomposition, the key step is to show that at the end of step (2), the total degree of each component is roughly balanced (whp). In the next section, we define a Polya urns process, which models the degrees of the components, and analyze it to prove one of our main theorems, Theorem 13, which states that at the end of step (2), the total degrees of any two components differ by at most a multiplicative factor of $(1+\epsilon)$ (whp).

Theorem 13 can be used to prove the final two properties hold (whp), because if we can show that total degree is roughly the same for each component at the end of step (2), then it is not hard to apply Azuma's inequality (see the analysis in Section 3.4.3 or Section 3.5.3 for how to apply Azuma's inequality) to show that the total degree of each component remains roughly balanced within a constant (whp) throughout step (3). Furthermore, if the components remain balanced within a multiplicative constant throughout step (3), then at the end of step (3), each component must have total degree $\Theta(n^{1/2+\epsilon})$. Therefore, each component contains at most $O(n^{1/2+\epsilon})$ nodes, and we can conclude property 1 holds (whp) with $a = \Theta(n^{1/2+\epsilon})$, provided Theorem 13 holds.

Moreover, Theorem 13 can also be used to show that property 4 holds (whp). Recall that property 4 states that each pair of components must intersect at some node. If we view these $\Theta(n^{1-2\epsilon})$ pairs of intersections as coupons we wish to collect, then step (3) provides us with $\Theta(n)$ opportunities to collect $\Theta(n^{1-2\epsilon})$ coupons. Even though we are not collecting coupons uniformly at random, Theorem 13 (and Azuma's inequality) implies that the degrees of the components remain balanced throughout step (3) (whp). Furthermore, this implies that each coupon is still collected with probability at least $\Omega(n^{-(1-2\epsilon)})$, and it is not hard to see that given $(\frac{n}{2})$ opportunities, our process collects all coupons (intersections) within the required steps (whp). Therefore, as long as we can prove Theorem 13, all four properties of our linked decomposition hold with high probability. ■

3.3 Polya Urns with the Power of Choice

To prove that the total degrees of the components become balanced, we analyze an equivalent random process $\mathcal{P}$ on n bins, where each bin represents a component and each bin's load represents the size of a component. (Here, n should be thought of as the number of components, called c in the definition of linked decomposition, and should not be confused with the number of nodes in our graph). Analogous to the preferential attachment process, our random process starts with $2mn$ balls distributed arbitrarily among n bins, such that each bin contains at least m balls each. Furthermore, it is known that with high probability our process starts with at most $O(n^{1/2+\epsilon})$ balls in each bin [FFF05], where $\epsilon > 0$ can be arbitrarily small, a fact which will be useful later on. At each step of our random process $\mathcal{P}$, $2m$ new balls are thrown into the n bins, according to the following rule:

- Pick m bins independently at random, with probability proportional to the bin load

  (I) Throw 1 ball into each of the m random bins picked

  (II) Throw m more balls into the least loaded of the m random bins picked (break ties arbitrarily).

We can prove the following theorem about our random process $\mathcal{P}$, for any arbitrarily small $\epsilon > 0$, provided m is a sufficiently large constant:

Theorem 13 *Given the starting conditions described above, any time after $\Omega(n^{2+\epsilon})$ iterations of our process $\mathcal{P}$, the loads of any two bins differ by a multiplicative factor of at most $(1+\epsilon)$ (whp).*

Remark 1 *Note that proving Theorem 13 is sufficient for proving that properties (1) and (4) of linked decomposition hold (whp), as we described previously, and completes the proof of Theorem 12.*

Even though step (II) of our random process tends to balance bin loads, proving Theorem 13 is a bit challenging since step (I) tends to preserve load imbalances. For these reasons, we study a simpler random process $\bar{\mathcal{P}}$ on n bins, which we call *Polya urns with the power of choice*. By proving $\bar{\mathcal{P}}$ balances bin loads (whp), we illustrate the main techniques needed to prove $\mathcal{P}$ produces balanced bin loads (Theorem 13). Our new process, Polya urns with the power of choice $\bar{\mathcal{P}}$, starts with n nonempty bins containing $N_0 = \hat{c}n$ balls total. At each step of our random process $\bar{\mathcal{P}}$, we throw one more ball into one of the n bins according to the following rule:

(a) Pick m bins i.i.d. at random, with probability proportional to bin load

(b) Throw 1 ball into the least loaded of the m random bins picked

We can prove the following theorem about Polya urns with the power of choice, for any $\epsilon > 0$, provided m is a sufficiently large constant:

Theorem 14 *Given the starting conditions described above, any time after $\Omega(n^{2+\epsilon})$ balls have been thrown, the loads of any two bins differ by a multiplicative factor of at most $(1+\epsilon)$ (whp).*

To be more precise, we show that if each bin starts with fractional load at least $\frac{1}{\hat{c}n}$ for $\hat{c} > 1$, then all bins become balanced within $(1+\hat{\epsilon})$ (whp) some time before

$O(n^{1/(1-(1-1/\hat{c})^{m-1})+1+\hat{\epsilon}})$ new balls arrive, and they stay balanced within $(1+\hat{\epsilon})$ (whp) at all later times, where $\hat{\epsilon} > 0$ can be arbitrarily small, provided n is large enough. Note that for sufficiently large m or sufficiently small $\hat{c}$, we can conclude the bin loads balance in $O(n^{2+\epsilon})$ steps (whp) for any $\epsilon > 0$.

In the next subsection, we provide a sketch of Theorem 14, and defer exact details to Section 3.4. Additionally, we defer the full proof of Theorem 13 to Section 3.5. The proof for Theorem 13 follows roughly the same steps as the proof for Theorem 14, although the details are slightly different, since our random process is slightly different.

3.3.1 An Alternate Random Process

To prove Theorem 14, we start by defining a new random process $\bar{\mathcal{P}}_0$ which is somewhat easier to analyze than the Polya urn process with the power of choice $\bar{\mathcal{P}}$. Our new random process $\bar{\mathcal{P}}_0$ also throws balls into bins one at a time, and for each ball thrown, m random bins are generated in the same manner as described for $\bar{\mathcal{P}}$. However, our new process $\bar{\mathcal{P}}_0$ sometimes ignores the power of m choices and throws a ball into a bin that is not the least loaded of the m bins. Eventually, our goal will be to show that for any arbitrarily small $\epsilon > 0$, after $N_F = \Theta(n^{2+\epsilon})$ balls are thrown according to $\bar{\mathcal{P}}_0$, the number of balls in the bins are within a factor of $(1+\epsilon)$ from one another, with high probability.

It is easy to see that, if $\vec{a}(t)$ represents the bin loads of the Polya urns process with the power of choice $\bar{\mathcal{P}}$ at time t and $\vec{b}(t)$ represents the bin loads of the new process $\bar{\mathcal{P}}_0$ (or any process that sometimes ignores the power of m choices) at time t, then there exists a coupling of the two processes such that $\vec{a}(t)$ always majorizes $\vec{b}(t)$; we omit the formal details of the coupling, which are not difficult. Hence, to prove the theorem, we only need to analyze our new random process $\bar{\mathcal{P}}_0$, which throws balls in two phases, defined below.

In phase A, we throw $N_A = \Theta(n^{2+\epsilon_0})$ balls into bins, where $\epsilon_0 > 0$ is an arbitrarily small constant. At any time in phase A, we classify the bins into two types of bins: A bin is considered to be *low* if it contains less than $\frac{c_1}{n}$ fraction of the balls, otherwise it is *high*. Here, c_1 is a constant greater than 1 to be defined later, dependent on ϵ_0. Given these two definitions, we throw each new ball into a bin as follows:

- If at least one of the random bins generated for the new ball is a low bin, then we

throw the new ball into the first low bin generated.

- If all the bins generated are high bins, then we throw the new ball into the first random (high) bin generated.

In phase B, we throw balls into bins until the bins contain a total of $N_B = \Theta(N_A^{1+\epsilon_0})$ balls. Let M be the $(\frac{n}{c_2})$th smallest bin at the end of phase A, where $c_2 > 1$ is a constant dependent on ϵ_0. We use bin M to classify the bins into three types of bins: A bin is considered to be a *middle* bin at the current time if it has obtained the same load as bin M, at some point in time in phase B, prior to or equal to the current time. A bin is *low* if it has never obtained the same load as bin M at any time during phase B, prior to the current time, and it currently contains strictly less load then bin M. The remaining bins, those that have so far always been strictly higher than bin M in phase B, are *high* bins. In Phase B, we throw each new ball into a bin as follows:

- We look at the random bins generated for the new ball, and we let $T \in$ {low, middle, high} be the lowest bin type generated.
- We throw a ball into the first bin generated of type T.

To complete the proof, we prove the following five claims about the two phases:

- At the end of phase A:

  Claim 1: Each bin contains at least $\omega(\log n)$ balls with high probability.

  Claim 2: Each bin contains at most $1.01(\frac{c_1}{n})$ fractional load with high probability.

- If each bin contains at least $\omega(\log n)$ balls and at most $1.01(\frac{c_1}{n})$ fractional load at the end of phase A, then with high probability:

  Claim 3: Every high bin becomes a middle bin sometime during phase B.

  Claim 4: Every low bin becomes a middle bin sometime during phase B.

  Claim 5: If a bin becomes a middle bin (i.e. obtains the same load as bin M) sometime during phase B, then at the end of phase B, the load of that bin is at

most $(1+\epsilon_1)$ times the load of bin M, and at least $(1-\epsilon_1)$ times the load of bin M, where $\epsilon_1 > 0$ can be arbitrarily small, provided n is sufficiently large.

Note that the five claims imply Theorem 14, and by making ϵ_1 and ϵ_0 arbitrarily small, we also make ϵ arbitrarily small. Proving the five claims requires applying a variety of probability theory techniques, but we only sketch the proofs here; the full details are in the next section.

To prove Claim 1, we first note that a coupling can be defined such that bin loads arising from the $\bar{\mathcal{P}}_0$ process majorize the bin loads arising from the standard Polya process, which does not utilize the power of choice. We can then utilize known results regarding the standard Polya urn process [CHJ03] to prove that each bin contains $\omega(\log n)$ balls with high probability.

To prove Claims 2 and 3, we show that (whp) any high bin H has probability $\leq \gamma p_t$ of obtaining a new ball, where $\gamma < 1$ and p_t is the fractional load of bin H at time t. Thus, the high bins suffer a *shrinkage condition*, which causes the fractional load of high bins to decrease (whp). Provided we can show that the fractional load decreases quickly enough, Claims 2 and 3 follow.

Similarly, (whp) any low bin L has probability $\geq \gamma p_t$ of obtaining a new ball, where $\gamma > 1$ and p_t is the fractional load of bin L at time t. Thus, the low bins suffer a *growth condition*, which can be used to prove Claim 4. Finally, Claim 5 can be proved by applying Azuma's inequality.

3.4 Proof of Theorem 14: Polya with the Power of Choice Balances Loads

3.4.1 Proof of Claim 2 for Theorem 14

The proof of claim 2 is long and divided into three subsections: the first and last subsections prove two key lemmas, and the main body of the proof is contained in Subsection 3.4.1.

Proof of First Lemma Needed for Claim 2

To prove Claim 2, we make use of the following lemma, stating that the fractional load of a bin decreases rapidly, provided the probability of throwing a ball into the bin is less than its fractional load:

Lemma 5 *Consider a set of n bins with $\hat{N}_0$ balls total at time 0, and let i be any bin that starts with at least $\omega(\log n)$ balls. Let be p_t be the fraction of balls in bin i at time t, and consider any process that throws balls into bins such that the following* shrinkage condition *holds for bin i for some constant $c < (\frac{1}{1.01})$, and at any time t when the system contains less than or equal to $\beta \equiv \hat{N}_0 \delta^{(\frac{1}{1.01c-1})}$ balls:*

$$\Pr[\textit{Throwing a ball into bin } i \textit{ at time } t] \leq cp_t$$

where $\delta \in (0,1)$. Then the fractional load of bin i becomes less than or equal to δp_0, at some time before the bins obtain strictly more than β balls, with probability at least $(1 - \frac{1}{n^\alpha})$, where $\alpha > 0$ can be any arbitrarily large constant, provided n is sufficiently large.

To prove the lemma, we start by showing the fractional load of bin i decreases with high probability, when $\epsilon\hat{N}_0$ balls are thrown, for some small constant $\epsilon > 0$. To show this first claim, we define the event $\mathcal{E}$ to be the event that bin i never obtains more than $(1+\epsilon)p_0$ fraction of balls while the first $\epsilon\hat{N}_0$ balls are thrown, and contains at most $p_0(e^{(1.01c-1)\epsilon})$ fraction of balls after $\epsilon\hat{N}_0$ balls are thrown. Note that if we can prove event $\mathcal{E}$ happens with high probability, then this implies the fractional load decreases with high probability, since $c < (\frac{1}{1.01})$. To prove event $\mathcal{E}$ happens with high probability, consider an alternative process that throws $\epsilon\hat{N}_0$ balls, such that each ball has exactly $c(1+\epsilon)p_0$ probability of landing in bin i. A simple coupling argument can show that the probability event $\mathcal{E}$ happens for this new process must be less than or equal to the probability that event $\mathcal{E}$ happens for our original balls and bins process. The coupling exists more or less because if both the shrinkage condition and $\mathcal{E}$ hold for the first $\epsilon\hat{N}_0$ balls thrown in our original process, then the probability of throwing each ball into bin i is upper bounded by $c(1+\epsilon)p_0$. Thus, if we can prove $\mathcal{E}$ happens with high probability for this alternative process, it must also happen with high probability for our original process.

Now to analyze our alternative process, note that for the alternative process, the expected number of balls added to bin i is $(c(1+\epsilon)p_0)\epsilon\hat{N}_0$. Moreover, since bin i starts with at least $p_0\hat{N}_0 = \omega(\log n)$ balls, there exists a sufficiently large n such that the expected number of balls added is at least $\frac{4\alpha'}{\epsilon^2}\log n$, for any fixed $\epsilon > 0$ and $\alpha' > 0$. Thus, we can apply a standard Chernoff bound to show that less than or equal to $(1+\epsilon)(c(1+\epsilon)p_0)\epsilon\hat{N}_0$ balls are added to bin i, with probability at least $(1-\frac{1}{n^{\alpha'}})$, for any arbitrarily large $\alpha' > 0$, provided n is sufficiently large. Therefore, after $\epsilon\hat{N}_0$ balls are thrown, bin i contains at most $p_0\hat{N}_0 + (c(1+\epsilon)^2p_0)\epsilon\hat{N}_0$ balls with probability at least $(1-\frac{1}{n^{\alpha'}})$, and the fractional load of bin i does not exceed $p_0(1+c(1+\epsilon)^2\epsilon)$ with high probability while the $\epsilon\hat{N}_0$ balls are thrown. Thus, provided ϵ is sufficiently small, the fractional load of bin i does not exceed $p_0(1+\epsilon)$ with probability at least $(1-\frac{1}{n^{\alpha'}})$, and the first condition of event $\mathcal{E}$ holds with high probability. Furthermore, after $\epsilon\hat{N}_0$ balls have been thrown, the fractional load of bin i is at most $p_0(1+c(1+\epsilon)^2\epsilon)/(1+\epsilon) \leq p_0(e^{(1.01c-1)\epsilon})$ with probability at least $(1-\frac{1}{n^{\alpha'}})$, for sufficiently small $\epsilon > 0$. Therefore, we have shown that event $\mathcal{E}$ happens with probability at least $(1-\frac{1}{n^{\alpha'}})$.

We can now conclude that for our original process after $\epsilon\hat{N}_0$ balls have been thrown with the shrinkage condition holding, the fractional load of bin i decreases by a factor $e^{(1.01c-1)\epsilon}$ with high probability, while the total number of balls increases by a factor of $(1+\epsilon)$. Note that if the shrinkage condition holds while the system contains less than or equal to $\hat{N}_0(1+\epsilon)^r$ balls, then we can we repeat this analysis r times to conclude the fractional load decreases by a factor of $e^{(1.01c-1)\epsilon r}$ with probability at least $(1-\frac{r}{n^{\alpha'}})$, after the system obtains $\hat{N}_0(1+\epsilon)^r$ balls. Now take $r = (\frac{1}{\epsilon})(\frac{\log\delta}{1.01c-1})$, and we see that the fractional load of bin i decreases by a factor of δ with high probability, provided the shrinkage condition holds while the total number of balls is less than or equal to $\beta = \hat{N}_0\delta^{(\frac{1}{1.01c-1})}$. Finally, note that since we can make α' arbitrarily large, we can also make the previous statement hold with probability at least $(1-\frac{1}{n^{\alpha}})$, for any arbitrarily large $\alpha > 0$, provided n is sufficiently large. Therefore, we have proven our lemma. ■

Remark 2 *In the previous analysis, for simplicity, we ignored the subtle point that $\epsilon\hat{N}_0$ and the number of rounds r might not be integer. To complete the analysis more precisely, one needs to run the analysis for r' rounds, where r' is an integer, and choose a potentially different ϵ for each of the r' rounds, such that the number of balls thrown (e.g. $\epsilon\hat{N}_0$) is*

always integer. For notational purposes, suppose ϵ_i is the value we choose for ϵ to ensure that the number of balls thrown in each round i is integer. For large n, it is easy to see that we can make each ϵ_i small, and roughly the same value as a single fixed ϵ. Furthermore, if we choose our r' and ϵ_i, such that $\sum_{i=1}^{r'} \epsilon_i$ is close to $(\frac{\log \delta}{1.01c-1})$, then we can apply roughly the same argument as before to obtain our result, where $r' \approx (\frac{1}{\epsilon})(\frac{\log \delta}{1.01c-1})$ is an integer. For simplicity, we also ignore the same rounding issue that occurs in Lemma 8, but it can be avoided in a manner similar to the one just described.

Application of Lemma 5

To prove Claim 2, we would like to apply Lemma 5 to high bins in phase A to show that even if bin i is high and starts with fractional load close to 1, the shrinkage condition still holds for some constant c, and the fractional load eventually decreases to $(\frac{c_1}{n})$. Once a bin decreases to fractional load $(\frac{c_1}{n})$, we need to show the load stays below $1.01(\frac{c_1}{n})$ with high probability. We omit proving this second point, but it can be done with the same techniques used to prove Lemma 5. To prove the first point, our goal is to apply Lemma 5 with $\delta = (\frac{c_1}{n})$. To apply the lemma, note that if there are at least $\omega(n \log n)$ balls in the bins, then in phase A the definition of high bin implies that each high bin contains at least $\omega(\log n)$ balls. Therefore, we can apply our lemma to any high bin, provided we set $\hat{N_0} = \omega(n \log n)$.

Now, if we can make sure the shrinkage condition holds for an arbitrarily small constant c, then our lemma shows that the fractional load of any high bin decreases to $(\frac{c_1}{n})$ with high probability sometime before $N_A = \Theta(n^{(2+\epsilon_0)})$ total balls have been thrown, where ϵ_0 can be arbitrarily small, provided c can be made arbitrarily small. In order to show that c can be made arbitrarily small, note that the probability of throwing a ball into a high bin at time t is equal to $(h_t)^m \cdot (\frac{p_t}{h_t}) = (h_t)^{m-1} \cdot p_t$, where h_t is the total fractional load of the high bins at time t and k is the number of choices allowed. Thus, if we can show that with high probability h_t is strictly less than 1, then we can show that c can be made arbitrarily small with high probability, provided we allow a sufficient number of random choices m. Note that in order to apply our lemma, we need to show the previous statement is true with high probability for any time t in phase A after the bins contain at least $\hat{N_0} = \omega(n \log n)$ balls.

To complete the proof, we now just need to prove that h_t is strictly less than 1 with high probability in phase A after the bins contain at least $\hat{N_0} = \omega(n \log n)$ balls. To prove this, first note that at any time in phase A at most $(\frac{1}{c_1})$ fraction of the bins are high bins, and at least $(1 - \frac{1}{c_1})$ fraction of bins are low bins. Furthermore, the low bins begin with at least a constant fraction of the load when the process starts with $N_0 = O(n)$ balls, since at least $(1 - \frac{1}{c_1})$ fraction of bins are low bins, and since each bin contains at least 1 ball. To show that the low bins continue to have at least some constant fraction of the load, requires another lemma, Lemma 6, which we prove in the next subsection.

Proof of Second Lemma Needed for Claim 2

Lemma 6 *Let N_0 balls be distributed among n bins, such that every subset of cn bins contains at least $c'N_0$ balls, where $c, c' \in (0,1)$ are constants. Define $f(c') \equiv (c' + (1 - c')\ln(1 - c'))$ and let ϵ be any small constant in $(0, \hat{\epsilon}]$, where $\hat{\epsilon} > 0$ is a small constant defined in the proof. Then after $t \geq \frac{6N_0}{\epsilon^2(1-\epsilon)f(c')} = \Theta(N_0)$ balls have been thrown according to the standard Polya urns process, every subset of cn bins contains at least $(1 - \epsilon)^3 f(c')$ balls with probability at least $(1 - \frac{1}{n^\alpha})$, where $\alpha > 0$ can be arbitrarily large, provided n is sufficiently large.*

Remark 3 *Note that over the interval $(0,1)$, $f(c')$ is strictly positive, increasing, and has range $(0,1)$. Furthermore, a simple coupling argument shows that the theorem also holds for any Polya process that sometimes uses the power of multiple choices.*

Although our original process starts with N_0 balls distributed among n bins, it will be easier to analyze a standard Polya process that starts with N_0 balls distributed evenly among $n' \equiv N_0$ bins such that there is one ball in each of the n' bins. Note that our new Polya process on n' bins can be used to simulate the random loads that occur in our original Polya process with n bins. For a bin i that starts with b_i balls in our original process, we represent bin i by using b_i distinct bins from the new Polya process on n' bins. Namely, we

define the current load of bin i to be the sum of the current loads of the b_i bins that are used to represent bin i. It is not difficult to show that the loads we have defined, based on the new process on n' bins, are equivalent in distribution to the random loads generated by the original Polya process on n bins.

In order to lower bound the loads of any subset of cn bins in the original process on n bins, first note that any subset of cn bins is represented by at least $c'n'$ bins in our new process on n' bins. Furthermore, if we can lower bound the total fractional load on any subset of $c'n'$ bins for our new process with a sufficient exponentially high probability, then we can also lower bound the total fractional load on any subset of cn bins in our original process with high probability.

Following this idea, we now analyze the standard Polya process on n' bins starting with one ball each, and show that a fixed subset of $c'n'$ bins has exponentially high probability of containing at least $(1-\epsilon)^3 f(c')$ fractional load after $t \geq \frac{6n}{\epsilon^2(1-\epsilon)f(c')}$ balls have been thrown. From previous work on the Polya urn model [CHJ03], we know that when n' bins start with one ball each and t more balls are thrown, the random bin loads that are generated are equivalent in distribution to the loads generated by the following random process:

1. Pick $(n'-1)$ points uniformly at random from the interval $[0,1]$, and define x_i to be the position of the ith lowest point generated for $i \in \{1,...,(n'-1)\}$. For notational purposes, define $x_0 = 0$ and $x_{n'} = 1$.

2. Pick t points uniformly at random from the interval $[0,1]$, and define the load of bin i to be number of points that fall in the interval (x_{i-1}, x_i) plus 1, for $i \in \{1,...,n'\}$.

By analyzing this alternate process over $[0,1]$, we can lower bound the fractional load of a fixed subset of $z \equiv c'n'$ bins with high probability. We start by defining Y to be the sum of the lengths of the smallest $c'n'$ intervals defined by the $(n'-1)$ points generated by our alternate process. If we can lower bound Y with high probability, then we are not far from lower bounding the the fractional load of any fixed subset of $c'n'$ bins with high probability. To start, one can show that Y is equivalent in distribution to:

$$\frac{\sum_{i=0}^{z-1}((z-i)X_{n'-i})}{\sum_{i=0}^{n'-1}((n'-i)X_{n'-i})}$$

where each X_j variable denotes an independent exponential random variable with rate j.

To lower bound Y with high probability, first define $X = \sum_{i=0}^{z-1}(z-i)X_{n'-i}$ to represent the numerator, and $X' = \sum_{i=0}^{n'-1}(n'-i)X_{n'-i}$ to represent the denominator. Now, we can lower bound Y by lower bounding X and upper bounding X'. To lower bound X, we can use an argument similar to the one used to prove Cheroff bounds. First note that $\mu \equiv \mathrm{E}[X] = \sum_{i=0}^{z-1}\left(\frac{z-i}{n'-i}\right)$, and consider any arbitrarily small fixed $\delta > 0$. We can show for negative t sufficiently close to 0:

$$\begin{aligned}
\Pr[X \leq (1-\delta)\mu] &= \Pr[e^{tX} \geq e^{t(1-\delta)\mu}] \\
&\leq \frac{\mathrm{E}[e^{tX}]}{e^{t(1-\delta)\mu}} \\
&\leq \frac{e^{t(1-\frac{\delta}{2})\mu}}{e^{t(1-\delta)\mu}} \\
&\leq e^{t(\frac{\delta}{2})\mu}
\end{aligned}$$

where going from the second line to the third line follows by upper bounding $\mathrm{E}[e^{tX}]$:

$$\begin{aligned}
\mathrm{E}[e^{tX}] &= \mathrm{E}[e^{t(\sum_{i=0}^{z-1}(z-i)X_{n'-i})}] \\
&= \prod_{i=0}^{z-1}\mathrm{E}[e^{t(z-i)X_{n'-i}}] \\
&= \prod_{i=0}^{z-1}\left(1 - t\left(\frac{z-i}{n'-i}\right)\right)^{-1} \\
&\leq \prod_{i=0}^{z-1} e^{t(1-\frac{\delta}{2})(\frac{z-i}{n'-i})} \\
&\leq e^{t(1-\frac{\delta}{2})\mu}
\end{aligned}$$

To go from the third line to the fourth line in the calculation above, we assume that t is negative and sufficiently close to 0.

Lastly, observe that $\sum_{i=0}^{z-1}\left(\frac{z-i}{n'-i}\right) + \sum_{i=0}^{z-1}\left(\frac{n'-z}{n'-i}\right) = z$, so that we can write μ in a more convenient form:

$$\begin{aligned}\mu &= \sum_{i=0}^{z-1} \left(\frac{z-i}{n'-i}\right) \\ &\approx z - (n'-z)(\ln n' - \ln (n'-z)) \\ &\approx c'n' - n'(1-c')\ln\left(\frac{1}{1-c'}\right) \\ &\approx n'(c' + (1-c')\ln(1-c'))\end{aligned}$$

By observing that $\mu = \Omega(n')$, we have therefore shown that $X > (1-\delta)\mu$ with exponentially high probability. It can be similarly shown that $X' < (1+\delta)n'$ with exponentially high probability, which means $Y > (1-\epsilon)f(c')$ with exponentially high probability, where $\epsilon \equiv 1 - (\frac{1-\delta}{1+\delta})$ can be made arbitrarily small by making $\delta > 0$ arbitrarily small. Now, recall that since Y is the sum of the smallest $c'n'$ intervals, we have actually shown that *every* subset of $c'n'$ bins is represented by intervals of total length at least $(1-\epsilon)f(c')$ with exponentially high probability. Furthermore, for any fixed set of $c'n'$ bins, if we throw balls according to our alternate process on the real interval $[0,1]$, then the probability that any ball thrown lands in one of the $c'n'$ bins is at least $(1-\epsilon)f(c')$.

Finally, consider exactly $t = \frac{6n'}{\epsilon^2(1-\epsilon)f(c')}$ balls being thrown. By applying a standard Chernoff bound, one can show that at least $(1-\epsilon)^2 f(c')$ fraction of the t balls thrown fall into the $c'n'$ bins with probability that at least $1 - e^{-3n'}$. Thus after $t = \frac{6n'}{\epsilon^2(1-\epsilon)f(c')}$ balls have been thrown, there are a total of $n'(1 + \frac{6}{\epsilon^2(1-\epsilon)f(c')})$ balls, and our $c'n'$ bins contain at least $n'(\frac{6(1-\epsilon)}{\epsilon^2})$ balls with probability at least $1 - e^{-3n'}$. Therefore, the fractional load of the $c'n'$ bins is at least $\frac{6(1-\epsilon)^2 f(c')}{\epsilon^2(1-\epsilon)f(c')+6} \geq (1-\epsilon)^3 f(c')$ with exponentially high probability, after $t = \frac{6n'}{\epsilon^2(1-\epsilon)f(c')}$ balls have been thrown. Note, the last inequality only follows provided ϵ is sufficiently small, which is the reason we require $\epsilon \leq \hat{\epsilon}$ for some small $\hat{\epsilon} > 0$. Additionally, note that although we proved the previous statement for $t = \frac{6n'}{\epsilon^2(1-\epsilon)f(c')}$, it is not hard to argue that the lower bound also holds when $t \geq \frac{6n'}{\epsilon^2(1-\epsilon)f(c')}$, by showing that the lower bound value only increases for higher values of t.

Lastly, note that there are only $2^{n'}$ subsets of n' bins, so we can apply a naive union bound to conclude that every subset of more than $c'n'$ bins contains at least $(1-\epsilon)^3 f(c')$ fraction of balls with probability at least $1 - e^{n'}$. Thus, it follows that after $t \geq \frac{6N_0}{\epsilon^2(1-\epsilon)f(c')}$ balls have been thrown in our original process on n bins, every subset of cn bins contains at least $(1-\epsilon)^3 f(c')$ fractional load with probability at least $(1 - \frac{1}{n^\alpha})$, where $\alpha > 0$ can be arbitrarily large, provided n is sufficiently large. ■

3.4.2 Proof of Claim 1 for Theorem 14

To prove Claim 1, we follow the same steps used to prove Lemma 6. For conciseness, we only sketch the main details. We start by lower bounding the number of balls in the least loaded bin when balls are thrown according the standard Polya process starting with $n' = N_0$ bins each with one ball each. By lower bounding the load of the least loaded bin for the standard Polya process on n' bins, a simple coupling argument shows that this also lower bounds the load of the least loaded bin for our original Polya process with the power choice on n bins. To lower bound the load of the least loaded bin for the standard Polya process on n' bins, we analyze the same alternative random process over the interval $[0, 1]$ as described in Lemma 6, used to generate the random bin loads. If we define the random variable X to be size of the smallest interval generated with $n'-1$ points are thrown uniformly over the interval $[0, 1]$, then it can be shown that X is equivalent in distribution to $X_{n'}/\sum_{i=0}^{n'-1}((n'-i)X_{n'-i})$, where each X_j is an exponential random variable with rate j. Then we can show:

$$\begin{aligned}\Pr\left[X \leq \frac{1.01}{(n')^{-(2+\epsilon_0/2)}}\right] &\leq \Pr\left[\sum_{i=0}^{n'-1}((n'-i)X_{n'-i}) \geq 1.01n'\right] + \Pr\left[X_{n'} \leq \frac{1}{(n')^{-(1+\epsilon_0/2)}}\right] \\ &\leq 0.01n^{(\epsilon_0/2)} + n^{(\epsilon_0/2)} \leq 1.01n^{(\epsilon_0/2)}\end{aligned}$$

Thus all bins are represented by intervals of size at least $\Omega((n')^{-(2+\epsilon_0/2)})$ with high probability (although the probability exponent is low). Furthermore, since $\Theta((n')^{2+\epsilon_0})$ balls are thrown, it is easy to apply a standard Chernoff bound to show that each bin obtains at least $\Theta((n')^{\epsilon_0/2})$ balls with high probability. Then our previous coupling argument implies that each of the n bins in our original process contains at least $\Omega(n^{\epsilon_0/2}) = \omega(\log n)$ balls with high probability, proving Claim 1.

3.4.3 Proof of Claim 5 for Theorem 14

To prove Claim 5, we start by proving the following lemma:

Lemma 7 *Let A and B be two bins each with $a_0 = b_0$ balls starting at time 0. For any time $t \geq 0$, let a_t and b_t represent the number of balls at time t in bin A and bin B, respectively, and consider a random process that throws a new ball at time t into bin A with probability $(\frac{a_t}{a_t+b_t})$, and bin B with probability $(\frac{b_t}{a_t+b_t})$. Then for any $T \geq 0$, $\Pr[((\frac{1}{2}-\epsilon)/(\frac{1}{2}+\epsilon)) \cdot b_T \leq a_T \leq ((\frac{1}{2}+\epsilon)/(\frac{1}{2}-\epsilon)) \cdot b_T] \geq 1 - 2e^{-\epsilon^2(a_0+b_0-1)/2}$.*

To prove the lemma, we define a sequence of random variables $X_t = \left(\frac{a_t}{a_t+b_t}\right)$, which is a martingale since:

$$\begin{aligned} \mathrm{E}[X_{t+1}|\ X_t] &= \mathrm{E}[X_{t+1}|\ a_t, b_t] \\ &= \left(\frac{a_t}{a_t+b_t}\right)\left(\frac{a_t+1}{a_t+b_t+1}\right) + \left(\frac{b_t}{a_t+b_t}\right)\left(\frac{a_t}{a_t+b_t+1}\right) \\ &= \left(\frac{a_t}{a_t+b_t}\right) = X_t \end{aligned}$$

Furthermore, note that at time t moving one ball from bin A to bin B or vice versa, changes X_t by $(\frac{1}{a_t+b_t})$. Thus, it is easy to see that $|X_{t+1} - X_t| \leq (\frac{1}{a_t+b_t}) = (\frac{1}{a_0+b_0+t})$. Now we can apply Azuma's inequality obtain $\Pr[X_T \geq \frac{1}{2} + \epsilon] \leq e^{-\epsilon^2/(2\sum_{i=0}^{T-1}(a_0+b_0+i)^{-2})} \leq e^{-\epsilon^2(a_0+b_0-1)/2}$, where the last inequality follows because $\sum_{i=0}^{T-1}(a_0+b_0+i)^{-2} \leq \int_{x=(a_0+b_0)}^{\infty}(x-1)^{-2} = (a_0+b_0-1)^{-1}$. Rearranging, we get $\Pr[a_T \geq ((\frac{1}{2}-\epsilon)/(\frac{1}{2}+\epsilon)) \cdot b_T] \leq e^{-\epsilon^2(a_0+b_0-1)/2}$. Similarly, we can also use Azuma's inequality to show $\Pr[a_T \leq ((\frac{1}{2}+\epsilon)/(\frac{1}{2}-\epsilon)) \cdot b_T] \leq e^{-\epsilon^2(a_0+b_0-1)/2}$, and the lemma follows. ■

Remark 4 *It is not hard to show the lemma also holds for any process that throws a ball at time t into bin A with probability $\gamma_t(\frac{a_t}{a_t+b_t})$, bin B with probability $\gamma_t(\frac{b_t}{a_t+b_t})$, and neither bin with probability $(1-\gamma_t)$, where $\gamma_t \in [0,1]$ can be a random variable dependent on past events.*

Now to complete the proof of Claim 5, we just need to apply Lemma 7 to bin M and any bin i that obtains the same load as bin M, starting at the time they obtain the same load. Note that we can apply Lemma 7 to our two bins due to the remark. Furthermore,

since bin M contains at least $\omega(\log n)$ balls by Claim 1, with high probability the load of that bin i is at most $(1+\epsilon_1)$ times the load of bin M, and at least $(1-\epsilon_1)$ times the load of bin M at the end of phase B, where ϵ_1 can be made arbitrarily small, provided we set ϵ small enough and n is large enough. Finally, note that for sufficiently large n the previous statement happens with high enough probability, so that we can take a naive union bound over all $n-1$ possible bins i to conclude that the previous statement happens for all bins i that obtain the same load as bin M. Therefore, Claim 5 holds with high probability.

3.4.4 Proof of Claim 3 for Theorem 14

To prove Claim 3, we start by following the steps used to prove Claim 2. For conciseness, we only sketch the main details. By following the same steps as Claim 2, one can show that the low and middle bins always contain a constant (dependent on c_2) fraction of the total load with high probability. Consequently, one can show that the shrinkage condition of Lemma 5 holds for some fixed constant dependent on c_2, for any high bin. From here, one can apply Lemma 5 to the high bins in phase B, with δ equal to a constant, to show that if any high bins remain after γN_A balls have been thrown, then they all contain at most $(\frac{1}{2})(\frac{1}{n})$ fractional load with high probability, where δ is a constant dependent on c_1, and γ is a constant dependent on δ and c_2.

The final step needed to prove Claim 3 is to show that with high probability it cannot be the case that there remains some set of high bins all with fractional load $(\frac{1}{2})(\frac{1}{n})$ with high probability. Claim 3 then follows because this implies that all high bins must have become middle bins at some point before $\Theta(N_A)$ balls have been thrown. To prove that with high probability it is not possible for some high bins to remain all with fractional load at most $(\frac{1}{2})(\frac{1}{n})$, we use a proof by contradiction. Note that if some high bins remain with fractional load at most $(\frac{1}{2})(\frac{1}{n})$, then bin M must have load at most $(\frac{1}{2})(\frac{1}{n})$. Furthermore, by applying the same analysis used in Claim 5, one can show that all the low and middle bins have fractional load at most $1.01(\frac{1}{2})(\frac{1}{n})$ with high probability. However, this implies the total fraction load of all bins is at most $1.01(\frac{1}{2}) < 1$ with high probability, which is a contradiction. Therefore, all high bins must have become middle bins sometime before γN_A balls have been thrown with high probability.

3.4.5 Proof of Claim 4 for Theorem 14

Before we can prove Claim 4, we need to prove the following lemma:

Lemma 8 *Consider a set of n bins with $\hat{N}_0 = O(poly(n))$ balls total at time 0, and let i be any bin that starts with at least $\omega(\log n)$ balls. Let be p_t be the fraction of balls in bin i at time t, and consider any process that throws balls into bins such that the following* growth condition *holds for bin i, for some constant $c > \frac{1}{.99}$, and at any time t with less than or equal to N_f total balls:*

$$\Pr[\textit{Throwing a ball into bin } i \textit{ at time } t] \geq cp_t$$

where $N_f \geq \hat{N}_0$ is a random integer stopping time. Then N_f must be less than $\beta \equiv \hat{N}_0^{1+(\frac{1}{.99c-1})}$ with probability at least $(1 - \frac{1}{n^\alpha})$, where $\alpha > 0$ can be any arbitrarily large constant, provided n is sufficiently large.

To prove the statement, we prove that if the growth condition does hold until β balls are in the system, then the fractional load of bin i becomes strictly greater than 1, with probability at least $(1 - \frac{1}{n^\alpha})$. Furthermore, since it is impossible for a bin to have fractional load more than 1, it must be the case $N_f \geq \beta$ happens with probability at most $\frac{1}{n^\alpha}$. Therefore, we can conclude N_f must be less than β with probability at least $(1 - \frac{1}{n^\alpha})$. To complete the proof, we now just need to show the fractional load of bin i becomes strictly greater than 1, with probability greater than $(1 - \frac{1}{n^\alpha})$, if the growth condition holds whenever there are less than or equal to β balls are in the system.

To prove this last statement, we start by showing that when $\epsilon\hat{N}_0$ balls are thrown, the fractional load of bin i increases to at least $p_0(e^{(.99c-1)\epsilon})$ with high probability, where $\epsilon > 0$ is some small constant. Note that even if no new balls are added to bin i, p_t is still lower bounded by $(\frac{p_0}{1+\epsilon})$ over this time period. In order to analyze our process, we first consider a simpler process that throws a ball into bin i with probability exactly $c(\frac{p_0}{1+\epsilon})$. Note that this new process can be coupled with our original process, so that the new process always adds fewer balls to bin i, while the $\epsilon\hat{N}_0$ balls are thrown. Thus, if we can say that for this new process, the fractional load of bin i increases to at least $p_0(e^{(.99c-1)\epsilon})$ with

high probability, then the fractional load also increases to at least $p_0(e^{(.99c-1)\epsilon})$ with high probability for our original process.

Now, if we look at this new process, the expected number of new balls added to bin i is $(\frac{cp_0}{1+\epsilon})\epsilon\hat{N}_0$, when $\epsilon\hat{N}_0$ balls are thrown. Furthermore, for any fixed $\epsilon > 0$ and constant $\alpha' > 0$, there exists a sufficiently large n such that the expected number of new balls added is at least $\frac{2\alpha'}{\epsilon^2}\log n$, since bin i starts with $p_0\hat{N}_0 = \omega(\log n)$ balls. Moreover, by applying a standard Chernoff bound, we can conclude the number of balls added to bin i is at least $(1-\epsilon)(\frac{cp_0}{1+\epsilon})\epsilon\hat{N}_0$ with probability at least $(1-\frac{1}{n^{\alpha'}})$. So the total number of balls in bin i is at least $p_0\hat{N}_0+(1-\epsilon)(\frac{cp_0}{1+\epsilon})\epsilon\hat{N}_0$ with high probability, and the fractional load of bin i is at least $p_0(1+c\epsilon\frac{1-\epsilon}{1+\epsilon})/(1+\epsilon)$ with high probability. For simplicity, we can lower bound the previous expression by $p_0(e^{(.99c-1)\epsilon})$, for sufficiently small ϵ. Therefore, by the previous coupling argument, we can conclude that for our original process, the fractional load increases by at least $p_0e^{(.99c-1)\epsilon}$ with probability at least $(1-\frac{1}{n^{\alpha'}})$.

Thus, after $\epsilon\hat{N}_0$ balls have been thrown, we have increased the fractional load of bin i by a multiplicative factor of $e^{(.99c-1)\epsilon}$ with high probability, and we have increased the total number of balls in the system by a multiplicative factor of $(1+\epsilon)$. Furthermore, we can repeat the same analysis r times, increasing the total number of balls to $\hat{N}_0(1+\epsilon)^r$, and increasing the fractional load of bin i by a factor of $e^{(.99c-1)\epsilon r}$ with probability at least $(1-\frac{r}{n^{\alpha'}})$, provided the growth condition holds for the new balls thrown over this time. Now, assuming that the growth condition holds until there are at least β balls in the system, we know that the growth condition holds over $r = (\frac{1}{\epsilon})(\frac{\log\hat{N}_0}{.99c-1})$ rounds, since $\hat{N}_0(1+\epsilon)^r \leq \hat{N}_0 \cdot \hat{N}_0^{1/(.99c-1)}$. Moreover, if the growth condition holds over $r = (\frac{1}{\epsilon})(\frac{\log\hat{N}_0}{.99c-1})$ rounds, then the fractional load of bin i increases by at least a factor of $\hat{N}_0$ with probability at least $(1-\frac{r}{n^{\alpha'}})$. Therefore, we can conclude the fractional load of bin i becomes strictly greater than 1 with high probability, since $p_0 > \frac{1}{\hat{N}_0}$. Lastly, note that since we can make α' arbitrarily large, we can also make the previous statement hold with probability at least $(1-\frac{1}{n^{\alpha}})$, for any arbitrarily large $\alpha > 0$, provided n is sufficiently large. Therefore, bin i obtains fractional load strictly greater than 1 with probability at least $(1-\frac{1}{n^{\alpha}})$, and by our previous reasoning, the lemma follows. ■

To complete the proof of Claim 4, we would like to apply Lemma 8 to the low bins in phase B. Note that by starting in phase B, we know that each low bin contains at least $\omega(\log n)$ balls with high probability, by Claim 1. So by setting with $\hat{N}_0 = N_A$, we automatically meet the first condition of the lemma with high probability. To complete the

proof, we need to show that for any low bin with fractional load p_t at time t in phase B, the probability of throwing a ball into the low bin is greater than or equal to cp_t, where $c > \frac{1}{.99}$ can be made arbitrarily large, provided c_2 can be arbitrarily large. By showing the growth condition holds for all low bins, we can apply Lemma 8 to show that with high probability the growth condition must fail to hold for the low bin before the bins obtain $\Theta(N_A^{(1+\epsilon_0)})$ balls. Claim 4 then follows immediately because the only way for the growth condition to stop holding is if the low bin obtains the same load as bin M and becomes a middle bin.

Therefore, we just need to show that the growth condition holds for low bins for arbitrarily large c, and then Lemma 8 implies Claim 4. To prove this, note that the probability of throwing a ball into the low bin at time t is equal to $\Pr[\text{Throw a ball into a low bin}](\frac{p_t}{l_t}) = (\frac{1-(1-l_t)^m}{l_t}) \cdot p_t$, where l_t is the fractional load of the low bins at time t. Thus, to complete the proof, we just need to show that by increasing c_2, l_t can be made arbitrarily small with high probability.

To complete the last step, we prove the total fractional load of the low bins is at most $\frac{1.01}{c_2-1}$ with high probability by showing that there are at most $(\frac{n}{c_2})$ low bins, and with high probability each low bin contains at most $1.01\frac{c_2}{n(c_2-1)}$ fractional load. The first part of the statement follows easily from the definition of low bin, and the second part of the statement follows by showing bin M contains at most $1.01\frac{c_2}{n(c_2-1)}$ fractional load with high probability. We can bound the the load of bin M, by noting that there are at least $(1-\frac{1}{c_2})n$ middle and high bins, whose total fractional load cannot exceed 1. This implies the fractional load of the least loaded middle or high bin must be less than or equal to $\frac{c_2}{n(c_2-1)}$. Furthermore, note that the analysis used to prove Claim 5, also shows that the fractional load of bin M has at most 1.01 times the load of the lowest middle bin with high probability, and bin M can never have more load than a high bin. Therefore, the fractional load of bin M is at most $1.01\frac{c_2}{n(c_2-1)}$ with high probability, which completes the proof of the last step.

Therefore, we have shown that every low bin eventually obtains the same load as bin M and joins the middle group with high probability sometime before the bins obtain $\Theta(N_A^{(1+\epsilon_0)})$ balls, where ϵ_0 can be arbitrarily small, by taking c_2 large enough.

3.5 Proof of Theorem 13: Component Sizes in Linked Decomposition Balance

We prove that our algorithm yields balanced component sizes (in terms of degree) with high probability when running on the preferential attachment model. To analyze the size of the components generated by our preferential attachment process, we analyze a random process on n bins, where each bin represents a component and the loads represent the size of the components. By looking at the preferential attachment process, we know that the random process starts with $2kn$ balls distributed arbitrarily among n bins, such that each bin contains at least k balls each. Furthermore, we also know that with high probability the process should start with at most $O(n^{1/2+\epsilon})$ balls each in bin [FFF05], a fact which will be useful later on. At each step of the process, $2k$ new balls are thrown into the n bins, according to the following rule:

- Pick k bins i.i.d. at random, with probability proportional to bin load

(I) Throw 1 ball into each of the k random bins picked

(II) Throw k more balls into the least loaded of the k random bins picked.

Let's call this random process $\mathcal{P}$. As before, we'll analyze another random process $\mathcal{P}_0$ which does not always throw the k balls in step (II) into the least loaded bin, but sometimes throws the k balls into a heavier bin. We can show that the random process $\mathcal{P}_0$ achieves roughly balanced loads (whp), and we can show a coupling exists where $\mathcal{P}$ always majorizes $\mathcal{P}_0$. Therefore, $\mathcal{P}$ also achieves roughly balanced loads (whp). We ignore the coupling argument and finish by defining $\mathcal{P}_0$ below and showing it becomes roughly balanced (whp).

Our random process $\mathcal{P}_0$, also throws balls in two phases, analogous to the two phases defined in section [cite]. For each phase, we throw the same number of balls as before, and we also choose where to throw the k balls in step (II) using the same rules as before, based on whether or not the bins selected are low, high, or middle. The only slight modification we employ is in phase B, where we say a low bin becomes a middle bin whenever it obtains load equal to *or greater than* the load of bin M, and a high bin becomes

a middle bin whenever it has obtains load equal to *or less than* the load of bin M. To prove the process $\mathcal{P}_0$ yields balanced loads, we prove the same five claims as before. The proof of each claim is roughly the same as before, but the details are slightly different as the two processes are slightly different.

3.5.1 Proof of Claim 1 for Theorem 13

To prove Claim 1, we analyze a new random process $\mathcal{P}_1$ on n bins, which starts with the same initial load distribution as the random process $\mathcal{P}_0$, but adds balls to bins in a slightly different manner. At each step of the process, $2k$ new balls are thrown into the n bins, according to the following rule:

- Pick k bins i.i.d. at random, with probability proportional to bin load

(I) Throw 1 ball into each of the random bins picked

(II) Throw k more balls into the last random bin picked

A simple coupling can be defined so that $\mathcal{P}_0$ always majorizes $\mathcal{P}_1$, so that we just need to lower bound the load of the least loaded bin when running the random process $\mathcal{P}_1$, in order to prove lower bounds on the load of the least loaded bin when running the random process $\mathcal{P}_0$. We omit the coupling argument for conciseness, but finish by proving that all bins contain at least $\Omega(\log n)$ balls with high probability after throwing $\Theta(n^{2+\epsilon})$ balls when running the random process $\mathcal{P}_1$, where $\epsilon > 0$ can be arbitrarily close to 0.

To prove the previous statement, we show that after throwing $\Theta(n^{2+\epsilon})$ balls, any fixed bin has $\Omega(\log n)$ balls with probability at least $1 - n^{-\alpha}$, where $\alpha > 0$ can be arbitrarily large, provided we can set k arbitrarily large. Our claim then follows by applying a simple union bound, which implies that after throwing $\Theta(n^{2+\epsilon})$ balls, all bins have load at least $\Omega(\log n)$ with probability at least $1 - n^{-\alpha+1}$.

To analyze the load of a single bin B, note that after t iterations of our process (i.e. after $2kt$ new balls have been thrown), bin B is picked as a random bin in the next iteration with probability at least $\frac{k}{2kn+2kt} = (\frac{1}{2})(\frac{1}{n+t})$. Now consider the random variable $Y \equiv \sum_{t=0}^{T} \sum_{j=1}^{k} X_{t,j}$, where $X_{t,j}$ are bernoulli random variables with success probability

$(\frac{1}{2})(\frac{1}{n+t})$, for $t \in \{0, \ldots, T\}$, $j \in \{1, \ldots, k\}$. It is easy to that Y is stochastically dominated by the number of balls thrown by step (I) of our process, and furthermore can be used to lower bound the number of balls added to bin B. Note that for $T = n^{2+\epsilon}$, $\mathrm{E}[Y] = \frac{k-1}{2}\sum_{t=0}^{T}(\frac{1}{n+t}) \approx \frac{k-1}{2}(1+\epsilon)\log n$. Now, we can apply a standard Chernoff bound on Y to conclude the at least $\Omega(\log n)$ balls are added to bin B with probability at least $1 - n^{-\alpha}$, where α can be made arbitrarily large, provided k can be arbitrarily large. Therefore, we have shown that after throwing $\Theta(n^{2+\epsilon})$ balls, all bins have load at least $\Omega(\log n)$ with probability at least $1 - n^{-\alpha+1}$.

3.5.2 Proof of Claim 2 for Theorem 13

Lemma 3 for Theorem 13

Consider running the random process $\mathcal{P}_1$ defined above in Claim 1, for $T_f = O(poly(n))$ number of steps. Assuming we have a constant $c \in [\frac{1}{2}, 1)$, which is sufficiently close to 1, we can prove the following lemma about the process above:

Lemma 9 *The lowest cn bins always contain at least $\frac{1}{8}$ fraction of the load, during all T_f iterations of the random process, with high probability.*

Remark 5 *A simple coupling argument implies that this statement also holds for our original random process $\mathcal{P}_0$.*

To prove the lemma, let's consider a fixed subset of cn bins. Let X_t denote the fraction of balls in cn bins after t iterations of our process (i.e. after $2kt$ new balls have been thrown). For notation, let $N_t \equiv 2kn + 2kt$ be the total number of balls in the n bins after iteration t, and let B_t represent the number of new balls added to our subset of cn bins in the tth iteration. Note that X_t is a martingale, since:

$$\begin{aligned} \mathrm{E}[X_{t+1}|\ X_t] &= \frac{X_t \cdot N_t + \mathrm{E}[B_{t+1}|X_t]}{N_t + 2k} \\ &= \frac{X_t \cdot N_t + 2k \cdot X_t}{N_t + 2k} \\ &= X_t \end{aligned}$$

Since X_t is a martingale, we make use of Azuma's inequality. Note that $c_t \equiv |X_{t+1} - X_t| \leq (\frac{2k}{N_t}) = (\frac{1}{n+t})$ and $\sum_{t=0}^{\infty} {c_t}^2 \leq \sum_{t=0}^{\infty}(n+t)^{-2} \leq \int_{x=n}^{\infty}(x-1)^{-2} = (n-1)^{-1}$. Furthermore, $X_0 \geq \frac{cnk}{2nk} = \frac{c}{2} \geq \frac{1}{4}$. Therefore, we can apply Azuma's inequality to obtain $\Pr[X_T \leq \frac{1}{8}] \leq e^{-(n-1)/128}$, for any fixed time T, which means that any fixed subset of cn bins has probability at most $e^{-(n-1)/128}$ of having less than $\frac{1}{8}$ load at time T. Provided c is sufficiently close to 1, we can then apply a union bound over all subsets of bins of size cn, and over all times $T \in \{0, \ldots, T_f\}$ to conclude that the lowest cn bins always have fractional load at least $\frac{1}{8}$ with high probability.

Using Lemma 3

Note that Lemma 3 (9) also lower bounds the fractional load of the lowest cn bins when running the $\mathcal{P}_0$ process, via a simple coupling/majorization argument. Lemma 3 is useful for proving Claim 2 because a lower bound the fractional load of the low bins in phase A of the $\mathcal{P}_0$ process can be used to upper bound the probability that balls get added to the high bins in phase A.

To start our proof, note that at most $\frac{1}{c_1}$ fraction of the bins have fractional load greater than or equal to $\frac{c_1}{n}$, which implies that at least $(1 - \frac{1}{c_1})$ fraction of bins are low bins. Thus, if we make c_1 sufficiently large and take $c = (1 - \frac{1}{c_1})$, then we can use Lemma 3 to conclude with high probability that the low bins in phase A always contain at least $\frac{1}{8}$ of the load and the high bins in phase A contain at most $\frac{7}{8}$ fractional load.

Now let p_t represent the fractional load of some high bin H at time t and let h_t represent the total fractional load of all high bins at time t. It is not hard to see the expected number of balls that get added to high bin H in phase A at time t equals $kp_t + k(h_t)^k(\frac{p_t}{h_t})$, which is upper bounded by $2k \cdot p_t(\frac{1}{2} + (\frac{7}{8})^{k-1})$ with high probability. Thus, we have shown with high probability that the fraction of $2k$ new balls that get added to bin H at time t

is less than or equal to γp_t, in expectation, where $\gamma = (\frac{1}{2} + (\frac{7}{8})^{k-1}) \approx \frac{1}{2}$ for large k. In other words, in expectation bin H is only obtaining $\gamma p_t \approx (\frac{1}{2})p_t$ fraction of the $2k$ new balls thrown at each time step, which should cause the fractional load of high bin H to decrease rapidly.

So in essence, the high bin H satisfies a *shrinkage condition* analogous to the one described in Lemma 2, with shrinkage constant $\gamma \approx \frac{1}{2}$. Furthermore, note that for our modified Polya urns process $\mathcal{P}_0$, we can assume that each bin starts with at most $O(n^{\frac{1}{2}+\epsilon})$ balls (or in other words $O(n^{-1/2+\epsilon})$ fractional load) with high probability [FFF05], where $\epsilon > 0$ can be arbitrarily small. Thus, the fractional load of bin H must only decrease by a factor of $\delta = \Theta(n^{-1/2-\epsilon})$ to ensure H has fractional load at most $\frac{c_1}{n}$. From here, we can essentially follow the same steps used in proving Lemma 2, to prove that bin H obtains load $\frac{c_1}{n}$ with high probability, sometime before at most $\beta = \hat{N}_0 \delta^{(\frac{1}{(1+\epsilon)\gamma-1})}$ balls are thrown by our process, where $\hat{N}_0$ is the total number of balls that start in our bins and $\epsilon > 0$ can be arbitrarily small. We omit the proof for conciseness, but it can also be proved using a series of Chernoff bounds as before.

Note that since γ can be arbitrarily close to $\frac{1}{2}$ for k sufficiently large and ϵ arbitrarily close to 0, then our new version of Lemma 2 implies that bin H obtains fractional load $\frac{c_1}{n}$ sometime before $\beta = O(\hat{N_0}^{2+\epsilon_0})$ balls are thrown. After bin H obtains fractional load $\frac{c_1}{n}$, it is not hard to show that with high probability bin H always maintains fractional load less than $1.01(\frac{c_1}{n})$ until the end of phase A, thus proving Claim 2.

3.5.3 Proof of Claim 5 for Theorem 13

To prove Claim 5, we follow the same steps as the original proof of Claim 5. Note that if a bin B first becomes a middle bin at time T_0 (i.e. after $2kT_0$ new balls have been thrown), then $|b_{T_0} - m_{T_0}| \leq 2k$, where we use the notation b_t to represent the number of balls in bin B at time t, and we use m_t to represent the number of balls in the "yardstick" middle bin M at time t. Now if we define $X_t = \frac{b_t}{m_t + b_t}$, then we know $X_{T_0} \in [\frac{1}{2} - \epsilon, \frac{1}{2} + \epsilon]$ (whp), where $\epsilon > 0$ can be arbitrarily small provided n is sufficiently large, since (whp) $m_{T_0} = \Omega(\log n)$ and $|b_{T_0} - m_{T_0}| \leq 2k$. Furthermore, it is not hard to show X_t is a martingale for $t \geq T_0$, and we can apply Azuma's inequality as before to prove that at the end phase B, $X_t \in [\frac{1}{2} - 2\epsilon, \frac{1}{2} + 2\epsilon]$, where $\epsilon > 0$ can be arbitrarily small. Therefore, we have proved

Claim 5.

3.5.4 Proof of Claim 4 for Theorem 13

To prove Claim 4, we follow the same steps as the original proof of Claim 4. We can show the total fractional load of the low bins is at most $\frac{1.01}{c_2-1}$ (whp), following the same steps detailed in the next to last paragraph of the original proof of Claim 4. Now, let p_t represent the fractional load a low bin L at time t of our process $\mathcal{P}_0$ (i.e. after $2kt$ new balls have been thrown in t iterations), and let l_t represent the total fractional load of all lows bins at time t. Then, the expected number of new balls added to bin L on the $t+1$th iteration of $\mathcal{P}_0$ is $kp_t + k(1-(1-l_t)^k)(\frac{p_t}{l_t})$, and (whp) this is greater than or equal to $k(\frac{1}{2})(\frac{p_t}{l_t}) \geq kp_t(\frac{c_2-1}{3})$, provided $c_2 \geq 4$.

Thus, we have shown with high probability that the expected fraction of the $2k$ new balls that get added to bin L at time t is greater than or equal to γp_t, where $\gamma = \frac{c_2-1}{6}$. In other words, (whp) the low bin L is satisfying a *growth condition* analogous to the one described in Lemma 5, where bin L obtains in expectation least $\gamma p_t = (\frac{c_2-1}{6})p_t$ fraction of the $2k$ new balls thrown at each time step. The growth condition implies the fractional load of bin L increases rapidly, and we can follow the same steps as in Lemma 5 to prove that bin L becomes a middle bin within $\beta \equiv \hat{N_0}^{1+(\frac{1}{.99\gamma-1})}$ iterations with high probability, where $\hat{N_0} = N_A$ is the total number of balls in the bins at the start of phase B. Furthermore, note that γ can be made arbitrarily large by taking c_2 large enough, which implies that bin L becomes a middle bin sometime before $N_A^{1+\epsilon}$ balls are thrown, where ϵ can be arbitrarily small provided by choosing c_2 large enough. Finally, we note bin L becomes a middle bin with sufficiently high probability, so that a union bound implies that all low bins must become middle bins with high probability, thus proving Claim 4.

3.5.5 Proof of Claim 3 for Theorem 13

To prove Claim 3, we follow roughly the same steps as those used in the original proof of Claim 3. We start by proving that after $N_A^{1+\epsilon}$ new balls are thrown in phase B, the fractional load of the low and middle bins must be at least $\frac{1.01}{c_2-1}$ (whp). To prove this, we take all bins that start as low and middle bins in phase B, say bins $B_1, \ldots, B_i$, and

imagine a fictional *super* bin $\mathcal{L}$, which contains bins $B_1, \ldots, B_i$ and whose load is the sum of the loads of bins $B_1, \ldots, B_i$. Note that if the low and middle bins contain less than $\frac{1.01}{c_2-1}$ fractional load, then super bin $\mathcal{L}$ satisfies the same growth condition as described previously, with the same growth constant $\gamma = \frac{c_2-1}{6}$. Furthermore, one can again use Lemma 5 to show that within $N_A^{1+\epsilon}$ steps, $\mathcal{L}$ must have fractional load at least $\frac{1.01}{c_2-1}$ (whp). Therefore, the low and middle bins must contain fractional load at least $\frac{1.01}{c_2-1}$ (whp) after $N_A^{1+\epsilon}$ balls have been thrown.

As a result, the high bins must contain fractional load at most $(1-\frac{1.01}{c_2-1})$ after $N_A^{1+\epsilon}$ new balls have been thrown (whp), and therefore, we can show any high bin H, satisfies a shrinkage condition with shrinkage constant $\gamma = (\frac{1}{2}+(1-\frac{1.01}{c_2-1})^{k-1})$, which can be arbitrarily close to $\frac{1}{2}$. At this point in our process, it is not hard to show that each high bin still contains at most $1.01\frac{c_1}{n}$ fractional load (whp), and must become a middle bin before $\hat{c}N_A^{1+\epsilon}$ more balls are thrown (whp), for sufficiently large $\hat{c}$. The latter statement must be true because if H does not become a middle bin, then H must satisfy the shrinkage condition for $\hat{c}N_A^{1+\epsilon}$ steps. One can then use the same analysis from Lemma 2 to conclude the fractional load of bin H must decrease to $\frac{1}{2n}$ (whp). However, following the same argument as in last paragraph of the original proof of Claim 3, we can show that this cannot happen (whp), and bin H must become a middle bin (whp). Finally, a simple union bound implies that all high bins must become middle bins (whp) sometime before $O(N_A^{1+\epsilon})$ new balls are thrown in phase B.

3.6 Linked Decompositions with Exceptions

The proof in the previous section that $\mathcal{PA}(m)$ graphs have linked decompositions requires m to be a large constant, and the graphs produced have expected degree $2m$. The Internet, however, has average degree about four; as a result, the $\mathcal{PA}(m)$ model is not considered a realistic model of the Internet unless m is very small. Our next result holds when $m \geq 2$, and achieves a slightly better $a = \sqrt{n \log n}$, but it achieves a weaker form of linked decomposition. In particular, define a *linked decomposition with exceptions* and parameters a, b, c, d to be a decomposition satisfying the 4 requirements, except that each subgraph is allowed to contain one "promiscuous" node, which may belong to more than b subgraphs. It is not hard to see that the routing properties of the decomposition (see

Section 3.7) are preserved in the face of a single exception per component (the remaining nodes essentially "route around it"). We can show:

Theorem 15 *A graph G_n generated by the $\mathcal{P}A(2)$ model has, with high probability, a linked decomposition with exceptions and $a = \Theta(\sqrt{n \log n})$, $b = 2$, $c = \Theta(\sqrt{n/\log n})$, and $d = \log n$.*

We start by running $\mathcal{P}A(2)$ for $t = n/\log n$ steps, and then we decompose the resulting G_t into $\sqrt{n/\log n}$ small diameter subgraphs, each with a total degree (sum of the degrees of its nodes) that is between s and $3s$, where $s = \sqrt{n \log n}$. These subgraphs will be completely disjoint, except for at most one exceptional node in each.

Due to space limitations, we only provide a rough sketch regarding how to obtain such this starting decomposition. we by computing a breadth-first search tree of G_t starting from the first node, thus achieving $\log n/\log\log n$ diameter with high probability [BR02]. From here, it is not hard to show that one can start from the bottom of the tree and iteratively find connected subtrees of total degree between s and $2s$, until one subtree remains of total degree at most $3s$. These subtrees cover all nodes, and each subtree only intersects other subtrees at one of its nodes (the node one closest to the root).

Once we have this initial decomposition, with total degrees balanced within a constant, we continue the $\mathcal{P}A(m)$ process for the remaining $n - t$ steps. Notice that, for each new node i and edge $[i, j]$ generated out of it, the subgraph to which the other endpoint j belongs is generated by a distribution that is, initially, approximately uniform at random; that is, each has probability $\Theta(1/c)$. We show next that, with high probability, this approximately uniform distribution will be maintained throughout the process. This follows from the following lemma, which can be proved using the same techniques as the ones used to prove Lemma 7 in the proof of Theorem **??**:

Lemma 10 *Let $x_1, \ldots, x_c$ be the fractional loads of c bins containing at least $\omega(\log c)$ balls each, and let $x'_1, \ldots, x'_c$ be the fractional loads of the bins at any later time, after running the Polya urns process. Then with high probability $|x'_k - x_k| \leq \epsilon$ for all $k \in \{1, \ldots, c\}$, where $\epsilon > 0$ can be arbitrarily small, provided c is large enough.*

How do we assign new nodes to (up to $b = 2$) of the c subgraphs? For each new node i, suppose that its two edges are $[i, j]$ and $[i, j']$; we assign i to *both* subgraphs to which j and j' belong; this way we ensure that these two subgraphs have a node in common. In other words, each new node i can be seen as a draw of one of the c^2 possible pairs of subgraphs, each with probability that is $\Omega(1/c^2)$. By the coupon collector principle, after $\Theta(c^2 \log c) < n - t$ nodes, all components will intersect with high probability, completing the proof. ■

Another important model of Internet-like graphs [ACL00; MPS03] is the one in which we start with a degree sequence $d_1 \geq d_2 \geq \cdots \geq d_n$, such that $d_i = d_1 \cdot i^{-\alpha}$ for some α between 1/2 and 1, and then an edge is added between nodes i and j with probability proportional to $d_i \cdot d_j$. It is clear that, in such a graph, the degrees will be, in expectation, proportional to the d_i's. We call this the *degree sequence model*. By a similar argument and construction, which we omit, we can show the following:

Theorem 16 *Linked decompositions with exceptions and $a = \Theta(\sqrt{n \log n})$ can be obtained in the degree sequence model, as well as for $G_{n,p}$, with $p = \Theta(\frac{1}{n})$.*

Finally, we note that by another simple argument, random graphs in the $G_{n,p}$ model have linked decompositions (without exceptions), provided that p is above $\log n/n$.

3.7 The Internet and Routing

In recent years, we have seen a surge of research activity aiming at a theoretical and foundational understanding of the Internet. The motivation for such a research agenda is twofold: first, the Internet is a novel, fascinating, and intellectually challenging computational artifact of central importance to computing technology and society in general, and hence it is natural that it is an attractive subject for theoreticians. Second, even though the Internet has emerged without much deliberate design, such design may become necessary in the future; foundational understanding and theoretical insights would be handy at such a juncture. Indeed, as challenges of scale accumulate, there are several serious efforts underway to "redesign" the Internet [FI05; GE05; NA; CCK+06]. It is within this framework that we see the concept of linked decomposition, and

the theoretical and experimental evidence presented here that it is feasible in the context of the Internet.

It is not hard to see that a linked decomposition with parameters a, b, c, d enables a novel form of Internet routing with routing tables of size $\tilde{O}(ab + n/a)$, delay $O(d)$, and $O(d)$ packets sent per message routed. To show how this can be done, first note that each node v only needs $\tilde{O}(ab)$ space to store enough information to route to any node u which belongs to one of v's components. Furthermore, in order to route to any node u, which does not belong to one of v's components, v just needs an intermediate node w, which belongs both to one of u's components and one of v's components. (Note that such a node w must exist due to property (4) of our linked decomposition). Once we have found our node w, it is easy to route our message from v to u through w.

Conceptually, storing an intermediate node w for each destination node u appears to require a table of size $O(n)$, but fortunately node v does not have to store the entire table, as it is not hard to distribute this table among the nodes in v's components, such that each node only has to store $\tilde{O}(n/a)$ information. In particular, we can use hash functions to distribute the information in a manner which allows u to find the intermediate node w for any destination node v. Thus, we have a routing protocol that uses $\tilde{O}(ab+n/a)$ storage and routes with delay $O(d)$.

For a concrete example, see Figure 3.1 below. To route a packet from vertex 1 to vertex 2 in the given example, vertex 1 knows how to reach any node in component S_1 and can route the packet entirely within component S_1 by following the links in S_1. To route a packet from vertex 1 to vertex 3, which is not in any component of vertex 1, we a send lookup messages within component S_1 to determine the intermediate vertex to use, in this case vertex 4. The packet is then routed to vertex 4, and vertex 4 uses its routing table to complete the route to vertex 3.

Note that the above routing scheme can be implemented even if the addresses of the vertices are totally amorphous strings (called "flat labels" in the Internet redesign literature [CCK+06]), as opposed to today's hierarchical and geographically specific IP addresses — a key feature of today's Internet, which is also the source of some of the most challenging problems of scale and evolution.

By giving up "flat labels" and loosening our linked decomposition requirements, we can decrease table size to about $n^{1/3}$, as follows: Instead of requiring that all components intersect, we only require that any two components *either intersect with one another, or*

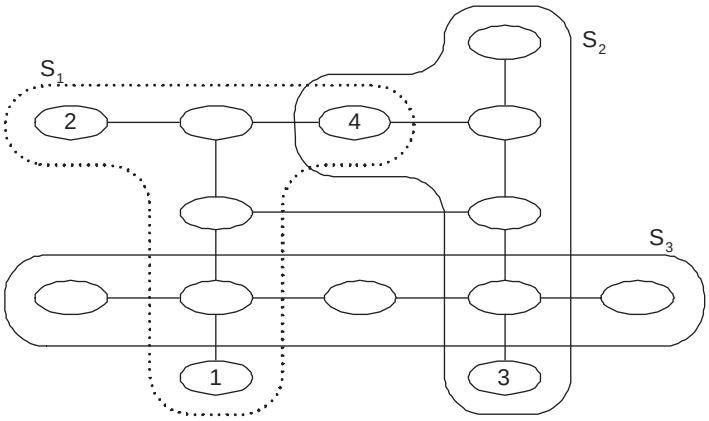

Figure 3.1: In this example, we show a decomposition with $a = 6$, $b = 2$, $c = 3$, and $d = 4$.

both intersect with the same component. That is, the intersection graph of the component is no longer a clique, but has diameter two. It turns out that, if the address of each node also contains a component indicating a subnetwork to which the node belongs, then routing tables of size about a suffice. And, by applying now both coupons collector and the birthday paradox (details omitted) it can be shown that $\mathcal{P}A(m)$ graphs have such a decomposition with $a = n^{1/3}$, whp.

The routing implications of our results explained in this section are related to two recent ideas. The one that provided direct inspiration is the *routing on flat labels (rofl)* proposal [CCK+06], a novel routing architecture borrowing methodologically from peer-to-peer networks. We came up with the concept of linked decompositions while trying to identify the limits of their approach. The rofl work is validated experimentally, and does not provide nontrivial performance guarantees. Another related body of work is that on *compact routing* [ACL+03; AGM+04; KFY04; Kkc] seeking routing algorithms with small routing tables and small *stretch* (worst-case ratio between routing delay and distance in the network). The strongest known such result achieves, for any graph, $\tilde{O}(\sqrt{n})$ tables and stretch 3 [AGM+04], and it is in fact known empirically [KFY04] that this algorithm runs better on the real Internet.

Although the existing compact routing work dominates our results, our approach is conceptually simpler, and does not require the use of landmark nodes. Avoiding the use of landmark nodes, may provide benefits in terms of congestion and robustness against failures, since as many as $\Omega(n^{3/2})$ pairs of nodes may route packets through a single landmark node

in the existing $\tilde{O}(\sqrt{n})$ routing scheme. As we describe in the open problems section, we hope to explore the potential benefits that linked decompositions may provide in terms of balanced congestion and robustness. In our experiments, the stretch of our routing algorithm is very rarely over 3. Furthermore, our work provides a rigorous explanation for the surprising empirical finding that in our experiments, shown in the next section, the actual Internet can be decomposed in such a demanding and unlikely way, which may prove useful in maintaining various control and management functions locally within each decomposition component.

3.8 Experiments

Preferential Attachment Graphs

In order to validate the linked decomposition idea, we apply the following simple algorithm to graphs generated according to $\mathcal{P}A(4)$: *Choose a number c to be the number of components (colors), and assign the first c nodes of the $\mathcal{P}A(4)$ process to their own component (color). For each new vertex of the $\mathcal{P}A(4)$ process after that, if it connects two components that are not yet connected, add it to both of them, otherwise add it to the smallest one. Continue generating nodes and assigning them to components until all components are connected.* Figure 3.2 below shows for each c, the number of nodes that need generated in order to connect all c components; as expected, it is quadratic.

After all the components become connected, we measure the smallest, largest, and average size of all components as a function of c, the number of components; we note occasional large deviations from the expected linear growth of the largest component.

Internet Graphs

We also applied our ideas to two actual Internet graphs, one showing connections between autonomous systems, known as the "AS graph" or "BGP graph", and one showing the connections between Internet routers, known as the "router graph". Both were obtained

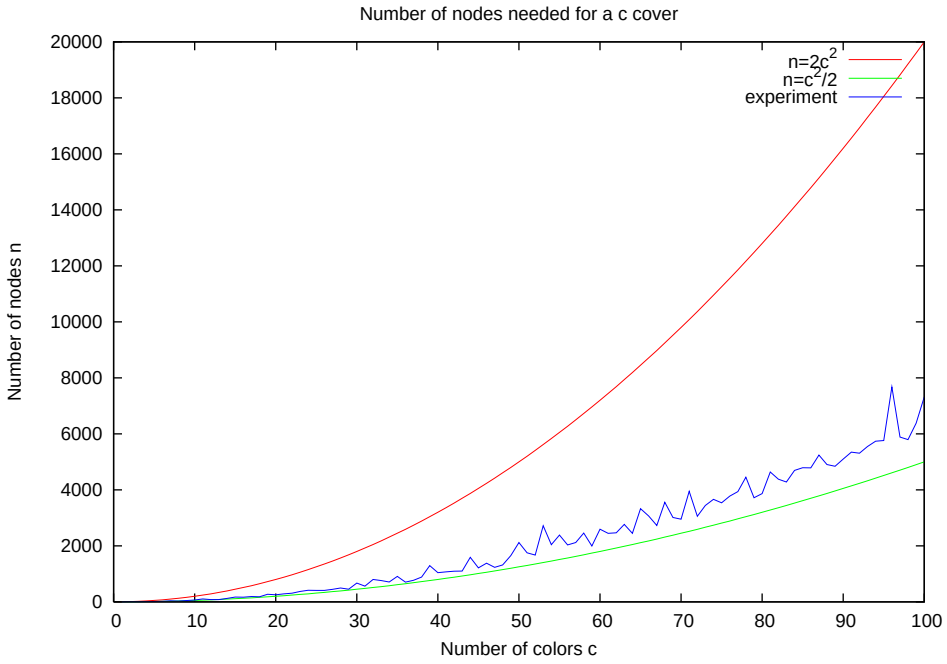

Figure 3.2: Number of nodes that need to be generated in order to connect c components (colors)

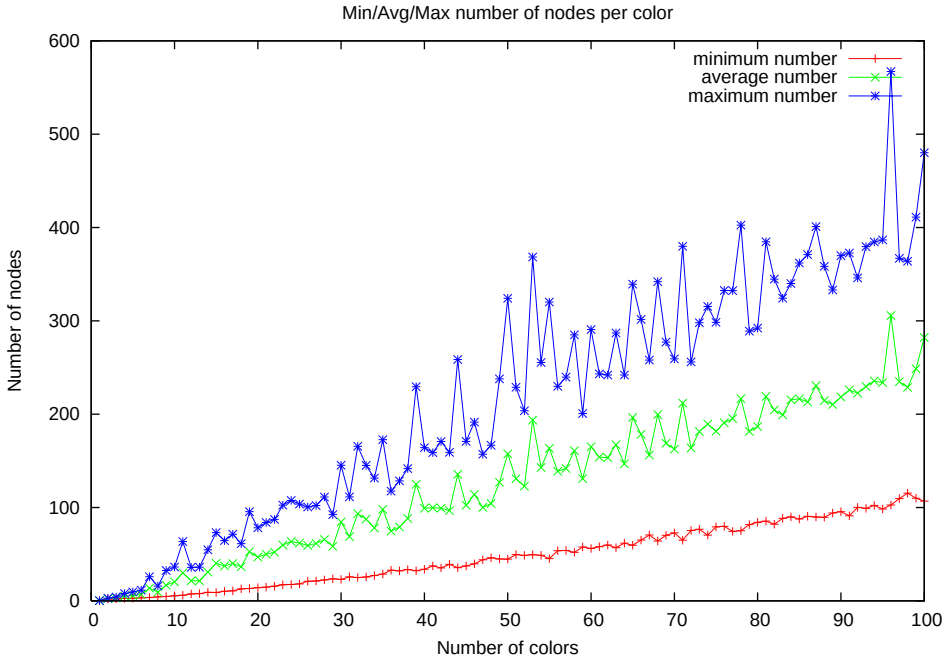

Figure 3.3: Final size of components after all c components become connected

from the CAIDA website [cai].

To run our algorithms on these graphs, we first preprocess them by repeatedly removing degree-one nodes. We approximate the (unknown) presumed arrival order of $\mathcal{PA}(m)$ by the order of decreasing degrees. One problem arises: Sometimes the next node is not connected to previous nodes. For this reason we maintain a priority queue containing such nodes, and remove them when a neighbor has been processed.

The BGP graph has, after the removal of leaves, 12358 nodes. With $b = 4$ it can be decomposed into 56 linked components, and with $b = 2$ to 35. Similarly, the router graph has, after the removal of leaves, 141,509 nodes. With $b = 4$ it can be decomposed into 215 components, but with $b = 2$ to only 93.

Interestingly, if we apply the same algorithm (sort by degree and then process, using a priority queue) to $\mathcal{PA}(m)$ data, after the end of the generation process, we get slightly worse results for small graphs, but slightly better results for larger graphs.

Routing

We simulated routing on the decompositions of real graphs that we obtained, by selecting 10,000 random source-destination pairs and measuring the stretch and congestion (number of times each node was used). The stretch results for both autonomous systems (BGP) graph and router graphs below show that the stretch very rarely exceeds 3.

But these experiments also showed that our scheme has a problem with congestion: In both cases, as roughly 10% of the traffic was directed *through one particular node!* Remedies are discussed briefly in this chapter's last section.

3.9 Open Problems

Our proofs are in some sense existential and non-constructive: Even though we give a decomposition algorithm, this algorithm needs to "see" the actual running of the $\mathcal{PA}(m)$ process in order to work with high probability; what if we are given ex post a graph that has been generated by $\mathcal{PA}(m)$, but with its nodes permuted? Or if we are actually given the graph of the Internet? In our experiments with permuted $\mathcal{PA}(m)$ graphs and Internet graphs we run our decomposition algorithms on the graph *with nodes ordered in*

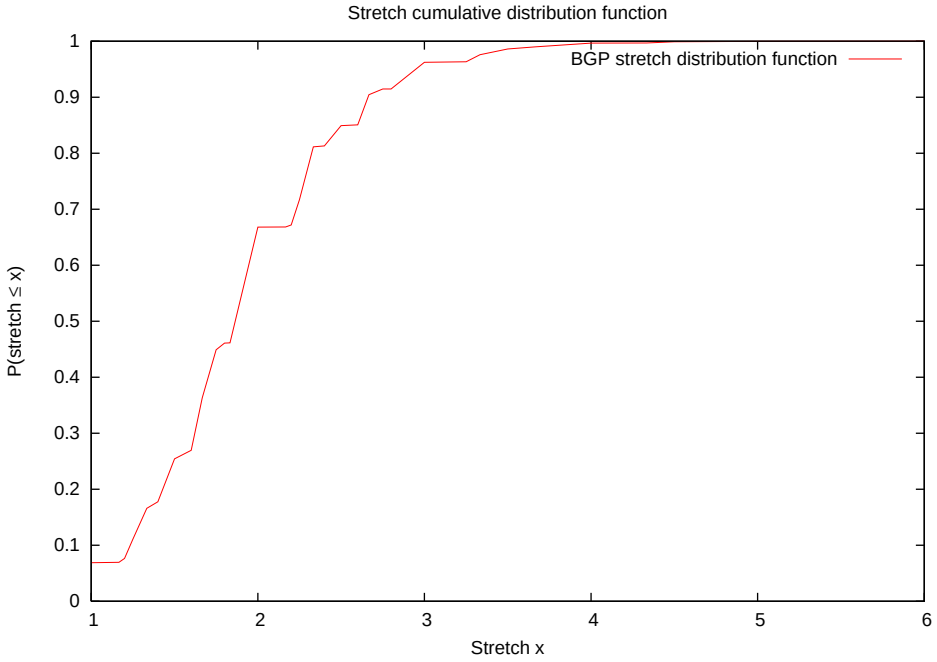

Figure 3.4: Stretch of linked decomposition scheme on the BGP graph.

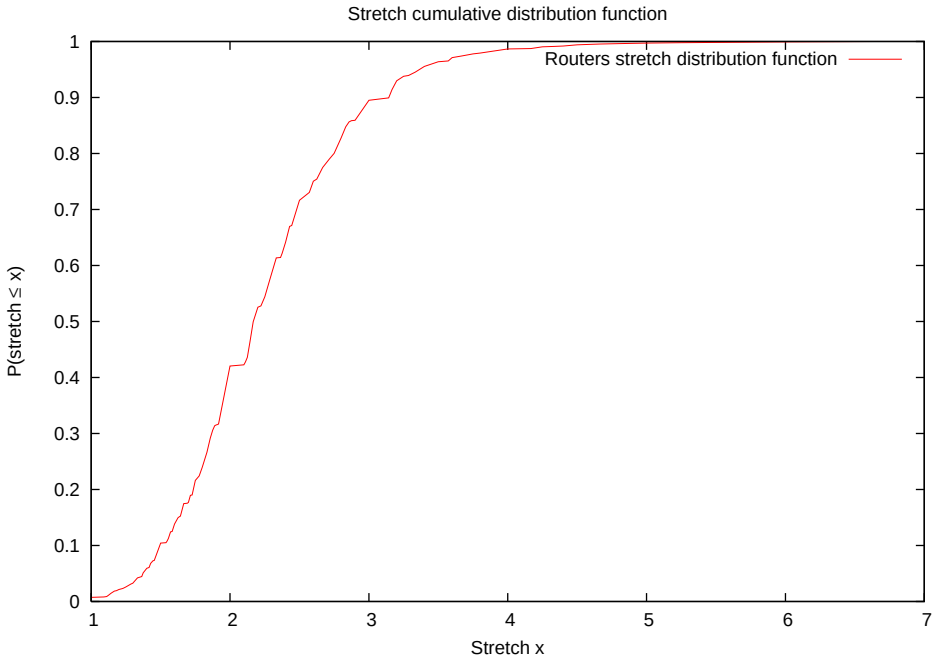

Figure 3.5: Stretch of linked decomposition scheme on Internet router graph.

decreasing degree. The intuition is that this is our best guess for the creation order. This works well in practice as shown in our experiments, and we would love to prove that it works with high probability on $\mathcal{P}A(m)$ graphs. In addition, some other open questions we would like to answer are:

- Does the Polya urn process with the power of *two* choices also balance after $O(n^{2+\epsilon})$ steps? In the case of interest, when each bin starts with fractional load at least $\frac{1}{2n}$, we can prove the power of two choices balances loads after $n^{3+\epsilon}$ steps (whp), but is an exponent arbitrarily close to two possible?
- Does our routing algorithm yield better worst-case congestion than the existing $\tilde{O}(\sqrt{n})$ routing scheme? We have removed the use of landmark nodes, but is it enough to balance congestion?
- Are there efficient ways to update the routing tables with the addition of new nodes and edges, or more importantly, node and edge deletions?
- Can autonomous systems in a network be appropriately incentivized to organize themselves in linked decompositions?

Chapter 4

An Online Bipartite Matching Problem

4.1 Background

In this chapter, we study an online bipartite matching problem, which models a scenario in which clients arrive over time and request permanent service from a set of given servers. As each client arrives, she announces a set of feasible servers capable of servicing her request, and our goal is to provide service to each client persistently by maintaining a matching at all times between clients who have arrived and servers capable of servicing their requests. We would like to assign clients to servers permanently without ever having to reassign clients to different servers, but when a new client arrives we may be forced to reassign existing clients to alternative servers to ensure that all clients can receive service. As it is often more important to provide service to all clients, the goal of our algorithm will be to maintain a matching always between arrived clients and allowed servers, while minimizing the *switching cost*, the total number of times that clients are reassigned to different servers.

Our online bipartite matching problem has a wide variety of applications spanning diverse areas, including streaming content delivery, web hosting, remote data storage, job scheduling, and hashing. We describe a few applications below, which can be modeled as an instance of the online bipartite matching problem we described above. In the following examples, we always refer to the entities requesting service as *clients* and the entities

providing service as *servers* for consistency. In examples where it is not clear, we mark in parenthesis which entities are clients and which entities are servers.

- **Streaming Content Delivery, Web Hosting, and Remote Data Storage** We have a set of servers capable of streaming content online, hosting web pages, or storing data remotely. A sequence of clients arrives requesting to have their content streamed online, their web pages hosted online, or their data stored online. Due to locality, security, cost, routing policy, or other reasons, the streaming content, web page, or data from each client can only be hosted at a subset of server locations.

- **Job Scheduling** We have a set of servers with differing capabilities available to process job requests from persistent sources - jobs that need to be processed over a long or indefinite period of time (e.g. protein folding, genomic research, SETI@HOME). A sequence of persistent job requests (clients) arrive and reveal a subset of servers capable of servicing their request.

- **Hashing** We have locations in a hash table (servers) available to store data objects (clients), and a set of hash functions. Data objects arrive over time and can be assigned to a location in the hash table, if one of the hash functions maps the data object to that location.

Note that in all the examples above, it is reasonable to assume that clients can be reassigned to different servers, but at a cost. For instance, in the streaming content example, clients may not be able to access their content while their content is being transferred from one server to another. Thus, it is desirable to minimize the number of the reassignments our algorithm incurs, which causes these interruptions.

Before stating our results, we define our problem more formally. Each instance of the online bipartite matching problem is defined by:

- A bipartite graph G between clients and servers, which represents the servers to which each client can connect

- A permutation σ, which represents the sequence in which the clients arrive

The graph G is unknown at the beginning, but as clients arrive according to σ, each client reveals the servers to which she has edges. The goal of the algorithm to is

to maintain a matching between arrived clients and servers at all times, while minimizing the total number of times that clients are reassigned to different servers. Alternatively, as each client arrives, our algorithm can be viewed as finding an augmenting path (a path that alternates between matched and unmatched edges), in the graph revealed so far, from the arriving client to an unmatched server. It should be clear that upper bounding the total length of the augmenting paths used by the algorithm, also upper bounds the total switching cost. As the total length of the augmenting paths is close to the switching cost, it can also be used to lower bound the switching cost in some cases as well.

For simplicity, note that we assume each server may service (or be matched to) at most one client at a time, although our results can be fully generalized to the case where servers can service multiple requests. We also assume that G consists of n clients and n servers, and G contains a perfect matching, although our results still hold essentially without these assumptions. See the next section on our model assumptions for further discussion.

Although this problem was introduced over a decade ago [GKKV95], we still know surprisingly little about the optimal algorithm. For example, there is a very natural *greedy algorithm*, which provides service to each new client by using an augmenting path of minimal length, but is this greedy algorithm optimal? What is its worst-case switching cost? There are no known upper bounds to show that the greedy algorithm or any other algorithm performs better than $O(n^2)$ in the worst case, but an upper bound of $O(n^2)$ is trivial since any reasonable algorithm only switches at most $O(n)$ clients per arriving client. Furthermore, an existing lower bound in [GKKV95] only shows that the total switching cost has to be $\Omega(n \log n)$, so a large gap exists between the known upper and lower bounds.

4.1.1 Online Matching Assumptions

Although we assume the total number of arriving clients is the same as the total number of servers, it is not difficult to see our upper bounds on the switching cost still hold if n clients arrive and there are $m > n$ servers. We omit formal details, but the bounds on the switching cost are the same for this case when measuring the switching costs relative to n, the number of clients.

We also assume that each server may service (i.e. be matched to) at most one client, but one should note that our results also hold when servers can service multiple clients. In this more general setting, when n clients are to be matched to m servers who

can service $s_1, s_2, \ldots, s_m$ clients each, our bounds also hold and are exactly the same as the unit capacity case, when measuring the cost relative to n the number of clients. The bounds from the unit capacity case also apply here, since we can reduce this more general problem to the unit capacity case by creating s_i server nodes for each server $i \in \{1, \ldots, m\}$. By treating each server i as s_i unit capacity servers, and creating edges appropriately, it is not hard to see that any bounds for the unit capacity case also yield equivalent bounds for the general capacity case in terms of n the number of clients that arrive.

Lastly, we assume that there is a matching at each step of the algorithm. We make this assumption because if a new client i arrives and no matching exists between the arrived clients and the servers, then we might as well ignore client i. We feel no remorse ignoring client i, since no algorithm could have matched i and the other arrived clients. By ignoring the clients who we cannot possibly serve (without disconnecting other clients), we are then left with an instance where at each step a matching exists between the arrived clients and the existing servers.

4.1.2 Our Results

Although worst case analysis for this problem has not yielded fruitful results, the worst case may not often occur in practice, and it may be reasonable to make certain assumptions about the graph G or the arrival order σ. For example, in the case of remote data storage, one might imagine that the graph G which determines the servers which can store a client's data is fixed, but perhaps the arrival order σ is random. Does the greedy algorithm perform provably better than $O(n^2)$ in this case? Moreover in some cases, it might be reasonable to assume that the graph G is generated randomly. For example, in the case of hashing, if $\Theta(\log n)$ random hash functions are chosen, then the set of edges between clients (data elements) and servers (hash locations) is a graph where each client has $\Theta(\log n)$ random edges to servers. Can better bounds be proved in this case as well?

In this chapter of the dissertation, we show that indeed the switching cost can be much better under these conditions, despite the lack of worst case upper bounds better than $O(n^2)$. In the first case, we show that when G is any arbitrary bipartite graph with a perfect matching, and σ is chosen uniformly at random, then the greedy algorithm performs well and achieves $O(n \log n)$ switching cost with high probability. This bound is tight as we show that there is a distribution over graphs for which any deterministic algorithm must

incur cost $\Omega(n \log n)$ in expectation, even if σ is chosen uniformly at random.

In the second case, where each client has $\Theta(\log n)$ random edges to servers, we show that the switching cost can be even better. In this case, we prove that the switching cost is $O(n)$ with high probability, and it nearly matches our lower bound of $\Omega(n/\log n)$. In terms of the number of edges used to generate the graph G, our result is tight, since $\Omega(\log n)$ random edges are needed per node, in order to guarantee a perfect matching with high probability.

Lastly, we also make the first progress in over a decade in the original worst case model where σ is chosen adversarially, but G is known to be acyclic (i.e. a forest). In this setting, we derive a new algorithm which achieves cost $O(n \log n)$, which matches the existing lower bound of $\Omega(n \log n)$ for forests. Although the networks that occur in practice are not often acyclic, we view our last result as making progress towards finding an $O(n \log n)$ solution in the general worst case setting.

4.1.3 Related Work

The problem studied here was first introduced by Grove et al. [GKKV95] in a paper, which focused on a special case of the problem where each client has degree at most two. For this special class of instances, they prove that the greedy algorithm incurs a worst-case switching cost of $O(n \log n)$, and they give a matching lower bound by showing there are cases where any algorithm must have cost $\Omega(n \log n)$. The paper goes on to consider the case in which clients can connect *and disconnect* over time; for this problem, assuming that each client has degree at most two, they present a randomized algorithm which has a competitive ratio of $O(\sqrt{n})$, where the *competitive ratio* of an online algorithm is the ratio between the cost incurred by the online algorithm, which does not know the input sequence in advance, and the cost incurred by an optimal algorithm which does know the input sequence in advance. Previous to our work, the special case in which each client has degree at most two, was the only class of instances for which an optimal online algorithm was known.

Our problem is related to previous work on online load balancing with task preemption, which has been studied by many authors, see [AGZ96; PW93; Wes95] for example. The main difference between our work and the previous work on load balancing is that our model assumes a hard capacity constraint on the servers and allows clients to be reassigned,

while the previous load balancing work generally does not assume a hard capacity constraint on the servers, or does not allow reassignments. Without hard capacity constraints or reassignments, the goal in online load balancing is often to minimize the maximum load, or if reassignments are allowed, the goal is to minimize the number of reassignments while always maintaining a maximum load which is close to the optimal maximum load achievable.

For example, a series of works by Azar et al. [ABK93; AKP$^+$93; ANR92] studied online load balancing *without* the possibility of task preemption (i.e. without reassigning clients, in our terminology). They established that when tasks arrive but never depart, the greedy algorithm, which assigns each task to the least-loaded admissible server, has $O(\log n)$ competitive ratio with respect to the maximum load. When tasks both arrive and depart, they show that the optimal competitive ratio is $O(\sqrt{n})$. When jobs arrive and depart and task preemption is allowed, the situation improves dramatically: Phillips and Westbrook [PW93] give an algorithm which always maintains a maximum load within a factor of $O(\log n)$ of optimal, while only incurring a reassignment cost of $O(m)$, where m is the number of arrivals and departures. Westbrook [Wes95] also gives an algorithm which is $O(1)$-competitive with respect to the maximum load and with a reassignment cost of $O(m \log n)$. Andrews et al. [AGZ96] study online load balancing with reassignment costs in a model in which any client may be assigned to any server, but clients have arbitrary sizes and reassignment costs. (The same model was considered, in lesser generality, in [Wes95].) They give an online algorithm which is 3.5981-competitive with respect to load and 6.8285-competitive with respect to reassignment cost.

Another line of related work involves a variant of our online bipartite matching problem, where reassignments are not allowed and capacity constraints must be maintained exactly. Given these restrictions, the goal of the online algorithm is to match as many clients as possible. This problem was first studied by Karp et al. [KVV90], and later generalized by Goel and Mehta [GM08], and Mehta et al. [MSVV07] for the purposes of studying an online adword placement problem.

Our work is also loosely related to the recent work of Godfrey, who analyzes the load balancing properties of certain random processes which assign clients to servers [God08]. Godfrey proves that only very weak conditions are needed on the random process, in order to ensure that the servers stay roughly balanced with high probability.

Our load-balancing scenario in Section 4.3 also has connections to hashing. A large body of theoretical and experimental research has focused on hashing schemes and

dictionary data structures; a dictionary data-structure is a table containing items, and the goal is to design a data-structure which supports fast insertions and accesses, with a fairly high space-utilization. Typically, dictionaries are used in combination with a hashing scheme, which is a mapping from the items to their locations in the table. A common assumption in theoretical work is that such mappings are random. Under this assumption, the connection between hashing and our setting is as follows: if an item has d randomly chosen locations where it can be inserted into the table, then the problem of inserting a series of items into the table reduces to the problem of finding a matching in a random bipartite graph in an online manner.

Under this setting, our algorithm is very similar to the Cuckoo Hashing scheme of Pagh and Rodler [PR01]. Cuckoo hashing, essentially, uses the following algorithm for inserting items to tables, or equivalently, matching new clients to servers. If there is an empty server adjacent to the client to be matched, then the client is matched to this server; otherwise, it is matched arbitrarily to one of its adjacent servers, and a new matching is recursively found for the client which was previously attached to this server. In [PR01; FPSS03], it was shown that if the degree of each node is $d = O(\ln \frac{1}{\epsilon})$, and if there are n clients and $n(1+\epsilon)$ servers, then the expected amortized switching cost per client is $O_\epsilon(1)$, where the $O_\epsilon(1)$ is a constant dependent on ϵ and the expectation is with respect to the randomness in the graph between clients and servers. In contrast, our results imply that if $d = O(\log n)$, and if there are n clients and n servers, the amortized switching cost per client is still $O(1)$ with high probability. By making ϵ small, say $\epsilon = O(\frac{1}{n})$, the result in [FPSS03] can also be used to prove bounds on the switching cost when n clients are matched to n servers, but the bounds produced are very weak, since the bound on the switching cost per client depends on ϵ and is super-polynomial in $(\frac{1}{\epsilon})$. Thus, our results produce a novel extension to results of [PR01; FPSS03] to the case where there is no surplus of servers over clients.

4.2 Random Arrival Order

In this section, we assume that the bipartite graph G between clients and servers is arbitrary, but that the clients arrive in a uniformly random order (i.e. σ is uniformly distributed over all $n!$ possible arrival orders). Under this assumption, we prove that the natural greedy algorithm for the problem, which connects each client by using the shortest

augmenting path available, suffers a switching cost of $O(n \log n)$ with high probability. We also prove a matching lower bound of $\Omega(n \log n)$ on the total switching cost for any online algorithm, which shows that $O(n \log n)$ is the best switching cost that can be achieved in the random-ordering model.

4.2.1 The Upper Bound

To obtain some intuition, we start by showing that the total switching cost is at most $O(n \log n)$ in expectation. Our analysis here is just for intuition, as a more sophisticated argument is needed to show that the total cost is $O(n \log n)$ with high probability. To prove that the total switching cost is $O(n \log n)$ in expectation, we start by proving the following lemma, which upper bounds the expected cost of each arriving client.

Lemma 11 *Given that i clients remain to arrive, the expected number of servers that the greedy algorithm needs to switch in order to connect the next client is at most $\frac{n}{i}$.*

Given the lemma, it is easy to sum over all clients and bound the total expected switching cost by $\sum_{i=1}^{n} \frac{n}{i} = O(n \log n)$. We now prove the lemma.

To prove the lemma, let us assume for notational purposes that clients $\{1, \ldots, i\}$ have yet to arrive, and let us define d_k to be the number of servers that need to be switched in a shortest augmenting path from client k to a free server, if client k arrives next, for $k \in \{1, \ldots, i\}$. Note that when clients $\{1, \ldots, i\}$ have yet to arrive, the expected cost of the next arriving client is $\sum_{k=1}^{i}(d_k \cdot \frac{1}{i}) = \frac{1}{i}\sum_{k=1}^{i} d_k$, since each client k has probability $\frac{1}{i}$ of arriving next and experiences switching cost d_k, if it arrives next. Now, if we can show that $\sum_{k=1}^{i} d_k \leq n$, then the lemma follows easily, since the expected switching cost of the next arriving client can then be upper bounded by $\frac{1}{i}\sum_{k=1}^{i} d_k \leq \frac{1}{i} \cdot n = \frac{n}{i}$.

To show that $\sum_{k=1}^{i} d_k \leq n$, note that if all i remaining clients were to arrive next all at once, then it would be easy to connect the remaining i clients with switching cost n: simply compute a perfect matching, and then switch all clients to match the same servers as designated in the perfect matching. This connects all remaining clients, and causes each client to switch servers at most once, and thus has total switching cost n. Furthermore, this implies the existence of i augmenting paths, which can be used to connect remaining clients $\{1, \ldots, i\}$, with total switching cost n. Moreover, if we look at $\sum_{k=1}^{i} d_k$, it must be the case that $\sum_{k=1}^{i} d_k \leq n$, since we have established that there exists a set of i augmenting

paths with total switching cost n, and d_k is the *fewest* number of servers that need to be switched if a client $k \in \{1, \ldots, i\}$ arrives next. Therefore, the expected switching cost of the next arriving client is at most $\frac{1}{i}\sum_{k=1}^{i} d_k \leq \frac{n}{i}$, and the lemma follows. ■

We now prove the main theorem of this section, which states the total switching cost is bounded by $O(n \log n)$ with high probability.

Theorem 17 *For any bipartite graph G between clients and servers, if the clients in G arrive in uniformly random order, then the switching cost incurred by the greedy algorithm is at most $O(n \log n)$ with probability at least $1 - n^{-2}$.*

To prove the cost is $O(n \log n)$ with high probability, we couple our online matching process with a sequence of n^2 binary random variables, $X_{i,j}$ for $i, j \in \{1, \ldots, n\}$, such that $\sum_{i,j\in\{1,\ldots,n\}} X_{i,j}$ stochastically dominates the total augmentation cost. We then prove a large deviation bound on $\sum_{i,j\in\{1,\ldots,n\}} X_{i,j}$. Ideally, it would be nice if we could simply define $X_{i,j}$ to be a random variable which is 1 if server j is switched by the greedy algorithm, when i clients have yet to arrive and a new client arrives, and is 0 otherwise. Although it is easy to see that the $\sum_{i,j\in\{1,\ldots,n\}} X_{i,j}$ is the total switching cost, it is difficult to prove a large deviation bound on $\sum_{i,j\in\{1,\ldots,n\}} X_{i,j}$, since the $X_{i,j}$ variables are not independent.

Thus, in order to prove a large deviation bound, we will define our $X_{i,j}$ variables in another way, such that $\sum_{i,j\in\{1,\ldots,n\}} X_{i,j}$ stochastically dominates the total switching cost, and each $X_{i,j}$ variable is independent of $X_{i',j}$, for all $i' \neq i$, and for each fixed j. We can then take advantage of this independence, and use a Chernoff bound to prove that for each fixed j, the $\sum_{i=1}^{n} X_{i,j}$ is $O(\log n)$ with high probability. From there, it follows almost immediately that $\sum_{i,j\in\{1,\ldots,n\}} X_{i,j}$ is $O(n \log n)$ with high probability.

Now to define our $X_{i,j}$ variables, each $X_{i,j}$ variable will be 0 or 1, depending on the servers that get switched when there are i clients yet to arrive and a new client arrives, and our coupling will be defined so that for each $i \in \{1, \ldots, n\}$, $\sum_{j=1}^{n} X_{i,j}$ stochastically dominates the number servers that need to be switched in order to connect a new arriving client, when i clients remain to arrive. Each $X_{i,j}$ variable will be 1 with probability $(\frac{1}{i})$, and will be independent of any other variable $X_{i',j}$, for $i' \neq i$ and fixed j.

To define $X_{i,j}$ and our coupling formally, let us assume that i clients $\{1, \ldots, i\}$ have yet to arrive, and let us define d_k to represent the minimum number of servers that need to be switched in order to connect client k, if it were to arrive next, for $k \in \{1, \ldots, i\}$. Recall that $\sum_{k=1}^{i} d_k \leq n$ from the argument in Lemma 11, and thus, we can assign to each client $k \in \{1, \ldots, i\}$, d_k distinct binary random variables from $\{X_{i,j} \mid j \in \{1, \ldots, n\}\}$ without overlap. These variables can be easily coupled to our random process to stochastically dominate the switching cost, if the kth client arrives next; if the kth client arrives next, then we set the d_k binary variables assigned to it to have value 1, and 0 otherwise. Note that this causes each binary variable assigned to a client in $\{X_{i,j} \mid j \in \{1, \ldots, n\}\}$ to be true with probability $(\frac{1}{i})$, since each client arrives with probability exactly $(\frac{1}{i})$. For consistency, we also set variables in $\{X_{i,j} \mid j \in \{1, \ldots, n\}\}$, which are not assigned to any client, to have value 1 independently at random with probability $(\frac{1}{i})$ and 0 otherwise. As we have defined our coupling, note that $\sum_{j=1}^{n} X_{i,j}$ always dominates the number servers that need to be switched in order to connect a new client, when i clients remain to arrive. Furthermore, $\Pr[X_{i,j} = 1] = (\frac{1}{i})$ for any $i, j \in \{1, \ldots, n\}$.

Note that even though we assign the variables to cover each client's switching cost in a way that is dependent on events in the past, each $X_{i,j}$ variable is 0 or 1 with probability $(\frac{1}{i})$, independent of all the previous $X_{i',j}$ random variables with $i' < i$, because regardless of which assignment is made, each $X_{i,j}$ variable is 0 or 1, based only on which client is selected to arrive at the current time, an event which is independent of the previous $X_{i',j}$ random variables. Thus, our random process defines the random variables $X_{i,j}$ in a way such that each $X_{i,j}$ variable is independent of random variables $X_{i',j}$, where $i' \neq i$ and j is fixed. Furthermore, it is not hard to see that $\mathbb{E}[\sum_{i=1}^{n} X_{i,j}] = \sum_{i=1}^{n} \frac{1}{i} = O(\log n)$, for any $j \in \{1, \ldots, n\}$, and it is easy to apply a standard Chernoff bound to conclude that $\sum_{i=1}^{n} X_{i,j} = O(\log n)$ with probability at least $(1 - n^{-3})$. A simple union bound then implies that $\sum_{i,j \in \{1,\ldots,n\}} X_{i,j} = \Theta(n \log n)$ with probability at least $(1 - n^{-2})$. Therefore, the total switching cost must be at most $O(n \log n)$ with probability at least $(1 - n^{-2})$, and the theorem holds. ■

4.2.2 The Lower Bound

We now prove a lower bound on the switching cost in our random arrival order model, which shows our upper bound is optimal.

Theorem 18 *For any online algorithm, there exists a graph G such that the algorithm incurs an expected switching cost of $\Omega(n \log n)$ when the clients of G arrive in random order.*

Using Yao's lemma, it suffices to specify a distribution on input instances such that for any deterministic online algorithm, the expected switching cost incurred by the algorithm is $\Omega(n \log n)$, where the expectation is taken over the random choice of instances and the random ordering of clients. We define our instance distribution as follows: form a random bipartite graph between n clients and n servers by choosing a random permutation π of $\{1, 2, \ldots, n\}$ and matching client i to servers $\pi(i)$ and $\pi(i+1)$, where $\pi(n+1)$ is interpreted to mean $\pi(1)$. In other words, the bipartite graph is a random $2n$-cycle on the set of clients and servers.

This input distribution is invariant under permutations of the clients, so we may assume without loss of generality that the clients' arrival order is $1, 2, \ldots, n$. At the arrival time of client k, the servers belong to $n - k + 1$ different connected components (including isolated servers) and each of these connected components is a path. Client k is adjacent to endpoints of two of these paths, namely, the arcs of the cycle which extend from client k to the first higher-numbered client encountered when going around the cycle in either direction. The theorem now follows from a sequence of observations outlined below.

1. Conditioning on the cyclic ordering of all clients besides k, client k is equally likely to be spliced anywhere into the cycle. In particular, it has probability $1/3$ of being spliced into the middle third of an arc between higher-numbered clients.

2. Consider the cyclic ordering of all clients besides k. The clients with label greater than k partition the ordering into $n - k$ arcs. It is easy to see the average length of

these arcs is $(n-1)/(n-k)$, and hence, the expected distance from client k to the nearest higher-numbered client is at least $\frac{n-1}{3(n-k)}$.

3. The input distribution is invariant under the operation of cutting out an arc of the cycle, with its servers at both endpoints, and reattaching it backwards. Therefore, conditional on the states of the two paths to which client $k < n$ is connected at the time of its arrival (i.e., the sets of clients and servers in those paths and the current matching between them), each of the ways for k to connect to these two paths is equally likely. As client k connects to these two paths by choosing an endpoint of each, the number of ways for k to connect to the two paths is the product of the number of endpoints of the paths. As each path either is a single node with 1 endpoint or has 2 endpoints, the number of ways of connecting client k to these two paths is at most 4. Thus, conditional on the states of the two paths at the time of client k's arrival, with probability at least $1/4$, client k connects to each of these paths at the endpoint which is furthest from the free server on that path.

4. Combining (2) and (3), we find that the expected cost incurred at the arrival time of client k is at least $\frac{n-1}{12(n-k)}$. Summing over k, we get the stated lower bound.

■

4.3 Random Connection Model

In this section we study the following scenario: there is an equal number n of servers and clients, the clients arrive sequentially, and each of them selects, at arrival, a random subset of $O(\log n)$ servers. We provide an online matching protocol for this setting which incurs total switching cost of $O(n)$ with high probability, so that the average switching cost is $O(1)$ per client. Moreover, we give a lower bound, showing that any online matching protocol requires switching cost $\Omega(n/\log n)$ with high probability, which establishes that our upper bound is tight to within a factor of $O(\log n)$. Before proceeding to the details of our results, we note that, since the number of servers and clients are equal, we need to assume that every client selects $\Omega(\log n)$ servers in order to guarantee that a matching exists, after all clients arrive.

4.3.1 The Upper Bound

The idea of our protocol is this: When a new client arrives, the current server-client graph is examined, and a binary tree originating at the new client is grown carefully, in order for an augmenting path to be found along the edges of that tree. Our analysis establishes that the binary tree that is explored contains a free server at a constant depth on average, and as a result, the augmenting path, and hence the resulting switching cost, is constant on average as well.

Theorem 19 *Consider the following online matching problem: There are n servers, and n clients arrive sequentially, each client selecting $O(\log n)$ servers at random. There exists a client-server assignment protocol for this online arrival model which, with high probability, succeeds in matching each client with a server with overall switching cost of $O(n)$.*

Let $\mathcal{S} = \{s_1, \ldots, s_n\}$ be the set of servers and $\mathcal{C} = \{c_1, \ldots, c_n\}$ the set of clients, and let us suppose that the clients arrive in the order $c_1, c_2, \ldots, c_n$, where t, $t = 1, \ldots, n$, is the arrival time of client c_t. Without loss of generality, we assume that $n = 2^\ell$, for some integer ℓ; we also assume that every client selects a random set of $\alpha \cdot \log_2 n$ servers from the set $\mathcal{S}$ with repetition, for some sufficiently large constant $\alpha > 0$; with minor modifications in the proof, the result extends to the case where n is not a power of 2 and the clients select servers without repetition. For this model, we show that the Online Matching Protocol described in Figure 4.1 satisfies the following properties with high probability. (Here and throughout this section, we interpret "with high probability" to mean "with probability at least $1 - O(1/n)$.")

- at every time step $t > 0$, the clients $c_1, \ldots, c_t$ are matched with a subset of t servers;
- the total switching cost incurred by the protocol is $O(n)$.

In the description of the protocol in Figure 4.1 we use the following notation. We denote by $f_t : \{c_1, \ldots, c_t\} \to \mathcal{S}$ the matching of clients to servers that the protocol maintains at time t, and by $g_t : \mathcal{S} \to \{\neg\} \cup \{c_1, \ldots, c_t\}$ the "inverse matching", defined so that $g_t(s) = c$ iff $c \in \{c_1, \ldots, c_t\}$ and $f_t(c) = s$. Finally, by $\mathcal{G}_t$ we denote the bipartite graph with node

set $\{c_1, \ldots, c_t\} \sqcup \mathcal{S}$ and an edge between client c_i, $i \leq t$, and server s iff $s \in \mathcal{S}_{c_i}$, where $\mathcal{S}_{c_i}$ is the set of servers that client c_i selects.

ONLINE MATCHING PROTOCOL

At time step $t > 0$:

1. client c_t chooses a random set $\mathcal{S}_{c_t}$ of $\alpha \cdot \log_2 n$ servers from the set $\mathcal{S}$ with repetition;
2. c_t orders $\mathcal{S}_{c_t}$ arbitrarily into a list $\mathcal{L}_{c_t}$ and sets $j_{c_t} = 1$ (to be used as an index in her list of servers);
3. $\mathcal{P}$:=BINARY_BFS(c_t, f_{t-1}, g_{t-1});
   /*upon success BINARY_BFS returns an augmenting path on the graph $\mathcal{G}_t$ originating at the node c_t*/

   if $\mathcal{P} \neq \emptyset$ **then** augment matching f_{t-1} along path $\mathcal{P}$; define f_t, g_t appropriately;
   else declare FAIL;

Figure 4.1: High-level description of the protocol; when client c_t joins the network an augmenting path from c_t to a free server is sought; if such a path is found, the current matching is augmented along this path to include client c_t.

BINARY_BFS

input: client c_t, current matching f_{t-1}, inverse matching g_{t-1};

output: augmenting path $\mathcal{P}$ on the graph $\mathcal{G}_t$ starting at node c_t, or $\emptyset$;

1. let $\sigma := \mathcal{L}_{c_t}(j_{c_t})$;
2. **if** $g_{t-1}(\sigma) = \neg$ **then** $\mathcal{P} := \langle (c_t, \sigma) \rangle$; **return** $\mathcal{P}$;
   else initialize a queue data structure $\mathcal{Q}$ containing the element $g_{t-1}(\sigma)$;
3. **while** $\mathcal{Q} \neq \emptyset$
   (a) $c := deQueue(Q)$;
   (b) **for** $r = 1, 2$
   if $j_c < \alpha \log_2 n$ **then**
   i. $j_c := j_c + 1$; $\sigma := \mathcal{L}_c(j_c)$;
   ii. **if** $g_{t-1}(\sigma) = \neg$ **then**
   let $\mathcal{P}$ be the path from node c_t to node σ on the tree created by the process;
   return $\mathcal{P}$;
   else push $g_{t-1}(\sigma)$ into $\mathcal{Q}$;
   else declare FAIL;

Figure 4.2: When a client c_t joins the network, the BINARY_BFS process explores the graph $\mathcal{G}_t$ in a Breadth-First-Search fashion in order to find an augmenting path from c_t to a free server. However, each time a client node is encountered by the BFS process, only two of its adjacent edges are explored. Note that our BINARY_BFS process may actually explore an edge e, which goes from some client to a server u, which has already been explored and thus creates a cycle. In this case, we do not add edge e to our tree as it creates a cycle, although we do continue our BINARY_BFS by exploring two more edges from the client with which server u is matched.

We will make use of the following fact about the Coupon Collector Model, which is not hard to prove.

Lemma 12 (Coupon Collector Model) *Suppose that there are n types of coupons. If every coupon has one of the n types uniformly at random, then, with probability at least $1-(2/e)^n$, after $k \cdot n$ coupons are requested at most $n/k-1$ types of coupons are uncollected.*

Let S be any set of $n - \frac{n}{k} = \left(1 - \frac{1}{k}\right) n$ coupon types. Then, if $k \cdot n$ coupons are requested,

$$\begin{aligned} &\Pr[\text{all } k \cdot n \text{ coupons are from the set of types } S] \\ &\qquad \leq \left(1 - \frac{1}{k}\right)^{kn} \leq e^{-n}. \end{aligned}$$

It follows that, if $k \cdot n$ coupons are requested, then

$$\begin{aligned} &\Pr\left[\text{at least } \frac{n}{k} \text{ coupons are uncollected}\right] \\ &\qquad \leq \sum_{S, |S| = \left(1-\frac{1}{k}\right)n} \Pr\begin{bmatrix}\text{all } k \cdot n \text{ coupons are from} \\ \text{the set of types } S\end{bmatrix} \\ &\qquad \leq \left(\frac{2}{e}\right)^n, \end{aligned}$$

where the summation ranges over all sets S of coupon types of size $|S| = \left(1 - \frac{1}{k}\right) n$. ■

To analyze the performance of the Online Matching Protocol we are going to temporarily forget the fact that every client has a list of $\alpha \cdot \log_2 n$ servers; we will pretend instead that every client has an infinite list of servers selected uniformly at random. Under this assumption, with probability 1, the protocol does not declare Fail and every client gets matched with a server. We establish the following lemma which concludes the proof of the theorem.

Lemma 13 *Under the assumption of infinite server-lists, the following are satisfied with high probability, i.e. with probability at least $1 - O(1/n)$:*

1. *the total augmentation cost incurred by the protocol is $O(n)$;*

2. *no client explores more than* $\alpha \cdot \log_2 n$ *servers from her infinite list of servers.*

To show Part 1 of the lemma, let us divide the arrival times of the clients into $\log_2 n+1 \equiv \ell+1$ progressively smaller intervals $\mathcal{I}_j = \{b_j, \ldots, e_j\}$, $j = 1, \ldots, \log_2 n + 1$, where each interval j has $\lceil \frac{n}{2^j} \rceil$ time steps. In particular, we define:

- $b_j = n - n/2^{j-1} + 1$, for all $j = 1, \ldots, \log_2 n + 1$;
- $e_j = n - \lfloor n/2^j \rfloor$, for all $j = 1, \ldots, \log_2 n + 1$;

Let T_i be the tree constructed by the BINARY_ BFS process when client c_i arrives and denote by $|V_i|$ the number of client nodes in T_i. We show the following lemma.

Lemma 14 *Under the infinite server-lists assumption, with high probability over the random choices of servers by the clients,*

$$\textit{for all } j \in \{1, \ldots, \log_2 n + 1\} : \sum_{i \in \mathcal{I}_j} |V_i| \leq 2^j n.$$

Let us fix any $j \in \{1, \ldots, \log_2 n + 1\}$. Let K_j be the number of times Steps 1 and 3(b)i of BINARY_BFS are invoked between the arrival of client c_{b_j} and until client c_{e_j} is matched with a server. Observe that, every time the Steps 1 and 3(b)i of BINARY_BFS are invoked, the identity of server σ is independent of the identities of the servers revealed in the preceding invocations of Steps 1 and 3(b)i of BINARY_BFS. Also, observe that, if the number of distinct servers that the invocations of Steps 1 and 3(b)i of BINARY_BFS have returned over the course of the protocol is at least $\lceil n - \frac{n}{2^j} \rceil$, then the clients of the set $\{c_i\}_{i \in \cup_{t \leq j} \mathcal{I}_t}$ have all been matched with servers. From Lemma 12 it follows that, with probability at least $1-(2/e)^n$, after $2^j n$ invocations of Steps 1 and 3(b)i of BINARY_BFS, $\lceil n - \frac{n}{2^j} \rceil$ distinct servers will be discovered. Hence, with probability at least $1 - (2/e)^n$,

$$K_j \leq 2^j n. \tag{4.1}$$

It is easy to see that the number of clients encountered in the trees in phase j is upper bounded by the number of new edges explored in phase j, so that the following holds:

$$\sum_{i \in \mathcal{I}_j} |V_i| \leq K_j \leq 2^j n.$$

The lemma then follows by a union bound. ■ From Lemma 14 it follows that, for all $j = 1, \ldots, \log_2 n + 1$,

$$\frac{1}{|\mathcal{I}_j|} \sum_{i \in \mathcal{I}_j} |V_i| \leq \frac{1}{|\mathcal{I}_j|} 2^j n \leq 2^{2j}$$

$$\Rightarrow \log_2 \left\{ \frac{1}{|\mathcal{I}_j|} \sum_{i \in \mathcal{I}_j} |V_i| \right\} \leq 2j. \tag{4.2}$$

By Jensen's inequality and the concavity of $\log_2$ it follows that

$$\log_2 \left\{ \frac{1}{|\mathcal{I}_j|} \sum_{i \in \mathcal{I}_j} |V_i| \right\} \geq \frac{1}{|\mathcal{I}_j|} \sum_{i \in \mathcal{I}_j} \log_2 |V_i|.$$

By combining the above with (4.2), it follows that

$$\sum_{i \in \mathcal{I}_j} \log_2 |V_i| \leq 2j \left\lceil \frac{n}{2^j} \right\rceil,$$

and summing over j it follows that

$$\sum_{i=1}^{n} \log_2 |V_i| \leq n \sum_{j=1}^{\log_2 n} \frac{2j}{2^j} + 2(\log_2 n + 1) = O(n).$$

Finally, observe that, when a client c_i joins the network, the augmenting path chosen by the protocol has length $O(1 + \log_2 |V_i|)$. It follows that the total augmentation cost paid by the protocol is $O(n)$.

To prove part 2 of the lemma, we make use of the following well known facts.

Lemma 15 (Coupon Collector Model) *In a coupon collector model with n coupon types, the probability that all coupons are not collected after $2n \log_e n$ steps is at most $\frac{1}{n}$.*

Lemma 16 (Balls in Bins) *In the balls and bins model, if m balls are thrown into n bins, then, with probability at least $1 - \frac{1}{n}$, the maximum load of a bin is $O(\frac{m}{n} + \log n)$.*

From Lemma 15, it follows that, with probability at least $1 - \frac{1}{n}$, the total number of invocations of Steps 1 and 3(b)i of BINARY_BFS throughout the course of the protocol is at most $2n \log_e n$. Since every client participates exactly once in an invocation of Step 1,

to conclude the proof of Part 2, it is enough to show that, with high probability, no client c participates more than $O(\log n)$ times in an invocation of Step 3(b)i. We argue this by coupling the process of selecting clients for invoking Step 3(b)i with the process of throwing $m = n \log n$ balls into n bins. For our purposes the bins are the clients and a ball, thrown at the time of an invocation of a Step 1 or a Step 3(b)i of BINARY_BFS, is received by a client (bin) c if the server selected by the invocation is matched with client c; as a result of this receipt, c will be the next client to participate in an invocation of Step 3(b)i, at which time another ball will be thrown, etc. Note that a ball might not be received by any client; in particular, for any ball that gets thrown the probability that a client c receives it is at most $\frac{1}{n}$. Hence, the maximum load that a client receives in our model is dominated by the maximum load of a bin in a balls in bins process whereby $2n \log_e n$ balls are thrown into n bins; the latter is $O(\log n)$ with probability at least $1 - \frac{1}{n}$ by Lemma 16. This concludes the proof of part 2 of the lemma. ■ ■

4.3.2 The Lower Bound

We now establish a lower bound of $\Omega(n/\log n)$ on the switching cost of any online protocol. Hence, the protocol described in the proof of Theorem 19 is optimal up to a factor of $O(\log n)$.

Theorem 20 *In the setting of Theorem 19, any online matching protocol has switching cost of* $\Omega(n/\log n)$*, with high probability.*

Let $\mathcal{S} = \{s_1, \ldots, s_n\}$ be the set of servers and $\mathcal{C} = \{c_1, \ldots, c_n\}$ the set of clients, and let us suppose that the clients arrive in the order $c_1, c_2, \ldots, c_n$, where t, $t = 1, \ldots, n$, is the arrival time of client c_t. Suppose also that every client selects a random set of $D := \alpha \cdot \log_2 n$ servers with repetition; as in the proof of Theorem 19, with minor modifications the argument extends to the case where the selection happens without repetition. Denoting by $\mathcal{S}_{c_t}$ the set of servers that client c_t chooses, let us define the following collection of events for $t = 1, \ldots, n$:

$$\mathcal{A}_t := \begin{array}{l}\text{“When client } c_t \text{ arrives all servers in} \\ \mathcal{S}_{c_t} \text{ are occupied by other clients.”}\end{array}$$

Recall that an online matching protocol needs to maintain a matching of the clients with servers at all times. Hence, if the event $\mathcal{A}_t$ happens, the protocol must incur an extra cost of at least 1 in period t in order to service client c_t. Moreover, observe that

$$\Pr[\mathcal{A}_t] = \left(\frac{t-1}{n}\right)^D.$$

Therefore, the expected switching cost incurred by the protocol is

$$\begin{aligned}\mathbb{E}[\text{switching cost}] &\geq \sum_{t=1}^{n}\left(\frac{t-1}{n}\right)^D = \frac{1}{n^D}\sum_{t=1}^{n-1} t^D \\ &\geq \frac{1}{n^D}\int_{t=0}^{n-2} t^D dt = \frac{1}{n^D}\frac{(n-2)^{D+1}}{D+1} \\ &= \frac{(n-2)^D}{n^D}\frac{n-2}{D+1} = \Omega\left(\frac{n}{\log n}\right).\end{aligned}$$

So, in expectation, every online matching protocol has cost $\Omega(n/\log n)$. To show that this is also true with high probability, note that the events $\{\mathcal{A}_t\}_{t=1}^n$ are independent. The result follows from an easy Chernoff bound. ■

4.4 Online Bipartite Matching on Forests

In this section, we provide an algorithm that achieves a switching cost of $O(n \log n)$, when the connection graph on n clients and n servers is a forest, and contains a perfect matching between clients and servers. The main result of this section is as follows.

Theorem 21 *Let G be the underlying graph of connections between clients and servers. If G is a forest, then, for any arrival order of clients, there is an algorithm which has a switching cost of $O(n \log n)$.*

Before we describe the algorithm, we first make some preliminary observations, which provide a foundation for stating our algorithm. Note that at the start of our online process, we have n server nodes and no edges, and thus our connection graph contains n connected components consisting of a single server node each. As new clients arrive with their edges, these connected components become merged. In particular, one should note that when a new client i arrives with $d_i \geq 2$ edges, the client's d_i edges must connect to d_i *distinct* components in the graph, since our final graph must be a forest. Thus a

client arriving with $d_i \geq 2$ edges causes d_i components to merge into a single connected component.

As our process proceeds, our online algorithm monitors these connected components, and designates a node from each connected component to be the *root* of the component. The root and size of each component are key elements in deciding the augmenting path to be used to connect a new client. At the beginning of the process, each server node starts as the root node of its own connected component; this will change as clients arrive and components merge. Now to define how our algorithm maintains the root nodes and selects augmenting paths to connect each new arriving client i, we have three different cases:

- **Case** *I*: If client i has a single edge, then this edge connects to a single connected component C and thus all potential augmenting paths from i must pass through component C. Among the set of potential augmenting paths, choose the augmenting path that stays furthest away from the root of C.

- If a client i has more than one edge, then we have two cases to consider, depending on the set of connected components to which client i can find an augmenting path. Let S_i denote the set of connected components to which client i can find an augmenting path, and let T_i denote the set of connected components to which client i has an edge.

  - **Case** *II*: If the set S_i contains only one connected component, and that connected component is the unique largest component of T_i, then follow the rule in case *I*.
  - **Case** *III*: Otherwise, choose any augmenting path into the smallest component in S_i.

  At the end of both case *II* and case *III*, all components in T_i have merged into a single component T. Assign T to have the same root node as the largest component in T_i, breaking ties arbitrarily if needed.

We now prove Theorem 21, by bounding the total switching cost of our algorithm.

Proof of Theorem 21: We first bound the total cost of augmentations arising from case *III* by $O(n \log n)$. Note that each time a server switches clients via case *III*, its component size at least doubles, and thus each server can switch clients at most $O(\log n)$ times via case *III* augmentations. Therefore, the total cost arising from case *III* augmentations is $O(n \log n)$.

Lemma 17 shows that the switching cost arising from case *I* and case *II* augmentations is at most $O(n \log n)$. The theorem thus follows by combining Lemma 17 with our bound on the total cost of augmentations arising from case *III*. ■

We now completes the proof of Theorem 21 by bounding the switching cost arising from case *I* and case *II* augmentations with the following lemma.

Lemma 17 *Let G be the underlying graph of connections between clients and servers. If G is a forest, then, for any arrival order of clients, the total switching cost of case I and case II augmentations is at most $O(n \log n)$.*

Note that for this lemma, we do not need to distinguish between case *I* and *II* augmentations; we just need to note that these augmentations seek to maximize the distance from the root node, and for servers involved in case *I* or case *II* augmentations, their root node does not change as a result of these augmentations.

Although we do not need to distinguish between these two types of augmentations anymore, we do need to further classify the client/server switches that arise as a result of these types of augmentations. For a server v which switches clients as a result of a case *I* or *II* augmentation, we classify v's switch as either an *upward* switch, a *downward* switch, or a *peak* switch, depending on the direction of the augmentation/switch relative to the root node. To classify these switches, suppose that a case *I* or case *II* augmenting path passes through a client u, a server v, and a client w in sequence and thus results in edge (u, v) being added to the matching and edge (w, v) being deleted from it. We say node v experiences an *upward* switch if client u is strictly closer to the component root than client w, and v experiences a *downward* switch if client u is strictly further from the component root than client w. If both client u and w are the same distance from the root, then server v experiences a *peak* switch. To complete the proof of the cost upper bound, we prove the following three statements:

- Each node experiences at most one downward switch.
- Each node experiences at most $O(\log n)$ upward switches.
- There are at most $O(n)$ peak switches in total.

Before we prove the three statements, we first define some terminology. For a set U of clients and servers, we say that U is *finished* if every client in U has already arrived, U contains an equal number of clients and servers, and no client in U is adjacent to a server in the complement of U. We say that a client or server is finished if there exists a finished set U that contains the client or server. Note that once a set U becomes finished it remains finished in the future. Also, all the clients in a finished set U must be matched to servers in U and there can be no augmenting paths passing through a node of U.

Now to prove the first statement, suppose that a server node v engages in a downward switch which results in its becoming connected to a client u. After this switch takes place, let U be the set consisting of v, u, and all the clients and servers reachable from u by an augmenting path that does not pass through v. It must be the case that every server in U is matched to a client in U, because otherwise u would have an augmenting path that stays strictly further from the root than any path passing through v, and therefore u would not have switched to v in the most recent step of the algorithm. Moreover, every client in U is matched to a server in U and no client in U is adjacent to a server $v' \notin U$, because this would imply the existence of an augmenting path from u to v' that does not pass through v, contradicting the assumption that $v' \notin U$. Thus, the set U is finished; in particular, this means v is finished, thus confirming that v can engage in at most one downward switch throughout the execution of the algorithm.

To prove the second statement, note that once a server v engages in an upward switch, if it engages in another switch before its component root changes then this switch must be a downward switch, which would finish v. Thus, the number of upward switches involving v is bounded above by the number of times the root of the component containing v changes. However, a server's root node can change at most $O(\log n)$ times, since the component size at least doubles each time the root changes. As a result, we have that each server may undergo at most $O(\log n)$ upward switches.

To prove that there are at most $O(n)$ peak switches, note that each augmentation of case I or case II, consists of a sequence of zero or more upward switches, a peak switch, and a sequence of zero or more downward switches. Thus, the second statement follows since each augmentation contains at most one peak switch and since there are at most n augmentations.

Thus, we have proven all three statements, and we have that the total cost of case I and II augmentations is $O(n \log n)$. ■

4.5 Conclusion

In this chapter, we have studied three variants of the online bipartite matching problem. First, we showed that when the underlying bipartite graph is arbitrary, and the clients arrive in a random order, the greedy algorithm, which always uses the shortest available switching path, achieves a switching-cost of $O(n \log n)$ with high probability. We also studied the problem when the arrival order is adversarial and the underlying graph is a forest, and provided a $O(n \log n)$ algorithm for this case. Finally, we studied the problem in random bipartite graphs of degree $O(\log n)$ and derived an algorithm which achieves $O(n)$ switching-cost.

The main open question is to find an algorithm which achieves a switching cost of $O(n \log n)$ for any arbitrary bipartite graph between clients and servers, and for an adversarial arrival order of clients. In particular, it would be interesting to find a proof that the greedy algorithm achieves a switching-cost of $O(n \log n)$ in any bipartite graph, for any arrival order of the clients, or to find a counterexample that suggests otherwise.

Bibliography

[ABK93] Yossi Azar, Andrei Broder, and Anna Karlin. On-line load balancing. In *Proc. 33rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 218–225, 1993.

[ABKU94] Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal. Balanced allocations (extended abstract). In *STOC '94: Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 593–602, New York, NY, USA, 1994. ACM Press.

[ACL00] W. Aiello, F. Chung, and L. Lu. A random graph model for massive graphs. In *STOC*, 2000.

[ACL+03] Marta Arias, Lenore J. Cowen, Kofi A. Laing, Rajmohan Rajaraman, and Orjeta Taka. Compact routing with name independence. In *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures (SPAA 03)*, pages 184–192, New York, NY, USA, 2003. ACM Press.

[ADTW03] E. Anshelevich, A. Dasgupta, É. Tardos, and T. Wexler. Near-optimal network design with selfish agents. In *Proceedings of the 35th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 511–520, 2003.

[AGLP89] Baruch Awerbuch, Andrew Goldberg, Michael Luby, and Serge Plotkin. Network decompositions and locality in distributed computation. 1989.

[AGM+04] I. Abraham, C. Gavoille, D. Malkhi, N. Nisan, and M. Thorup. Compact name-independent routing with minimum stretch. *The Sixteenth ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 04)*, 2004.

[AGZ96] Matthew Andrews, Michel X. Goemans, and Lisa Zhang. Improved bounds for on-line load balancing. In *Computing and Combinatorics, Second Annual International Conference (COCOON)*, pages 1–10, 1996.

[AKL+08] Christos Amanatidis, Richard M. Karp, Henry Lin, Christos H. Papadimitriou, and Martha Sideri. Linked decompositions, internet routing, and the power of choice in polya urns. In *Symposium on Discrete Algorithms (SODA)*, 2008.

[AKP+93] Yossi Azar, Bala Kalyanasundaram, Serge Plotkin, Kirk Pruhs, and Orli Waarts. Online load balancing of temporary tasks. In *Proc. 2nd International Workshop on Algorithms and Data Structures (WADS)*, pages 119–130, 1993.

[ANR92] Yossi Azar, Joseph Naor, and Raphael Rom. The competitiveness of on-line assignments. In *Proc. 3rd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 203–210, 1992.

[AP90] Baruch Awerbuch and David Peleg. Sparse partitions. 1990.

[BA99] A. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

[BMW56] M. Beckmann, C. B. McGuire, and C. B. Winsten. *Studies in the Economics of Transportation*. Yale University Press, 1956.

[BR02] B. Bollobás and O. Riordan. Mathematical results on scale-free random graphs. In *Handbook of Graphs and Networks*. Wiley-VCH, Berlin, 2002.

[Bra68] D. Braess. Uber ein paradoxon der verkehrsplanung. *Unternehmensforschung*, 12:258–268, 1968. Available from `http://homepage.ruhr-uni-bochum.de/Dietrich.Braess/`.

[cai] Cooperative Association for Internet Data Analysis. http://www.caida.org/home/.

[CCK+06] M. Caesar, T. Condie, J. Kumar Kannan, K. Lakshminarayanan, I. Stoica, and S. Shenker. ROFL: Routing on Flat Labels. *SIGCOMM*, 2006.

[CDKL] Kamalika Chaudhuri, Costis Daskalakis, Robert Kleinberg, and Henry Lin. Online bipartite matching with augmentations. In INFOCOM 2009.

[CHJ03] F. Chung, S. Handjani, and D. Jungreis. Generalizations of polya's urn problem. *Annals of Combinatorics*, 7:141–153, 2003.

[CS03] C. K. Chau and K. M. Sim. The price of anarchy for non-atomic congestion games with symmetric cost maps and elastic demands. *Operations Research Letters*, 31(5):327–335, 2003.

[CSM04a] J. R. Correa, A. S. Schulz, and N. E. Stier Moses. Computational complexity, fairness, and the price of anarchy of the maximum latency problem. In *Tenth Conference on Integer Programming and Combinatorial Optimization (IPCO)*, 2004. To appear.

[CSM04b] J. R. Correa, A. S. Schulz, and N. E. Stier Moses. Selfish routing in capacitated networks. *Mathematics of Operations Research*, 2004. To appear.

[Czuar] A. Czumaj. Selfish routing on the Internet. In J. Leung, editor, *Handbook of Scheduling*. CRC Press, Boca Raton, FL, 2004, to appear.

[DGK+04] N. Devanur, N. Garg, R. Khandekar, V. Pandit, and A. Saberi. Price of anarchy, locality gap, and a network service provider game. Unpublished manuscript, 2004.

[FFF05] A. Frieze, A. Flaxman, and T. Fenner. High degree vertices and eigenvalues in the preferential attachment graph. *Internet Mathematics*, 2:1–20, 2005.

[FGL+03] R. Feldmann, M. Gairing, T. Lücking, B. Monien, and M. Rode. Selfish routing in non-cooperative networks: A survey. In *Proceedings of the Conference on Mathematical Foundations of Computer Science (MFCS)*, volume 2747 of *Lecture Notes in Computer Science*, pages 21–45, 2003.

[FI05] FIND: Future Internet Network Design. http://find.isi.edu/, 2005.

[FLM+03] A. Fabrikant, A. Luthra, E. Maneva, C. H. Papadimitriou, and S. J. Shenker. On a network creation game. In *Proceedings of 22nd PODC*, pages 347–351, 2003.

[FPSS03] Dimitris Fotakis, Rasmus Pagh, Peter Sanders, and Paul G. Spirakis. Space efficient hash tables with worst case constant access times. In *Proc. 20th Annual*

Symposium on Theoretical Aspects of Computer Science (STACS), pages 271–282, 2003.

[GE05] GENI: Global Environment for Network Innovations. http://www.geni.net/, 2005.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[GKKV95] Edward F. Grove, Ming-Yang Kao, P. Krishnan, and Jeffrey Scott Vitter. Online perfect matching and mobile computing. In *Proc. 4th International Workshop on Algorithms and Data Structures (WADS)*, pages 194–205, 1995.

[GM08] Gagan Goel and Aranyak Mehta. Online budgeted matching in random input models with applications to adwords. In *SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 982–991, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

[God08] Brighten Godfrey. Balls and bins with structure: Balanced allocations on hypergraphs. In *Proc. 19th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2008.

[JMS00] O. Jahn, R. Möhring, and A. S. Schulz. Optimal routing of traffic flows with length restrictions in networks with congestion. In *Operations Research Proceedings 1999*, pages 437–442, 2000.

[JT04] R. Johari and J. N. Tsitsiklis. Network resource allocation and a congestion game. *Mathematics of Operations Research*, 2004. To appear.

[KFY04] D. Krioukov, K. Fall, and X. Yang. Compact routing on internet-like graphs. *INFOCOM*, 2004.

[Kkc] D. Krioukov and kc claffy. Toward compact interdomain routing. `arXiv:cs.NI/0508021`.

[KLadH92] Richard M. Karp, Michael Luby, and Friedhelm Meyer auf der Heide. Efficient pram simulation on a distributed memory machine. In *STOC '92: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 318–326, New York, NY, USA, 1992. ACM Press.

[KP99] E. Koutsoupias and C. H. Papadimitriou. Worst-case equilibria. In *Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science*, pages 404–413, 1999.

[KVV90] R. M. Karp, U. V. Vazirani, and V. V. Vazirani. An optimal algorithm for on-line bipartite matching. In *STOC '90: Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 352–358, New York, NY, USA, 1990. ACM.

[LRT04] H. Lin, T. Roughgarden, and É. Tardos. A stronger bound on braess's paradox. In *Proceedings of the 15th Annual Symposium on Discrete Algorithms (SODA)*, pages 333–334, 2004.

[LRTW05] Henry Lin, Tim Roughgarden, Eva Tardos, and Asher Walkover. Braess's paradox, fibonacci numbers, and exponential inapproximability. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, 2005.

[Mit01] Michael Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1094–1104, 2001.

[MPS03] Milena Mihail, Christos Papadimitriou, and Amin Saberi. On certain connectivity properties of the internet topology. In *FOCS '03: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, page 28, Washington, DC, USA, 2003. IEEE Computer Society.

[MSVV07] Aranyak Mehta, Amin Saberi, Umesh Vazirani, and Vijay Vazirani. Adwords and generalized online matching. *J. ACM*, 54(5):22, 2007.

[NA] NewArch Project: Future-Generation Internet Architecture. http://www.isi.edu/newarch/.

[Pap01] C. H. Papadimitriou. Algorithms, games, and the Internet. In *Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing*, pages 749–753, 2001.

[Per04] G. Perakis. The price of anarchy when costs are separable and asymmetric. In *Tenth Conference on Integer Programming and Combinatorial Optimization (IPCO)*, 2004. To appear.

[PR01] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. In *Proc. 9th Annual European Symposium on Algorithms (ESA)*, pages 121–133, 2001.

[PS89] David Peleg and Alejandro A. Schaffer. Graph spanners. 1989.

[PW93] Steven Phillips and Jeffrey Westbrook. Online load balancing and network flow. In *Proc. 25th ACM Symposium on Theory of Computing (STOC)*, pages 402–411, 1993.

[Rou] T. Roughgarden. Designing networks for selfish users is hard. In *FOCS '01*, pages 472–481.

[Rou02a] T. Roughgarden. How unfair is optimal routing? In *Proceedings of the 13th Annual Symposium on Discrete Algorithms*, pages 203–204, 2002.

[Rou02b] T. Roughgarden. The price of anarchy is independent of the network topology. In *Proceedings of the 34th Annual ACM Symposium on the Theory of Computing*, 2002. To appear.

[Rou04] T. Roughgarden. The maximum latency of selfish routing. In *Proceedings of the 15th Annual Symposium on Discrete Algorithms (SODA)*, pages 973–974, 2004.

[Rou05] T. Roughgarden. *Selfish Routing and the Price of Anarchy*. MIT Press, 2005.

[Rou07] T. Roughgarden. Selfish routing and the price of anarchy (survey). *OPTIMA No. 74*, 2007.

[RT02] T. Roughgarden and É. Tardos. How bad is selfish routing? *Journal of the ACM*, 49(2):236–259, 2002. Preliminary version in *FOCS '00*.

[RT04] T. Roughgarden and É. Tardos. Bounding the inefficiency of equilibria in nonatomic congestion games. *Games and Economic Behavior*, 47(2):389–403, 2004.

[Smi79] M. J. Smith. The existence, uniqueness and stability of traffic equilibria. *Transportation Research*, 13B:295–304, 1979.

[Vet02] A. Vetta. Nash equilibria in competitive societies, with applications to facility location, traffic routing and auctions. Manuscript, 2002.

[Wei01] D. Weitz. The price of anarchy. UCB CS294-1 final project, 2001.

[Wes95] Jeffrey Westbrook. Load balancing for response time. In *Proc. 3rd Annual European Symposium on Algorithms (ESA)*, pages 355–368, 1995.