Abstract

Title of dissertation:	Resource Allocation in Networked and Distributed Environments
	Srinivasan Parthasarathy Doctor of Philosophy, 2006
Dissertation directed by:	Professor Aravind Srinivasan Department of Computer Science

A central challenge in networked and distributed systems is resource management: how can we partition the available resources in the system across competing users, such that individual users are satisfied and certain system-wide objectives of interest are optimized? In this thesis, we deal with many such fundamental and practical resource allocation problems that arise in networked and distributed environments. We invoke two sophisticated paradigms – linear programming and probabilistic methods – and develop *provably-good approximation algorithms* for a diverse collection of applications. Our main contributions are as follows.

1. **Assignment problems:** An assignment problem involves a collection of objects and locations, and a load value associated with each object-location pair. Our goal is to assign the objects to locations while minimizing various cost functions of the assignment (determined by the load values). This abstract setting

models many applications in manufacturing, parallel processing, distributed storage, and wireless networks. We present a *single* algorithm for assignment which generalizes and unifies many classical assignment schemes known in the literature (*V. S. Anil Kumar, Madhav V. Marathe, Srinivasan Parthasarathy, and Aravind Srinivasan. Approximation Algorithms for Scheduling on Multiple Machines. IEEE FOCS 2005*). Our scheme is derived through a fusion of linear algebra and randomization. In conjunction with other ideas, it leads to novel guarantees for multi-criteria parallel scheduling, broadcast scheduling, and social network modeling (*Samir Khuller, Rajiv Gandhi, Srinivasan Parthasarathy, and Aravind Srinivasan. Dependent Rounding in Bipartite Graphs. To appear in Journal of the ACM; earlier version appears in IEEE FOCS 2002*).

2. **Precedence constrained scheduling:** We consider two precedence constrained scheduling problems, namely sweep scheduling (*V. S. Anil Kumar, Madhav V. Marathe, Srinivasan Parthasarathy, Aravind Srinivasan, and Sybille Zust. Provable Parallel Scheduling for Generalized Sweep Scheduling. To appear in Journal of Parallel and Distributed Computing; earlier version appears in IEEE IPDPS 2005*) and tree scheduling (*V. S. Anil Kumar, Madhav V. Marathe, Srinivasan Parthasarathy, and Aravind Srinivasan. Scheduling on Unrelated Machines under Tree-like Precedence Constraints. To appear in Algorithmica; earlier version appears in APPROX 2005*), which are inspired by emerging applications in high performance computing. Through a careful use of randomization, we devise the first approximation algorithms for these problems with

near-optimal performance guarantees.

3. **Wireless communication:** Wireless networks are prone to interference. This prohibits proximate nodes in the network from transmitting simultaneously, and introduces fundamental challenges in the design of wireless communication protocols. We develop fresh geometric insights for characterizing and reasoning about wireless interference. We combine our geometric analysis with linear programming and randomization, to obtain centralized and distributed algorithms for latency minimization (*V. S. Anil Kumar, Madhav V. Marathe, Srinivasan Parthasarathy, and Aravind Srinivasan. End-to-End Packet Scheduling in Wireless Ad Hoc Networks. ACM-SIAM SODA 2004*) and throughput capacity estimation in wireless networks (*V. S. Anil Kumar, Madhav V. Marathe, Srinivasan Parthasarathy, and Aravind Srinivasan. Algorithmic Aspects of Capacity in Wireless Networks. ACM SIGMETRICS 2005*).

In summary, the innovative use of linear programming and probabilistic techniques for resource allocation, and the novel ways of connecting them with application-specific ideas is the pivotal theme and the focal point of this thesis.

Resource Allocation in Networked and Distributed Environments

by

Srinivasan Parthasarathy

2006

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2006

Advisory Committee:

Professor Aravind Srinivasan, Chair / Advisor

Professor Michael O. Ball

Professor Samir Khuller

Professor V. S. Anil Kumar

Professor A. Udaya Shankar

To Amma, Appa and Niru

Acknowledgments

First and foremost, I wish to thank my advisor Aravind Srinivasan. I am deeply indebted to him for his guidance on all things technical. This includes defining and approaching research problems, reading and writing papers, preparing slides, giving talks, searching for jobs, and interviewing. I am even more grateful for his guidance on various non-technical aspects and ultimately shaping my attitude towards research and work. Each and every activity in which he is involved is defined by his rigorous pursuit of excellence combined with his immense empathy for the people around him; I hope to replicate these qualities in the future, at least in some small measure, as I step into my professional career.

I owe a lot to Samir Khuller, Madhav Marathe (V. Tech.), and Anil Kumar (V. Tech.) for their constant guidance and encouragement throughout my graduate life. Madhav – who was my mentor and Anil – who was my colleague during my internships at Los Alamos National labs, introduced me to the sweep scheduling and wireless latency problems which have eventually blossomed into large portions of this thesis. It has been a terrific journey for me since I met Madhav and Anil at Los Alamos, and they have been great collaborators and well-wishers ever since.

I adore Samir for his friendly disposition and easy-to-approach personality; he

has been a tremendous influence in my growth as a researcher and his counsel has been beneficial to me on numerous occasions. His course on *Approximation Algorithms* kindled my fascination for CS theory, and played a decisive role in my choice of a research area.

It is said that a great friend is someone who knows a lot about you but likes you nevertheless! I wish to express my gratitude to one such special friend, Rajiv Gandhi. I have gained a lot from his friendship and guidance, especially during my initial years at UMD.

This thesis grew out of my collaboration with the people mentioned above. All the results in this thesis were obtained in collaboration with my advisor, Aravind Srinivasan. In addition, the results in Chapter 4 are due to joint work with Samir Khuller and Rajiv Gandhi; the results in Chapters 3, 5, 6, 7, 8, and 9 represent collaborative work done with Madhav Marathe and Anil Kumar. Chapter 5 also features collaboration with Sibylle Zust (formerly, at Los Alamos National Labs), and Chapter 9 features joint work with Dave Levin (University of Maryland) and Deepti Chafekar (Virginia Tech.).

I thank Neil Spring and Amol Deshpande for reviewing my job application materials, and their insights about the job search process in general. I am especially thankful to Neil for supporting me through the Summer of 2006, and initiating me into the world of Internet measurements. I am grateful for the financial support I received through NSF (NSF Award CCR-0208005 and NSF ITR Award CNS-0426683). I also thank Los Alamos National Labs, and Bell Labs – Lucent for supporting me through summer internships (during summers 2002 - 2003, and 2004 respectively).

I am grateful to Udaya Shankar and Michael Ball who are members of my dissertation examination committee, and David Mount who was a member of my Ph.D. proposal committee.

I owe a great deal to my (former and present) colleagues and fellow students at Maryland for several stimulating discussions and the lively ambiance in my department. They include Suman Banerjee, Indrajit Bhattacharya, Vijay Gopalakrishnan, Srinivas Kashyap, Seungjoon Lee, Dave Levin, Julian Mestre, Arunesh Mishra, Ruggero Morselli, Tamer Nadeem, Arunchandar Vasan, Justin Wan, and Nan Wang, though I have certainly missed a few others. I especially cherish my collaborations (on projects not included in this thesis) with Rajiv Gandhi, Indrajit Bhattacharya, and Srinivas Kashyap.

I am thankful to my former roommates Arunchandar Vasan, Guruprasad Pundoor, and Sadagopan Srinivasan for putting up with me. While many delightful incidents come to my mind due to my association with them, I will always remember Sada's inspired (and perhaps unorthodox) culinary ventures and their unique consequences!

There was never a better way than a game of cricket to release the inevitable frustrations cooked up in grad school, and I will surely miss my cricket buddies Ashok, Ranga, and Indrajit among others in the future. I was also extremely lucky to enjoy the company of an incredible group of friends from DESI – Arunsankar, Saurabh, Ajay, Vinod, and Utsav to name a few – who made my stay at Maryland so enjoyable.

Finally, my most ardent note of thanks goes to my family – my wife Niru, my

parents, and my brother – for their unconditional love. I am grateful to Niru for her infinite kindness, patience, support, and faith in my abilities, even during the times when I am not deserving of such generosity. I am grateful to my parents for the countless sacrifices they have made for my sake. Dear Amma, Appa, and Niru – I love you and thank you again and again and again.

Srinivasan Parthasarathy

University of Maryland, College Park

July 2006

Contents

Chapter 1

Introduction

The recent past has witnessed the arrival of several exciting networking and distributed technologies. Advances in peer-to-peer networking, content distribution, massively parallel computing, and multihop wireless networking now enable us to offer new services or deploy new applications that were inconceivable in the past. However, as we continue to develop these novel and complex technologies, we are also required to contend with several deep and fundamental system design challenges. A cardinal challenge in such complex systems is resource management, where the goal is to utilize the limited available resources productively in order to best serve the needs of the users of the system. In this thesis, we study several fundamental and practical problems in resource management that arise in the context of networked and distributed environments.

All the questions we consider in this thesis share the same essential flavor and are of the following form: how can we allocate the available resources in the system to a set of competing users in such a way that the individual users are satisfied and certain overall system-wide objectives are optimized? The precise nature of the constraints imposed by the system and the exact objectives of interest depend upon the context of the application. For instance, the available resources could be link-

bandwidths in a network, processing elements in a distributed computing system, or storage space in a content distribution system; the objectives of interest could be the network throughput, the run-time of a parallel schedule, and the cost of accessing data in the content distribution system, respectively. We explore strategies for handling such problems that occur in a diverse collection of applications.

Most interesting optimization problems encountered in complex systems turn out to be NP-Hard; this is the case with all the problems we consider as well. This implies that efficient algorithms (whose run times are polynomial in the size of their input) for solving these problems *optimally* are unlikely to exist. In the absence of efficient mechanisms for finding the optimal solution, we are forced to seek efficient mechanisms which yield approximate solutions. Our goal in this work is the design of *provably-good* approximation algorithms for various resource optimization problems: we seek algorithms that come with a *quantitative guarantee* that the approximate solutions discovered by them is *not* too far away from the optimal.

We invoke two algorithmic devices in our quest, namely *linear programming based methods* and *probabilistic methods*. All the results derived in this thesis is founded on one or both of these paradigms. Indeed, both paradigms have yielded a slew of stunning results in the past three decades, and have played a profound role in the field of approximation algorithms (see for instance Vazirani [126], and Motwani and Raghavan [95]). Our contribution in this thesis is two-fold: (i) we combine linear programming and probabilistic methods with application specific ideas to derive novel approximation algorithms for several fundamental resource optimization problems in networking, and parallel & distributed systems; for most applications considered in this thesis, our results yield the current best (or the only) known analytical performance guarantees in the literature; (ii) we also design generic techniques for algorithm design which can handle a broad realm of related applications, and hence are of independent interest. In order to lay out the contributions of our work in more specific

terms, we need to discuss the context of our applications in more detail, and we do so next.

1.1 Motivating Applications and our Contributions

We study three broad classes of applications namely, assignment algorithms, precedence-constrained parallel scheduling, and wireless routing & scheduling in parts I, II, and III of this thesis respectively. The following is a brief synopsis of the problems we consider in each of these parts and our specific contributions.

1.1.1 Assignment Algorithms

We start with a fundamental assignment problem in the general setting of *unrelated* parallel machines. We are given a collection of jobs and machines, a running time associated with each job-machine pair, and a weight associated with each job; the term *unrelated* emphasizes the fact that machines may possess different operational characteristics and in general, the running time of a job on one machine may not be related to the running time of the same job on another machine. We need to assign the jobs to the machines, in a way which minimizes the maximum load on any machine (makespan), the total weighted completion time of the jobs, and the ℓ_p-norm of the machine loads. This abstract setting models many applications in the areas of manufacturing, parallel processing, and operations research. More significantly, assignment also appears as an important subproblem in the context of countless other applications such as peer-to-peer network design for streaming media applications [4], data-migration in distributed storage systems [68], scheduling in high-speed wireless networks [5, 22], max-min fair network routing and bandwidth allocation [70], and profit earning facility location [94].

Prior results: Optimizing any of the three objectives - makespan, weighted completion time, and ℓ_p-norm - is known to be NP-Hard and some of the seminal results in scheduling theory deal with *individually* optimizing *one* of them. The classical work of Lenstra, Shmoys, and Tardos [84] presents a linear-programming based 2-approximation algorithm for minimizing the makespan. Here and in the rest of this thesis, as per standard convention, a ρ-approximation algorithm for an optimization problem is an algorithm whose solution is never more than a multiplicative factor of ρ away from the optimal solution. The 2-approximation algorithm for makespan was generalized by Shmoys and Tardos [117] who showed how to to minimize a linear cost function of the assignment in addition to preserving the makespan approximation guarantee. Ever since their discovery, both these algorithms [84, 117] have continued to play a seminal role as important subroutines for assignment, and have featured in the solutions of various other optimization problems [4, 9, 68, 5, 22, 70]. Skutella [119] presented a randomized $\frac{3}{2}$-approximation algorithm for minimizing the weighted completion time. Recently, Azar and Epstein [9] used the approach of Lenstra, Shmoys, and Tardos [84] to develop a 2-approximation algorithm for minimizing the ℓ_p-norm of the machine loads, for any $p \geq 1$.

Our contributions: We present a *single* algorithm for assignment which generalizes and unifies all the above mentioned assignment schemes: i.e., we can employ our assignment algorithm to recover the best known approximation guarantees provided by the previous results [84, 117, 119, 9]. In addition, our algorithm yields several novel *multi-criteria* guarantees for assignment, which were not known earlier and cannot be obtained through the earlier approaches. The following is a subset of results yielded by our algorithm: (a) a $(2, \frac{3}{2})$-bicriteria approximation guarantee for *simultaneously* optimizing both makespan and weighted completion time; significantly, our bi-criteria guarantee matches the best known guarantee for each of these

objectives individually; (b) a constant-factor multi-criteria guarantee for simultaneously optimizing makespan, weighted completion time, and any given collection of integral ℓ_p-norms; (iii) a better-than-two approximation guarantee for minimizing the ℓ_p-norm of the machine loads, for any $p \geq 1$, thus improving upon the result of Azar and Epstein [9].

Our scheme is derived through a fusion of ideas from linear algebra and randomization. While the primary motivation for our scheme is the assignment problem, there are diverse application scenarios whose combinatorial requirements are similar to that of assignment. Interesting consequences ensue in such scenarios by massaging our scheme to fit the needs of the specific application context. Two prominent examples in this category are broadcast scheduling and random-graph modeling, where our approach leads to the current best known approximation guarantees. To summarize, *in the first part of this thesis, we develop a unified scheme for assignment which generalizes and improves upon the previous known guarantees for assignment and draws its power through a synthesis of linear algebra and randomization. We also show how to combine it with problem-specific insights to derive novel approximation algorithms across a broad spectrum of applications.*

1.1.2 Precedence-constrained scheduling

High-performance computing has evolved as the main enabling technology for scalable simulation and analysis of several important physical and biological processes. In the second part of this thesis, we consider two fundamental precedence-constrained parallel scheduling problems motivated by emerging applications in high performance computing. Our first problem is inspired by radiation transport methods, which are commonly used in the parallel simulation of a variety of physical phenomena such as medical imaging, nuclear reactor design, weapons effect, and the spread of forest fires [99, 104, 102]. In its generality, this process involves computing the propagation

of a radiation flux across an unstructured mesh (or network) of elements, by iteratively *sweeping* across the mesh in multiple directions. Each sweep involves solving a system of equations locally at each mesh element. However, each direction induces a partial order or a set of *precedence constraints* according to which this computation can proceed across different mesh elements. On a parallel computing system, our goal is to assign the computations at a mesh element to a processor, and schedule the computation across all directions, so that the precedence constraints are not violated, and the length of the schedule is minimized. Due to data locality and coupling considerations, we have a crucial additional constraint that a mesh cell must be processed on the same processor along each direction. Problems with similar requirements arise in several other high-performance computing applications, and here we formulate a combinatorial generalization of this problem that captures the sweep scheduling constraints, and call it the generalized sweep scheduling problem.

Next, motivated by applications such as evaluating large expression-trees in parallel, and fast simulation of tree-shaped physical processes, we introduce two *tree-scheduling problems*. In both these problems, we are given (a) a set of n jobs with *forest-shaped* precedence constraints, that introduce a partial order on the jobs; i.e., the undirected graph underlying the precedence constraints form a forest; (b) a set of m machines, each of which can process at most one job at any time; (c) as in the setting of unrelated parallel machines in part I of the thesis, an arbitrary set of values $\{p_{i,j}\}$, where $p_{i,j}$ denotes the processing time of job j on machine i. We need to assign each job to a machine, and run the jobs in an order consistent with the precedence constraints. Our goal in the first problem is to design assignment and scheduling algorithms which minimize the makespan objective, or the maximum time it takes for any job to complete; in the second problem, we wish to minimize the total weighted completion times of all the jobs in the system.

Prior results: Sweep scheduling has been a very active area of research due to its general applicability. However, all known approaches to this problem [99, 104, 102, 92, 90] are heuristics with *no* known provable performance guarantees. Approximation guarantees are not known for the tree-scheduling problem as well, for either the makespan or the weighted completion time objectives. In fact, the only case of precedence constrained scheduling on unrelated parallel machines for which non-trivial approximation algorithms are known is when the job assignment to the machines is given as part of the input, and the precedence-constraints are in the form of disjoint chains. This is the well-known job-shop scheduling problem, which itself enjoys a distinguished history in scheduling literature. The first approximation algorithm for job-shop scheduling was the breakthrough work of Leighton *et al.* [83, 82]; Leighton *et al.* [83, 82] introduced the *random-delays* technique, and almost all the subsequent approximation algorithms for job-shop scheduling [43, 49, 118] are based on variants of this technique. To the best of our knowledge, prior to our work, no known results were known for minimizing weighted completion time on unrelated machines in the presence of precedence constraints of *any* nontrivial kind.

Our contribution: Our main contribution here is to generalize the random-delays technique of Leighton *et al.* to handle the precedence constraints imposed by sweep scheduling and tree-scheduling problems. In the sweep scheduling problem, we show how to combine the random-delays with a randomized processor assignment of jobs, to derive a randomized polylogarithmic approximation algorithm; this the first and only known algorithm for sweep scheduling with provably good analytical performance guarantees.

For the tree-scheduling problem with the makespan objective, we first generalize the approach of Lenstra, Shmoys, and Tardos [84] to obtain an assignment of jobs to the machines; next, we combine the random-delays technique with a so-

phisticated tree-decomposition technique for deriving polylogarithmic approximation algorithms for the makespan minimization problem. We then demonstrate a reduction of the tree-scheduling problem with the weighted completion time objective, to the problem with the makespan objective; by exploiting our polylogarithmic approximation guarantee for makespan, and combining it with other ideas, we derive polylog approximation algorithms for the weighted completion time objective as well. To summarize, *our contribution in the second part of this thesis is to generalize the powerful random-delays technique of Leighton et al. [83, 82], in order to derive the first algorithms with provably-good performance guarantees for the sweep scheduling and tree-scheduling problems.*

1.1.3 Wireless Routing and Scheduling

The last decade has beheld astounding advances in the evolution and deployment of wireless technologies. A technology of special interest is wireless multihop networking, where network nodes communicate either directly or through the help of other intermediate nodes in the network, without the aid of any pre-provisioned routing infrastructure. This autonomous, self-configuring nature of wireless networks make them ideal candidates for a multitude of applications such as community (mesh) networking, sensing and monitoring, battlefield operations, and disaster recovery. At the same time, the unique characteristics of wireless signal propagation lead to conflicts or *interference* among proximate transmissions in the network. This phenomenon introduces fundamental challenges in the design of wireless communication protocols.

In the third part of this thesis, we investigate algorithms for wireless communication which can effectively deal with interference and utilize the network close to its capacity. We consider two central questions: (i) given a multihop wireless network and a collection of packets in the network (each packet encodes its source, its destination, and the path in the network it needs to traverse to reach from its source to

its destination), how can we schedule the transmission of packets across their links so that the schedule is interference-free and the packets reach their respective destinations as quickly as possible (latency minimization)? (ii) given a multihop wireless network, and a collection of source-destination pairs, what is the rate at which the sources should inject packets into the network towards their respective destinations, how should we route the packet flows from the source to the destination, and how can we schedule the links to avoid interference, so as to maximize the total rate at which data is carried in the network (throughput capacity)? Latency and throughput are arguably the two most important objectives of interest in communication networks, and our goal is to optimize these objectives in wireless networks subject to interference constraints.

Prior results: Latency minimization and throughput capacity are both well understood in the case of *wired* networks. As mentioned earlier, the influential work of Leighton *et al.* [83, 82] pioneered the use of random-delays technique and derived constant-factor approximation algorithms for latency minimization in wired networks. The throughput capacity problem in wired networks can be formulated as the well-known multi-commodity flow linear program and solved optimally in an efficient manner [1]. In the case of wireless networks, prior to our work, we are not aware of any interference-aware algorithms for end-to-end packet scheduling with near-optimal provably good performance guarantees. In their seminal work on the capacity of wireless networks, Gupta and Kumar [53] discovered precise mathematical bounds for throughput scaling in *randomly* constructed wireless networks under the assumption of *random* traffic demands. This has inspired a slew of other related results on throughput scaling in random wireless networks, under a variety of communication and infrastructure models [17, 88, 47, 52, 75, 109, 100, 128]. In contrast, we initiate the study of *algorithmic aspects of capacity in wireless networks*, and seek

efficient algorithms to estimate the capacity of an *arbitrary* wireless network under arbitrary traffic demands, both of which are *given to us* as part of the input.

Our contribution: Our contribution in this part of the thesis is two-fold: (i) we develop fresh geometric-packing insights for characterizing and reasoning about interference in wireless networks; (ii) we show how these insights can be combined with further ideas to derive near-optimal approximation algorithms for latency minimization and throughput capacity estimation and maximization. Specifically, through a fusion of geometric analysis and the random-delays technique [83, 82], we obtain near-optimal centralized *and* distributed scheduling algorithms for latency minimization. Next, we show how to combine geometric analysis with linear programming techniques to estimate the throughput capacity of a given wireless network, and to design joint routing, scheduling, and end-to-end rate allocation algorithms which achieves this capacity. This result can be viewed as an algorithmic equivalent of the results of Gupta and Kumar [53] on wireless capacity, and is very significant from the perspective of practical protocol design and analysis. Finally, through a combination of geometric insights, linear programming, and probabilistic analysis, we extend our algorithms for capacity estimation to the case of random-access wireless networks. Random-access protocols such as the IEEE 802.11 standard are ubiquitously used on wireless devices in practice, and our aim is to bring analytical techniques to bear upon such practical protocols. To summarize, *our contribution in the third part of this thesis is to develop key geometric insights to characterize wireless interference, and combining it with probabilistic techniques and linear programming to derive provably-good interference-aware algorithms for wireless communication.*

1.2 Interplay between theory and applications

The key theoretical tools we employ, the applications we investigate, and the interplay between theory and applications in this thesis are illustrated in Figure 1.1. Linear programming and probabilistic methods are the theoretical techniques which lie at the heart of all our applications; further, the geometric analysis of wireless interference plays a critical role in all the wireless applications. Our algorithms and their performance guarantees come about by manipulating the underlying theoretical tools to fit the needs of individual applications. Along the way, we device two comprehensive templates for the design and analysis of resource allocation algorithms, namely, the unified scheme for assignment, and the geometric analysis of wireless interference. These templates have already yielded the numerous applications presented in this thesis, and promise to be of independent interest in the future. In summary, the use of linear programming and probabilistic techniques, and the innovative ways of combining them with various fundamental questions and practical applications in resource optimization is the pivotal theme and the focal point of our work.

1.3 How to read this Thesis

This thesis is aimed at audience from three different streams: computer science, operations research, and electrical engineering. While all parts of the thesis could be of interest to computer scientists, parts I and II of the thesis could be of interest to operations researchers, while part III could be of interest to electrical engineers. Our goal is to make this thesis accessible to any one with a basic background in probability, and algorithm design & analysis. Most of the required concepts such as approximation algorithms, and the role of linear programming and probabilistic methods in the design of approximation algorithms are either introduced in Chapter 2 as background material or built up along the way as and when required. We also

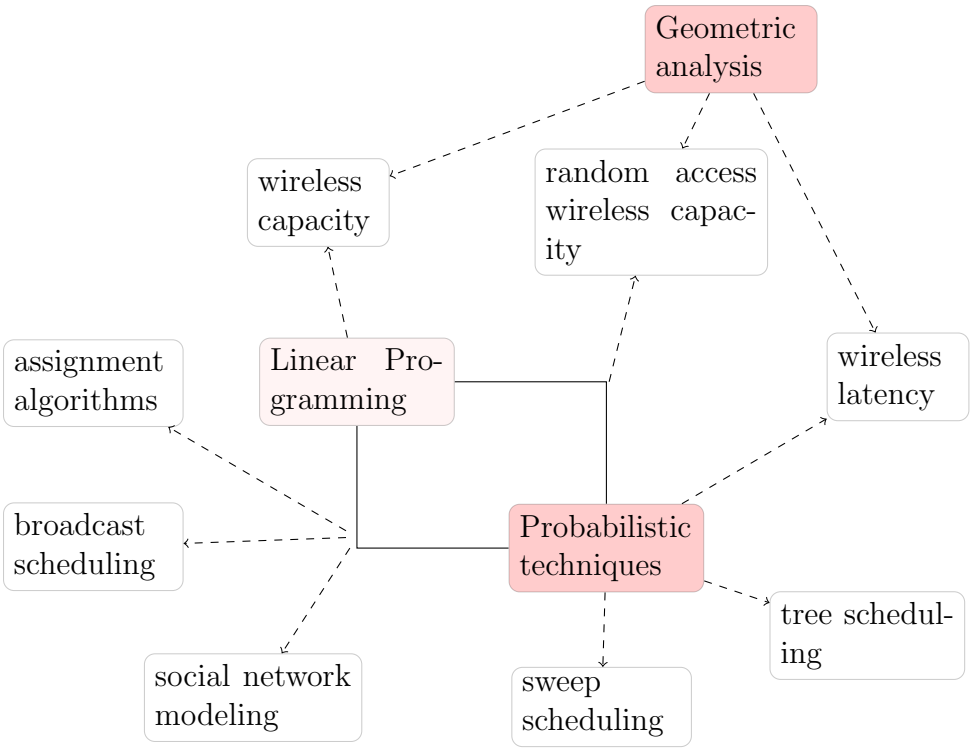


Figure 1.1: Interplay between theory and applications: the main theoretical tools (shaded boxes) at the heart of all our applications are linear programming and probabilistic techniques; in addition, geometric analysis plays a key role in the wireless applications. The arrows denote the dependencies between the theoretical techniques, and the applications (unshaded boxes).

include a formal description of the results obtained in this thesis in Chapter 2. Readers who are broadly familiar with the area of approximation algorithms can focus only on the description of our results, and skip the background material in Chapter 2.

Chapters 3 and 4 deal with assignment algorithms and comprise the first part of this thesis. In Chapter 3, we develop our unified algorithm for assignment in the context of the unrelated parallel machine scheduling problem, and derive the principal properties of our algorithm. In Chapter 4, we exploit these properties for developing our approximation algorithms for broadcast scheduling and social network modeling. Thus, some Sections in Chapter 3 are prerequisites for reading Chapter 4. Part II of the thesis is comprised of Chapters 5 and 6, which deal with approximation algorithms for sweep-scheduling and tree scheduling respectively. The novel use of the

random-delays technique [83, 82] is the common feature in both these applications. These chapters can both be read independently of all other chapters in this thesis. Chapters 7, 8, and 9 deal with wireless applications and comprise part III of this thesis. Chapter 7 deals with latency minimization: we develop our key geometric insights on wireless interference in this chapter, and show how to combine geometric insights with the random-delays technique to solve the latency minimization problem. Note that Chapter 7 also employs the random-delays technique, and is similar to Chapters 5 and 6 in this respect. In Chapter 8, we show how to combine our geometric insights with linear programming techniques to estimate the throughput capacity of wireless networks. In Chapter 9, we extend the basic techniques we develop in Chapter 8 to random-access wireless networks. Thus, some Sections in Chapter 8 are prerequisites for reading Chapter 9, while some Sections in Chapter 7 are prerequisites for both Chapters 8 and 9. We present our conclusions and the main open problems inspired by our work in Chapter 10.

Chapter 2

Preliminaries and Detailed Results

In this chapter, we review the basic definitions and notation used in this thesis. We start with a quick primer on approximation algorithms (Section 2.1) followed by the role of linear programming based techniques, and probabilistic techniques in the design of approximation algorithms (Sections 2.2 and 2.3). Following this, we present a more detailed explanation of the results obtained in the latter chapters, with an emphasis on how linear programming and probabilistic techniques are employed in our applications.

2.1 Approximation Algorithms

Many practical optimization problems are NP-Hard. While it is possible to define NP-Hard problems and the notion of approximation algorithms for them precisely, this definition is very technical in nature and we refer the interested reader to Vazirani [126]. For our purposes, it suffices to treat NP-Hard problems as optimization problems which are unlikely to admit efficient algorithms for solving them exactly (efficient algorithms are algorithms that run in polynomial time in the size of their input). In the absence of efficient algorithms for solving a problem exactly, we need to derive efficient algorithms for solving the problem *approximately*. Throughout our study,

our main focus will be on the *quality of approximation* guaranteed by our approximation algorithms; while we almost never discuss the running times of our algorithms, it will mostly be clear from the context that they are indeed polynomial-time. We now recall the notion of an approximation algorithm.

An optimization problem $\mathcal{P}$ is either a maximization or a minimization problem. Each valid instance $\mathcal{I}$ of $\mathcal{P}$ has a non-empty set of feasible solutions, each of which is assigned a non-negative objective function value. A feasible solution that achieves the optimal objective function value is called an optimal solution. Given an optimization problem $\mathcal{P}$ and an instance $\mathcal{I}$ of $\mathcal{P}$, let $OPT(I)$ denote the optimal objective function value for instance $\mathcal{I}$. If $\mathcal{P}$ is a maximization problem, an approximation algorithm $\mathcal{A}$ for problem $\mathcal{P}$ is an efficient algorithm which, for every input instance $\mathcal{I}$ of $\mathcal{P}$, produces a solution whose objective function value is at least $\frac{OPT(I)}{\lambda}$; here, $\lambda \geq 1$ is the *approximation ratio* or *approximation guarantee* provided by algorithm $\mathcal{A}$. Analogously, if $\mathcal{P}$ is a minimization problem, an approximation algorithm $\mathcal{A}$ for $\mathcal{P}$ produces, for every input instance $\mathcal{I}$ of $\mathcal{P}$, a solution whose objective function value is *at most* $\lambda OPT(I)$, where $\lambda \geq 1$ is the approximation guarantee.

Designing an approximation algorithm for a given problem typically involves unraveling the intricate combinatorial structure that lies beneath the problem. While this unraveling process is often very problem-specific, there are a few design paradigms in approximation algorithms that have evolved over the past few years, which are suitable for a broad realm of problems. The most telling examples of such paradigms include linear programming (or more generally mathematical programming) based techniques, primal dual algorithms, local search heuristics, and probabilistic techniques. All the application in this thesis come about through the use of two of these sophisticated paradigms namely, linear programming based techniques, and probabilistic techniques. We now briefly review the typical role played by these paradigms in the design of approximation algorithms.

2.2 Linear Programming in Approximation Algorithms

Linear programming is the problem of optimizing a linear objective function subject to a collection of linear inequality constraints. The most important fact about linear programming from our perspective is that exact algorithms are known for solving them in polynomial time. This fact yields the following arsenal for attacking NP-Hard optimization problems. Given an instance $\mathcal{I}$ of an optimization problem $\mathcal{P}$, we can obtain a *relaxation* of $\mathcal{I}$ using a linear program (LP). This LP relaxation *enlarges* the set of feasible solutions for instance $\mathcal{I}$, and can be solved optimally in polynomial time. The optimal LP solution is useful for two related reasons: first, if $\mathcal{P}$ is a maximization (minimization) problem, the optimal value of the LP relaxation yields an upper bound (respectively, lower bound) on the optimal value of instance $\mathcal{I}$, $OPT(I)$. Next and more significantly, the optimal LP solution also gives us useful hints which can guide us in designing a good solution which is feasible for the original instance $\mathcal{I}$. In particular, suppose we have an efficient algorithm for converting (*rounding*) the LP solution $\mathcal{S}'$ to a feasible solution $\mathcal{S}$; further, suppose for *any* given instance $\mathcal{I}$, the rounded solution $\mathcal{S}$ has an objective function value which is at most a factor of λ away from $OPT(I)$; then, we have an approximation algorithm for $\mathcal{P}$ whose approximation ratio is λ. For a given problem, we seek efficient rounding schemes which lead to provably-good approximation algorithms with an approximation ratio λ close to one. The term *rounding* signifies the fact that, the LP is obtained by relaxing integrality constraints on variable (such as $x_i \in \{0, 1\}$) to their real analogs ($x_i \in [0, l]$), and the rounding algorithm needs to convert the potentially non-integral values in the solution to appropriate integers.

We now illustrate the LP rounding paradigm in the context of the assignment problem for unrelated parallel machines (studied in Part-I of the thesis). Recall that

in this problem, we are given a set of n jobs, a set of m machines, and a running time $p_{i,j}$ associated with the job j - machine i pair. Consider the following question: given a target T, we wish to compute assignment of jobs to machines such that the total load (or the sum of running times) on any machine i is at most T. This problem can be formulated as an *integer program* as follows:

$$\sum_i X_{i,j} = 1 \qquad \forall \text{ jobs } j \tag{2.1}$$

$$\sum_j X_{i,j} p_{i,j} \leq T \qquad \forall \text{ machines } i \tag{2.2}$$

$$X_{i,j} = 0 \qquad \forall \{i,j\} \text{ such that } p_{i,j} > T \tag{2.3}$$

$$X_{i,j} \in \{0,1\} \qquad \forall \{i,j\} \tag{2.4}$$

The interpretation of the above integer program is as follows. $X_{i,j}$ is the indicator variable which is set to 1 if job j is assigned to machine i, and set to 0 otherwise. Constraints (2.1) enforce the fact that each job must be assigned to some machine, and constraints (2.2) and (2.3) represent the fact the load on any machine should not exceed T. Since, this assignment problem is NP-Hard, there is no known polynomial algorithm which can check if the integer program is feasible, and produce a feasible solution (if one exists). However, we can obtain a linear relaxation by replacing the integrality constraints (2.4) with linear constraints $X_{i,j} \in [0,1]$ for all $\{i,j\}$. If this LP is infeasible (which can be checked in polynomial time), then clearly the integer program and hence the input instance is infeasible. More significantly, if the LP has a feasible solution, the classical work of Lenstra, Shmoys, and Tardos [84] demonstrates how to round this fractional LP assignment into a feasible integral solution, so that the maximum load of any machine in the integral solution is at most $2T$. Combined with the fact that we can perform a bi-section search for the minimum (fractionally) feasible target T, we have a 2-approximation for the problem of minimizing the maximum machine load in unrelated parallel machine assignment.

This general template of rounding infeasible fractional solutions (obtained through LP, or more generally mathematical programming based relaxations) into integral feasible solution is at the heart of several approximation algorithms in general, as well as many algorithms developed in this thesis.

2.3 Randomized Algorithms

Randomized algorithms are algorithms which make random choices during the course of their execution. Often, the power of a randomized algorithm arises from the fact that there is no single worst-case input which can drive the algorithm to perform poorly – rather, the random choices made during the execution of the algorithm guarantee that the performance of the algorithm is good *in expectation, or with high probability* on any given input. We now briefly discuss three specific ways in which randomization is employed, both in the context of this thesis as well as broadly in the design of algorithms.

Recall from Section 2.2 that the use of linear programming in approximation algorithms (i) involves casting the optimization problem into an LP by relaxing the integrality constraints, (ii) solving the LP, and (iii) rounding the LP solution into an integral solution. Randomization is a natural ingredient during the rounding stage. For instance, we may use *independent randomized rounding* for a 0/1 optimization problem where the variables are binary valued (e.g., the assignment problem), as follows: if the value of a decision variable x in the LP solution is $\alpha \in [0, 1]$, we may set $x = 1$ with probability α, and $x = 0$ with $1 - \alpha$ in the rounded solution. If the objective function is a linear function of the decision variables, it follows immediately that the expected value of the rounded solution equals the fractional objective value yielded by the LP. The pioneering work of Raghavan and Thompson [108] introduced the independent randomized rounding technique; this has now blossomed into an

indispensable tool in LP rounding based algorithms (see for instance, the survey by Srinivasan [122] for several applications of this technique).

A major issue which we need to deal with in randomized rounding is constraint violation: in general, the rounded variables will violate the constraints which were part of the LP, and which were originally satisfied by the fractional LP solution. Proving the quality of the rounded solution in such scenarios involves bounding the extent to which each constraint is violated by the rounded solution. For this purpose, we will exploit a very desirable property of random variables: when we aggregate a large number of independent 0/1 random variables, their sum is very sharply concentrated around its mean value. This is the second setting in which probabilistic techniques play a crucial role. This sharp concentration property, captured by the celebrated Chernoff-Hoeffding bounds [30, 58], is of independent interest and has played a central role in numerous applications [95]. The random-delays technique of Leighton *et al.* [82, 83] is an important setting in which these bounds are employed, and our results in Chapters 5, 6, and 7 (all of which are based on the random-delays technique) will repeatedly make use of Chernoff-Hoeffding and related bounds.

A third setting in which randomization plays a central role is distributed symmetry breaking. Suppose we have a single resource which needs to be accessed by multiple *uncoordinated* users over time. If multiple users access the resource at the same time, then a conflict arises and none of them can successfully gain access to the resource. Since the users are uncoordinated, they cannot communicate amongst themselves to arbitrate the use of the resource. A randomized protocol in this scenario involves each user attempting to access the resource during a time slot *probabilistically*. Choosing the access probabilities for each user carefully can ensure that the resource is utilized well, and each user experiences conflicts with low probability. Randomized contention resolution is at the heart of practical distributed channel access protocols in Ethernet, optical networks, and wireless networks. Crucially, randomization is not

just one attractive alternative in such distributed symmetry breaking applications, but often the *only* possible solution.

2.4 Detailed Results

We now present a synopsis of the major results derived in this thesis. In the description below, our focus is on the performance guarantees we provide for various applications, and the key technical novelties that underlie our algorithms.

Assignment algorithms: The first part of this thesis deals with assignment algorithms for a range of problems arising in the settings of unrelated parallel machines, broadcast scheduling, and social network modeling. Our main technical innovation here is a *dependent randomized rounding algorithm*, which takes as input a fractional assignment (typically obtained through a linear or convex programming relaxation of a constrained optimization problem), and *rounds* it probabilistically into an integral assignment. The term *dependent* underscores the fact that, unlike independent randomized rounding, the various random choices made by our algorithm are not independent of each other, but depend upon each other in a careful manner. The dependent rounding algorithm satisfies three desirable probabilistic properties, namely, marginal distribution, load preservation, and negative correlation (see Chapter 3 for descriptions of these properties). Our results come about by exploiting these properties in a problem-specific manner.

We develop the dependent rounding scheme in Chapter 3, and describe its use in multi-criteria approximation algorithms for unrelated parallel machine assignment. Some significant results we present here include (i) a $(2, \frac{3}{2})$-bicriteria approximation algorithm for simultaneously optimizing makespan and the weighted completion time of the assignment; notably, both the components of our bicriteria guarantee match

the best known approximation ratios for the respective individual objectives. Thus, improving either of the two components even while arbitrarily worsening the other would be a significant breakthrough; (ii) a 3.2-factor multi-criteria approximation algorithm for simultaneously optimizing makespan, weighted completion time, and any given collection of integral ℓ_p norms; (iii) a better-than-two approximation guarantee for minimizing ℓ_p-norm, for any fixed $p > 1$. Our first two results are the best known multicriteria results of their kind, and the third result improves upon the recent work of Azar and Epstein [9]. It is interesting to note that while the dependent rounding scheme is fundamentally based on linear algebraic principles, it also exhibits probabilistically good behaviour with respect to certain classes of non-linear (but convex) objectives such as weighted completion time, and ℓ_p-norms.

In Chapter 4, we present two further applications of the dependent rounding scheme. The first application pertains to modeling connectivity in social networks. The explosive growth of Internet, WWW, and other such massive networks has lead to a tremendous interest in modeling such networks using appropriate random graphs [42, 129, 33]. In particular, the uncovering of the power-law behavior of the vertex-degrees of many such graphs (see, e.g., [36, 34, 35]) has lead to much interest in generating (and studying) random graphs with a given degree-sequence (see, e.g., [50]). Web/Internet measurements capture a lot of connectivity information in the graph, in addition to the distribution of the degrees of the nodes. Our question here is: since a network is much more than its degree sequence, can we model connectivity in addition to the degree-sequence? Concretely, given n, connectivity values between pairs of nodes $\{x_{i,j} \in [0,1] \ : \ i < j\}$, and a degree-sequence $d_1, d_2, \ldots, d_n$ (realized by the values $x_{i,j}$), we wish to generate an n-vertex random graph $G = (\{1, 2, \ldots, n\}, E)$ in which: **(A1)** vertex i has degree d_i with probability 1, and **(A2)** the probability of edge (i, j) occurring is $x_{i,j}$ (note that we must have $d_i = \sum_j x_{i,j}$). Unfortunately, there exist simple input instance for which no space of random graphs satisfy **(A1)**

and **(A2)** simultaneously; hence we need to compromise to some extent. We design a graph sampling algorithm which satisfies property **(A2)**, but violates the degree requirement for any vertex by an additive factor of at most 2. This essentially achieves the best possible result, and is significantly better than what is achievable through independent sampling techniques.

Our second application in Chapter 4 is broadcast scheduling, which is motivated by the recent growth in multimedia technologies. The key feature of the service model here is that the service for certain requests can be *batched* and processed together: e.g., users waiting to receive the same movie from a satellite server, simultaneously get satisfied when their movie is broadcast. In our setting, the broadcast server has m distinct pages and can transmit at most one page during each time slot. Requests for various pages arrive over time, and a request for page p which arrives at slot t is serviced at the first time slot $t' > t$ such that the server broadcasts page p during slot t'. In this case, the service time for this request is $t' - t$. Given the set of requests, our goal is to assign the pages to slots such that the total service time for all the requests is minimized. We study this problem under the resource augmentation framework introduced by Kalyanasundaram *et al.* [65]: we allow our server to broadcast up to α pages during a slot (α-*speed* server), and attempt to produce a solution whose cost is at most a factor β away from the optimal solution of a 1-speed server (β-approximation). We devise a 2-speed 1-approximate or a $(2, 1)$-factor solution, which improves upon a long line of results due to Kalyanasundaram *et al.* [65] ($(3, 3)$-factor), Erlebach and Hall [39] ($(6, 1)$-factor), and Gandhi *et al.* [44] ($(2, 2)$, $(3, 1.5)$, $(4, 1)$-factors).

Precedence constrained scheduling: In the second part of this thesis, we consider two precedence-constrained parallel scheduling problems motivated by applications in high performance computing. The common feature in these problems is the novel use

of the random delays technique introduced by Leighton *et al.* [82, 83] in conjunction with other ideas.

We consider the sweep scheduling problem in Chapter 5. This arises in the parallel simulation of many large-scale physical processes such as medical imaging, nuclear reactor design, weapons effect, and forest fire modeling [99, 104, 102]. In the sweep scheduling problem, we are given an underlying graph $G = (V, E)$, whose n vertices represent distinct computations which need to be carried out on a set of m processors. Further, we are given a set of k directions; each direction specifies a set of precedence conditions for scheduling the computations, and the vertices must be processed in each direction i according to the precedence constraints imposed by direction i. Due to data locality considerations, we have a crucial additional requirement that computations associated with a fixed vertex across different directions need to be performed on the same processor. Our goal is to derive an assignment of vertices to processors, and schedule the computations across each direction such that the precedence constraints are not violated and the length of the schedule is minimized. We propose an algorithm based on the random-delays technique and randomized assignment of vertices to processors, whose approximation ratio is $O(\log^2 n)$ (where, $n = |V|$). We then show a slightly modified algorithm which, coupled with a much improved analysis, leads to an $O(\log \log \log \log m)$-approximation. In contrast to existing heuristics [99, 104, 102], ours is the first known algorithm for sweep scheduling with provably good performance guarantees.

In Chapter 6, motivated by fast parallel evaluation of large expression trees, and fast simulation of tree shaped physical processes, we considered parallel scheduling with tree-shaped precedences. We present polylogarithmic approximation algorithms for minimizing the makespan and weighted completion time in the setting of unrelated parallel machine scheduling, when the graph underlying the precedence relations forms a forest. For the makespan minimization problem, we first

obtain an assignment of jobs to the machines by modifying the assignment algorithm of Lenstra, Shmoys, and Tardos [84]; we then show how to combine the random delays technique with a novel tree-decomposition technique and obtain an $O(\frac{\log^2 n}{\log\log n}\lceil\frac{\log \min(p_{\max},n)}{\log\log n}\rceil)$ approximation for makespan. Here, n is the total number of jobs and $p_{\max}$ is the length of the longest job on any machine. When the forests are in/out-directed arborescences, we show how to derive an improved approximation guarantee of $O(\log n\lceil\frac{\log \min(p_{\max},n)}{\log\log n}\rceil)$. For the special case when all processing times are unit length, this becomes $O(\log n)$. Next, we show a reduction from the problem of minimizing weighted completion time to the problem of minimizing the makespan, and exploit this to devise algorithms with polylogarithmic performance guarantees for the former problem.

Wireless networks: The third part of this thesis deals with end-to-end algorithms for routing and scheduling in wireless networks. In Chapter 7, we start with the problem of minimizing end-to-end latency minimization in wireless networks. Here, we are given a wireless network $G = (V, E)$ and a collection of packets $\{1, \ldots, k\}$; packet i encodes its source s_i, its destination t_i, and the path $\mathcal{P}_i$ in the network which it needs to traverse starting at s_i and terminating at t_i. Each link in the network can transmit at most one packet per time slot, and our goal is to schedule the movement of packets across the links in their paths, in order to minimize the end-to-end latency (or the maximum number of slots it takes any packet to reach its destination). The key issue which we need to contend with is wireless interference, which prohibits transmissions on nearby links in the networks at the same time. In our work, under the standard disk graph model for network and geometric models for wireless interference, we derive *distributed* and *centralized* end-to-end scheduling algorithms for latency minimization with polylogarithmic performance ratios. A key driver in this work is the geometric analysis we employ for dealing with interference, and the novel

ways of combining this analysis with the random delays technique of Leighton *et al.* [82, 83]. For general disk graphs, we present a distributed $O(\log^2 n(1 + \lceil \log \frac{r_{\max}}{r_{\min}} \rceil))$-approximation algorithm where $r_{\max}$ and $r_{\min}$ are the maximum and minimum node transmission radii, and a centralized $O(\log n)$-approximation algorithm. For unit-disk graphs, we present a distributed $O(\log n)$-approximation algorithm and a centralized $O(1)$-approximation algorithm. The above distributed algorithms are in the synchronous model of communication; for unit-disk graphs, we also obtain a distributed $O(\log^2 n)$ approximation in an asynchronous communication model.

In Chapter 8, we consider the problem of throughput capacity estimation and maximization in wireless networks. As in the setting of Chapter 7, we are given a wireless network $G = (V, E)$ which is subject to interference, and a collection of source-destination pairs $\{s_i, t_i\}$. Here, we are concerned with the *rates* at which data can be sent from the sources to the destinations in the network. Specifically, we are given the link capacities which specify the maximum rate at which each link in the network can transmit data, and our goal is to determine (i) the rate r_i at which source s_i should inject data into the network for its destination (ii) the path(s) along which the data flow from each source must be routed to its corresponding destination, and (iii) an interference free scheduling algorithm for activating the links; our objective is to maximize the total end-to-end throughput $\sum_i r_i$. We derive the first constant factor approximation algorithm for this problem under various models of interference by combining the geometric analysis of wireless interference developed in Chapter 7, with linear programming techniques. A key ingredient in this work is an inductive scheduling protocol for scheduling the links of the network in an interference-free manner, which is potentially of independent interest. All our algorithms and proof techniques generalize to the case when we have concave utility functions of throughput, and constraints pertaining to end-to-end fairness, energy and dilation. This result can be viewed as an algorithmic version of the seminal work of Gupta

and Kumar [53] who defined and studied the throughput capacity of *random* wireless networks with *random* traffic patterns, as opposed to an arbitrarily specified wireless network with given traffic demands.

In Chapter 9, we initiate the study of capacity estimation in multihop wireless networks whose nodes employ *random-access* scheduling protocols for communication. Random-access scheduling protocols such as the IEEE 802.11 standard, are ubiquitous mechanisms for scheduling in wireless devices. However, the complex stochastic processes which underlie these mechanisms are inherently non-convex and do not readily lend themselves to efficient modeling and optimization. Our contributions here are as follows: (i) we formulate the capacity estimation problem for a broad class of random-access protocols using a non-linear program (NLP); although, this NLP is computationally intractable, we show that it can be approximated provably well by a linear program. This yields a powerful tool for capacity planning and analysis in random-access wireless networks; (ii) using our analysis, we precisely quantify what two existing distributed routing metrics (ETX [37] and ETT [38]) represent; we also develop the Available Capacity Metric (ACM), which more accurately reflects the quality of a link and results in better end-to-end throughput, without incurring any additional overhead compared to ETX or ETT. The broad goal of our work in this chapter is to bring our analytical insights to bear upon wireless protocols that are deployed widely in practice.

Most of the results presented in this thesis have also been published (often, in their preliminary forms) in peer-reviewed journals and conferences. The results from Chapters 3, 4, 5, 6, 7, and 8 respectively appear in [78], [45], [79], [80], [76], and [77]. We present our concluding remarks in Chapter 10, along with a discussion of the main open problems that are inspired by this thesis, and some directions for future research.

Chapter 3

Assignment Algorithms for Unrelated Parallel Machines

3.1 Introduction

The complexity and approximability of scheduling problems for multiple machines is an area of active research [81, 116]. A particularly general (and challenging) case involves scheduling on *unrelated parallel machines*, where the processing times of jobs depend arbitrarily on the machines to which they are assigned. That is, we are given n jobs and m machines, and each job needs to be scheduled on exactly one machine; we are also given a collection of integer values $p_{i,j}$ such that if we schedule job j on machine i, then the processing time of operation j is $p_{i,j}$. Three major objective functions considered in this context are to minimize the weighted completion-time of the jobs, the L_p norm of the loads on the machines, and the maximum completion-time of the machines, or the *makespan* (i.e., the L_∞ norm of the machine-loads) [84, 117, 119, 9]. Apart from its traditional applications in parallel scheduling, this abstract setting models many applications in the areas of manufacturing, distributed storage, operations research and wireless networking. More significantly, such assign-

ment problems also appears as an important subproblem in the context of countless other applications such as peer-to-peer network design for streaming media applications [4], data-migration in distributed storage systems [68], scheduling in high-speed wireless networks [5, 22], max-min fair network routing and bandwidth allocation [70], and profit earning facility location [94].

There is no measure that is considered "universally good", and therefore there has been much interest in *simultaneously* optimizing many given objective functions: if there is a schedule that simultaneously has cost T_i with respect to objective i for each i, we aim to efficiently construct a schedule that has cost $\lambda_i T_i$ for the ith objective, for each i. (One typical goal here is to minimize $\max_i \lambda_i$.) Most of the best results for these single-criterion or multi-criteria problems are based on constructing fractional solutions by different linear programming (LP)-, quadratic programming-, and convex programming-relaxations and then rounding them into integral solutions. Two major rounding approaches for these problems are those of [84, 117], and standard randomized rounding [108] as applied to specific problems in [119, 9].

In this chapter, we develop a single rounding technique that works with all of these relaxations, gives improved bounds for scheduling under the L_p norms, and most importantly, helps develop schedules that are good for multiple combinations of the completion-time and L_p-norm criteria. For the case of simultaneous weighted completion time and makespan objectives, our approach yields a bicriteria approximation with the best known guarantees for both these objectives. We start by presenting our applications, and then discuss our rounding technique.

(i) **Simultaneous approximation of weighted completion-time and makespan.** In the weighted completion-time objective problem, we are given an integral weight w_j for each job; we need to assign each job to a machine, and also order the jobs assigned to each machine, in order to minimize the weighted completion-times of the

jobs. The current-best approximations for weighted completion-time and makespan are 3/2 [119] and 2 [84], respectively. We construct schedules that achieve these bounds *simultaneously*: if there exists a schedule with (weighted completion-time, makespan) $\leq (C, T)$ coordinate wise, our schedule has a pair $\leq (1.5C, 2T)$. This is noticeably better than the bounds obtained by using general bicriteria results for (weighted completion-time, makespan) such as Stein and Wein [123] and Aslam *et al.* [6]: e.g., we would get $\leq (2.7C, 3.6T)$ using the methods of [123]. More importantly, note that if we can improve one component of our pair $(1.5, 2)$ (while worsening the other arbitrarily), we would improve on the current-best approximation known for weighted completion-time or makespan.

(ii) **Minimizing the L_p norm of machine loads.** Note that the makespan is the L_∞ norm of the machine loads, and that the L_1 norm is easily minimizable. The L_p norm of the machine loads, for $1 < p < \infty$, interpolate between these "minmax" and "minsum" criteria. (See, e.g., [10] for an example that motivates the L_2 norm in this context.) A very recent breakthrough of [9] improves upon the $\Theta(p)$-approximation for minimizing the L_p norm of machine loads [8], by presenting a 2-approximation for each $p > 1$, and a $\sqrt{2}$-approximation for $p = 2$. Our algorithm further improves upon [9] by giving better-than-2 approximation algorithms for all p, $1 \leq p < \infty$: e.g., we get approximations of 1.585, $\sqrt{2}$, 1.381, 1.372, 1.382, 1.389, 1.41, 1.436, 1.46, and 1.485 for $p = 1.5$, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5 and 6 respectively.

(iii) **Multicriteria approximations for completion time and multiple L_p norms.** There has been much interest in schedules that are simultaneously near-optimal w.r.t. multiple objectives and in particular, multiple L_p norms [27, 2, 10, 11, 48, 70] in various special cases of unrelated parallel machines. For unrelated parallel machines, it is easy to show instances where, for example, any schedule that is reasonably close to optimal w.r.t. the L_2 norm will be far from optimal for, say, the L_∞ norm; thus, such simultaneous approximations cannot hold. However, we can still ask multi-criteria

questions. Given an arbitrary (finite, but not necessarily of bounded size) set of positive integers $p_1, p_2, \ldots, p_r$, suppose we are given that there exists a schedule in which: (a) for each i, the L_{p_i} norm of the machine loads is at most some given T_i, and (b) the weighted completion-time is at most some given C. We show how to efficiently construct a schedule in which the L_{p_i} norm of the machine loads is at most $3.2 \cdot T_i$ for each i, and the weighted completion-time is at most $3.2 \cdot C$. To our knowledge, this is the first such multi-criteria approximation algorithm with a constant-factor approximation guarantee. We also present several additional results, some of which generalize our application (i) above, and others that improve upon the results of [10, 48].

Our approach in brief. Suppose we are given a fractional assignment $\{x^*_{i,j}\}$ of jobs j to machines i; i.e., $\sum_i x^*_{i,j} = 1$ for all j. Let $t^*_i = \sum_j p_{i,j} x^*_{i,j}$ be the fractional load on machine i. We round the $x_{i,j}$ in iterations by a melding of linear algebra and randomization. Let $X^{(h)}_{i,j}$ denote the random value of $x_{i,j}$ at the end of iteration h. For one, we maintain the invariant that $\mathbf{E}[X^{(h)}_{i,j}] = x^*_{i,j}$ for all i, j and h. Second, we "protect" each machine i almost until the end: the load $\sum_j p_{i,j} X^{(h)}_{i,j}$ on i at the end of iteration h equals its initial value t^*_i with probability 1, until the remaining fractional assignment on i falls into a small set of simple configurations. Informally, these two properties respectively capture some of the utility of independent randomized rounding [108] and those of [84, 117], and play a crucial role in our derandomization techniques. Importantly, while our algorithm is fundamentally based on linear systems, in Lemma 4, we show that it has good behavior w.r.t. a certain family of *quadratic* functions as well. Similarly, the precise details of our rounding help us show better-than-2 approximations for L_p norms of the machine-loads.

Thus, our main algorithm helps improve upon various basic results in scheduling. In particular, different rounding techniques have thus far been applied for diverse objective functions: e.g., the approach of [84, 117] in [9] for general L_p norms, and

independent randomized rounding [108] for weighted completion time in [119] and for the special case of the L_2 norm in [9]. Our algorithm unifies and strengthens all these results. Furthermore, since it works with differing objective functions such as weighted completion-time and L_p norms of machine loads, it is the first approximation algorithm to construct schedules that are good w.r.t. many such objectives simultaneously. We thus expect our approach to be of use in further contexts as well. Our main algorithm is presented in Section 3.2, followed by the applications. In Chapter 4, we present two further applications of our rounding algorithm, namely, social network modeling & broadcast scheduling.

3.2 The Main Rounding Algorithm

We now present our rounding algorithm which takes as input a fractional assignment x^* of jobs to machines, as well as the processing time $p_{i,j}$ of each job j on each machine i, and produces an integral assignment. Let $x^*_{i,j} \in [0,1]$ denote the fraction of job j assigned to machine i in x^*, and note that for all j, $\sum_i x^*_{i,j} = 1$. Initialize $x = x^*$. Our rounding algorithm iteratively modifies x such that x becomes integral in the end. At least one coordinate of x is rounded to zero or one during each iteration; we will throughout *maintain the invariant* "$\forall j,\ \sum_i x_{i,j} = 1$". Once a co-ordinate is rounded to 0 or 1, it is unchanged from then on.

Notation. Let M denote the set of machines and J denote the set of jobs; let $m = |M|$ and $n = |J|$. The (random) values at the *end* of iteration h will be denoted $X^{(h)}_{i,j}$.

Our algorithm will first go through **Phase 1**, followed by **Phase 2** (one of these phases could be empty). We start by saying when we transition from Phase 1 to Phase 2, and then describe a generic iteration in each of these phases. Suppose we are at the *beginning* of some iteration $h+1$ of the algorithm; so, we are currently

looking at the values $X_{i,j}^{(h)}$. Let a job j be called a *floating job* if it is currently assigned fractionally to more than one machine. Let a machine i be called a *floating machine* if it currently has at least one floating job assigned to it. Machine i is called a *singleton machine* if it has **exactly one** floating job assigned to it currently. Let J' and M' denote the current set of floating jobs and **non-singleton** floating machines respectively. Let $n' = |J'|$ and $m' = |M'|$. Define V to be the set of yet-unrounded pairs currently; i.e., $V = \{(i,j) : X_{i,j}^{(h)} \in (0,1)\}$, and let $v = |V|$. We emphasize that all these definitions are w.r.t. the values at the beginning of iteration $(h+1)$. The current iteration (the $(h+1)^{st}$ iteration) is a Phase 1 iteration if $v > m' + n'$; at the first time we observe that $v \leq m' + n'$, we move to Phase 2. So, we initially have some number of iterations at the start of each of which, we have $v > m' + n'$; these constitute Phase 1. Phase 2 starts at the beginning of the first iteration where we have $v \leq m' + n'$. We next describe iteration $(h+1)$, based on which phase it is in.

Case I: *Iteration $(h+1)$ is in Phase 1.* Let J', M', n', m', V and v be as defined above, and recall that $v > m' + n'$. Consider the following linear system: **(E1)** $\forall j \in J'$, $\sum_{i \in M} x_{i,j} = 1$; and **(E2)** $\forall i \in M'$, $\sum_{j \in J'} x_{i,j} \cdot p_{i,j} = \sum_{j \in J'} X_{i,j}^{(h)} \cdot p_{i,j}$. The point $P = (X_{i,j}^{(h)} : i \in M, j \in J')$ is a feasible solution for the variables $\{x_{i,j}\}$, and all the coordinates of P lie in $(0,1)$. Crucially, the number of variables v in the linear system **(E1)**, **(E2)** exceeds the number of constraints $n' + m'$; so, there exists a v-dimensional unit vector r which can be computed in polynomial time such that starting at point P and moving along r or $-r$ does not violate **(E1)** or **(E2)**. Let α and β be the strictly-positive quantities such that starting at point P, α and β are the minimum distances to be traveled along directions r and $-r$ respectively before one of the variables gets rounded to 0 or 1. We now obtain $X^{(h+1)}$ as follows. As mentioned before, all values $X^{(h)}$ which lie in $\{0,1\}$, remain unchanged. For the remaining coordinates, i.e., for the projection $X_V^{(h+1)}$ of $X^{(h+1)}$ along the coordinates V, we do the following: with probability $\frac{\beta}{\alpha+\beta}$, set $X_V^{(h+1)} = X_V^{(h)} + \alpha \cdot r$; with the

complementary probability of $\frac{\alpha}{\alpha+\beta}$, set $X_V^{(h+1)} = X_V^{(h)} - \beta \cdot r$.

This way, it is easy to observe that the new system $X^{(h+1)}$ still satisfies **(E1)** and **(E2)**, has rounded at least one further variable, and also satisfies $\mathbf{E}[X_{i,j}^{(h+1)}] = X_{i,j}^{(h)}$ (for all i, j).

Case II: *Iteration* $(h+1)$ *is in Phase 2.* Let J', M' etc. be defined w.r.t. the values at the start of this (i.e., the $(h+1)^{st}$) iteration. Consider the bipartite graph $G = (M, J', E)$ in which we have an edge (i, j) between job $j \in J'$ and machine $i \in M$ iff $X_{i,j}^{(h)} \in (0, 1)$. We employ the a simpler bipartite dependent-rounding algorithm here. Choose an even cycle $\mathcal{C}$ or a *maximal* path $\mathcal{P}$ in G, and partition the edges in $\mathcal{C}$ or $\mathcal{P}$ into two matchings $\mathcal{M}_1$ and $\mathcal{M}_2$ (it is easy to see that such a partition exists and is unique). Define positive scalars α and β as follows.

$$\begin{aligned}
\alpha &= \min\{\gamma > 0 : \ ((\exists (i,j) \in \mathcal{M}_1 : X_{i,j}^{(h)} + \gamma = 1) \\
&\vee \ (\exists (i,j) \in \mathcal{M}_2 : X_{i,j}^{(h)} - \gamma = 0))\}; \\
\beta &= \min\{\gamma > 0 : \ ((\exists (i,j) \in \mathcal{M}_1 : X_{i,j}^{(h)} - \gamma = 0) \\
&\vee \ (\exists (i,j) \in \mathcal{M}_2 : X_{i,j}^{(h)} + \gamma = 1))\}.
\end{aligned}$$

We execute the following randomized step, which rounds at least one variable to 0 or 1:

> With probability $\beta/(\alpha+\beta)$, set $X_{i,j}^{(h+1)} := X_{i,j}^{(h)} + \alpha$ for all $(i,j) \in \mathcal{M}_1$, and $X_{i,j}^{(h+1)} := X_{i,j}^{(h)} - \alpha$ for all $(i,j) \in \mathcal{M}_2$; with the complementary probability of $\alpha/(\alpha+\beta)$, set $X_{i,j}^{(h+1)} := X_{i,j}^{(h)} - \beta$ for all $(i,j) \in \mathcal{M}_1$, and $X_{i,j}^{(h+1)} := X_{i,j}^{(h)} + \beta$ for all $(i,j) \in \mathcal{M}_2$.

This completes the description of Phase 2, and of our algorithm.[1]

Define machine i to be *protected* during iteration $h+1$ if iteration $h+1$ was

[1]Phase 2 is also based on linear-algebraic and probabilistic ideas as Phase 1 and can be viewed as an "unweighted" version of Phase 1. We elaborate on the connection between these two phases in Section 4.2 of Chapter 4.

in Phase 1, and if i was **not** a singleton machine at the start of iteration $h+1$. If i was then a non-singleton floating machine, then since Phase 1 respects **(E2)**, we will have, for any given value of $X^{(h)}$, that

$$\sum_{j \in J} X_{i,j}^{(h+1)} \cdot p_{i,j} = \sum_{j \in J} X_{i,j}^{(h)} \cdot p_{i,j} \tag{3.1}$$

with probability one. This of course also holds if i had no floating jobs assigned to it at the beginning of iteration $h+1$. Thus, if i is protected in iteration $(h+1)$, the total (fractional) load on it is the same at the beginning and end of this iteration with probability 1.

Our algorithm requires some $t < mn$ iterations. Let X denote the final rounded vector output by our algorithm. We now present the following three lemmas about our algorithm.

Lemma 1 *(i) In any iteration of Phase 2, any floating machine has at most two floating jobs assigned fractionally to it. (ii) Let ϕ and J' denote the fractional assignment and set of floating jobs respectively, at the beginning of Phase 2. For any values of these random variables, we have with probability one that for all $i \in M$, $\sum_{j \in J'} X_{i,j} \in \{\lfloor \sum_{j \in J'} \phi_{i,j} \rfloor, \lceil \sum_{j \in J'} \phi_{i,j} \rceil\}$.*

Proof We start by making some observations about the beginning of the first iteration of Phase 2. Consider the values v, m', n' the beginning of that iteration. At this point, we had $v \leq n' + m'$; also observe that $v \geq 2n'$ and $v \geq 2m'$ since every job $j \in J'$ is fractionally assigned to at least two machines and every machine $i \in M'$ is a non-singleton floating machine. Therefore, we must have $v = 2n' = 2m'$; in particular, we have that every non-singleton floating machine has *exactly two floating jobs fractionally assigned to it.* The remaining machines of interest, the singleton floating machines, have exactly one floating job assigned to them. This proves part (i).

Recall that each iteration of Phase 2 chooses a cycle or a *maximal* path. So, it is easy to see that if i had two fractional jobs j_1 and j_2 assigned fractionally to it at the beginning of iteration $h+1$ in Phase 2, then we have $X_{i,j_1}^{(h+1)} + X_{i,j_2}^{(h+1)} = X_{i,j_1}^{(h)} + X_{i,j_2}^{(h)}$ with probability 1. This equality, combined with part (i), helps us prove part (ii). ■

Lemma 2 *For all i, j, h, v, $\mathbf{E}[X_{i,j}^{(h+1)} \mid (X_{i,j}^{(h)} = v)] = v$. In particular, $\mathbf{E}[X_{i,j}^{(h)}] = x_{i,j}^*$ for all i, j, h.*

Proof Consider $\mathbf{E}[x(h+1) \mid x(h) = a]$. Observe that irrespective of whether case 1 or case 2 of the algorithm occurs after iteration h, the following holds in iteration $h+1$: there exists a unit vector r and two scalers α and β such that $x(h+1) = a + \alpha \cdot r$ with probability $\frac{\beta}{\alpha+\beta}$ and $x(h+1) = a - \beta \cdot r$ with the complementary probability. Thus, we clearly have $\mathbf{E}[x(h+1) \mid x(h) = a] = a$. It is now easy to see using linearity of expectations and induction on h that the lemma holds. ■

Lemma 3 *(i) Let machine i be protected during iteration $h+1$. Then $\forall h' \in \{0, \ldots, h+1\}$, $\sum_{j \in J} X_{i,j}^{(h')} \cdot p_{i,j} = \sum_{j \in J} x_{i,j}^* \cdot p_{i,j}$ with probability 1. (ii) For all i, $\sum_{j \in J} X_{i,j} \cdot p_{i,j} < \sum_{j \in J} x_{i,j}^* \cdot p_{i,j} + \max_{j \in J:\ x_{i,j}^* \in (0,1)} p_{i,j}$ with probability 1.*

Proof Part (i) follows from equation (3.1), and from the fact that if a machine was protected in any one iteration, it is also protected in all previous ones.

For part (ii), if i remained protected throughout the algorithm, then its total load never changes and the lemma holds. Assume i become a singleton machine when it became unprotected. The total load on i when it became unprotected is $\sum_{j \in J} x_{i,j}^* \cdot p_{i,j}$ and irrespective how the floating job on i gets rounded, the additional load on i is strictly less than $\max_{j \in J:\ x_{i,j}^* \in (0,1)} p_{i,j}$. Hence the lemma holds. Finally, assume that i had two floating jobs j_1 and j_2 when it became unprotected (Lemma 1(i) shows that this is the only remaining possibility); let the fractional assignments of j_1

and j_2 on i at this time be ϕ_{i,j_1} and ϕ_{i,j_2} respectively. Let $\phi_{i,j_1} + \phi_{i,j_2} \in (0,1]$. Hence, by Lemma 1(ii), at most one of these jobs is finally assigned to i. So, the additional load on i is strictly less than $\sum_{j \in J} x^*_{i,j} \cdot p_{i,j} + \max_{j \in J:\ x^*_{i,j} \in (0,1)} p_{i,j}$. A similar argument holds when $\phi_{i,j_1} + \phi_{i,j_2} \in (1,2]$. Hence, the lemma holds. ■

3.3 Weighted Completion Time and Makespan

We present a $(\frac{3}{2}, 2)$-bicriteria approximation algorithm for (weighted completion time, makespan) with unrelated parallel machines. Given a pair (C, T), where C is the target value of the weighted completion time and T, the target makespan, our algorithm either proves that no schedule exists which simultaneously satisfies both these bounds, or yields a solution whose cost is at most $(\frac{3C}{2}, 2T)$. Our algorithm builds on the ideas of Skutella [119] and those of Section 3.2; as we will see, the makespan bound needs less work, but managing the weighted completion time simultaneously needs much more care. Let w_j denote the weight of job j. For a given assignment of jobs to machines, the sequencing of the assigned jobs can be done optimally on each machine i by applying Smith's ratio rule [120]: schedule the jobs in the order of non-increasing ratios $\frac{w_j}{p_{i,j}}$. Let this order on machine i be denoted $\prec_i$. Given an assignment-vector x and a machine i, let $\Phi_i(x) = \sum_{(k,j):\ k \prec_i j} w_j x_{i,j} x_{i,k} p_{i,k}$. Note that if x is an *integral* assignment, then $\sum_i \sum_{k:\ k \prec_i j} x_{i,j} x_{i,k} p_{i,k}$ is the amount of time that job j waits before getting scheduled. Thus, for integral assignments x, the total weighted completion time is

$$(\sum_{i,j} w_j p_{i,j} x_{i,j}) + (\sum_i \Phi_i(x)). \tag{3.2}$$

Given a pair (C, T), we write the following Integer Quadratic Program (IQP) motivated by [119]. The $x_{i,j}$ are the usual assignment variables, and z denotes an upper bound on the weighted completion time. The IQP is to minimize z subject to

"$\forall j,\ \sum_i x_{i,j} = 1$", "$\forall i, j,\ x_{i,j} \in \{0,1\}$", and:

$$z \geq (\textstyle\sum_j w_j \sum_i \frac{x_{i,j}(1+x_{i,j})}{2} p_{i,j}) + (\sum_i \Phi_i(x)); \tag{3.3}$$

$$z \geq \textstyle\sum_j w_j \sum_i x_{i,j} p_{i,j}; \tag{3.4}$$

$$\forall i,\ T \geq \textstyle\sum_j p_{i,j} x_{i,j}; \tag{3.5}$$

$$\forall (i,j),\ (p_{i,j} > T) \Rightarrow (x_{i,j} = 0). \tag{3.6}$$

The constraint (3.6) is easily seen to be valid, since we want solutions of makespan at most T. Next, since $u(1+u)/2 = u$ for $u \in \{0,1\}$, (3.2) shows that constraints (3.3) and (3.4) are valid: z denotes an upper bound on the weighted completion time, subject to the makespan being at most T. Crucially, as shown in [119], the quadratic constraint (3.3) is *convex*, and hence the convex-programming relaxation (CPR) of the IQP wherein we set $x_{i,j} \in [0,1]$ for all i, j, is solvable in polynomial time. Technically, we can only solve the relaxation to within an additional error ϵ that is, say, any positive constant. As shown in [119], this is easily dealt with by derandomizing the algorithm. Let ϵ be a suitably small positive constant. We find a (near-)optimal solution to the CPR, with additive error at most ϵ. If this solution has value more than $C+\epsilon$, then we have shown that (C,T) is an infeasible pair. Else, we construct an integral solution by employing our rounding algorithm of Section 3.2 on the fractional assignment x. Assuming that we obtained such a fractional assignment, let us now analyze this algorithm. Let $X^{(h)}$ denote the (random) fractional assignment at the end of iteration h of our rounding algorithm. Our key lemma is:

Lemma 4 *For all i and h, $\mathbf{E}[\Phi_i(X^{(h+1)})] \leq \mathbf{E}[\Phi_i(X^{(h)})]$.*

Proof Fix a machine i and iteration h. Also fix the fractional assignment at the end of iteration h to be some arbitrary $x^{(h)} = \{x_{i,j}^{(h)}\}$. So, our goal is to show, conditional on this fractional assignment, that $\mathbf{E}[\Phi_i(X^{(h+1)})] \leq \Phi_i(x^{(h)})$. We may assume that $\Phi_i(x^{(h)}) > 0$, since $\mathbf{E}[\Phi_i(X^{(h+1)})] = 0$ if $\Phi_i(x^{(h)}) = 0$. We first show

by a perturbation argument that the value $\alpha = \mathbf{E}[\Phi_i(X^{(h+1)})]/\Phi_i(x^{(h)})$ is maximized when all jobs with nonzero weight have the same $\frac{w_j}{p_{i,j}}$ ratio. Partition the jobs into sets $S_1, \ldots, S_k$ such that in each partition, the jobs have the same $\frac{w_j}{p_{i,j}}$ ratio. Let the ratio for set S_g be r_g and let $r_1, \ldots, r_k$ be in non-decreasing order. For each job $j \in S_1$, we set $w'_j = w_j + \lambda p_{i,j}$ where λ has sufficiently small absolute value so that the relative ordering of $r_1, \ldots, r_k$ does not change. This changes the value of α to a new value $\alpha'(\lambda) = \frac{a+b\lambda}{c+d\lambda}$, where a, b, c and d are constants independent of λ, $\alpha = a/b$, and $a, b > 0$. Crucially, since $\alpha'(\lambda)$ is a ratio of two linear functions, its value depends monotonically (either increasing or decreasing) on λ, in the allowed range for λ. Hence, there exists an allowed value for λ such that $\alpha'(\lambda) \geq \alpha$, and either $r'_1 = r_2$ or $r'_1 = 0$. The terms for jobs with zero weight can be removed. We continue this process until all jobs with non-zero weight have the same ratio $\frac{w_j}{p_{i,j}}$. So, we assume w.l.o.g. that all jobs have the same value of this ratio; thus we can rewrite, for some constant $\gamma > 0$,

$$\Phi_i(x^{(h)}) = \gamma \cdot \sum_{\{k,j\}:k \prec_i j} p_{i,j} p_{i,k} x_{i,j}^{(h)} x_{i,k}^{(h)};$$
$$\mathbf{E}[\Phi_i(X^{(h+1)})] = \gamma \cdot \mathbf{E}[\sum_{\{k,j\}:k \prec_i j} p_{i,j} p_{i,k} X_{i,j}^{(h+1)} X_{i,k}^{(h+1)}].$$

(Again, the above expectations are taken conditional on $X^{(h)} = x^{(h)}$.) There are three possibilities for a machine i during iteration $h+1$: **Case I:** *i is protected in iteration $h+1$.* In this case,

$$\begin{aligned}
&\mathbf{E}[\Phi_i(X^{(h+1)})] \\
&= \tfrac{\gamma}{2} \cdot (\mathbf{E}[(\textstyle\sum_j p_{i,j} X_{i,j}^{(h+1)})^2] - \textstyle\sum_j \mathbf{E}[(p_{i,j} X_{i,j}^{(h+1)})^2]) \\
&= \tfrac{\gamma}{2} \cdot ((\textstyle\sum_j p_{i,j} x_{i,j}^{(h)})^2 - \textstyle\sum_j \mathbf{E}[(p_{i,j} X_{i,j}^{(h+1)})^2])
\end{aligned}$$

where the latter equality follows since i is protected in iteration $h+1$. Further, for

any j, the probabilistic rounding ensures that there exists a pair of positive reals (u, v) such that

$$\begin{aligned} \mathbf{E}[(X_{i,j}^{(h+1)})^2] &= \frac{v}{u+v}(x_{i,j}^{(h)}+u)^2 + \frac{u}{u+v}(x_{i,j}^{(h)}-v)^2 \\ &\geq (x_{i,j}^{(h)})^2 \end{aligned}$$

Hence, $\mathbf{E}[\Phi_i(X^{(h+1)})] \leq \Phi_i(x^{(h)})$ in this case. **Case II:** *i is unprotected since it was a singleton machine at the start of iteration $h+1$.* Let j be the single floating job assigned to i. Then, $\Phi_i(X^{(h+1)})$ is a linear function of $X_{i,j}^{(h+1)}$, and so $\mathbf{E}[\Phi_i(X^{(h+1)})] = \Phi_i(x^{(h)})$ by the linearity of expectation. **Case III:** *Iteration $h+1$ is in Phase 2, and i had two floating jobs then.* (Lemma 1(i) shows that this is the only remaining case.) Let j and j' be the floating jobs on i. $\Phi_i(X^{(h+1)})$ has: (i) constant terms, (ii) terms that are linear in $X_{i,j}^{(h+1)}$ or $X_{i,j'}^{(h+1)}$, and (iii) the term $X_{i,j}^{(h+1)} \cdot X_{i,j'}^{(h+1)}$ with a non-negative coefficient. Terms of type (i) and (ii) are handled by the linearity of expectation, just as in Case II. Now consider the term $X_{i,j}^{(h+1)} \cdot X_{i,j'}^{(h+1)}$; we claim that the two factors here are *negatively correlated*. Indeed, in each iteration of Phase 2, there are positive values u, v such that we set $(X_{i,j}^{(h+1)}, X_{i,j'}^{(h+1)})$ to $(x_{i,j}^{(h)}+v, x_{i,j'}^{(h)}-v)$ with probability $u/(u+v)$, and to $(x_{i,j}^{(h)}-u, x_{i,j'}^{(h)}+u)$ with probability $v/(u+v)$. We can verify now that $\mathbf{E}[X_{i,j}^{(h+1)} \cdot X_{i,j'}^{(h+1)}] \leq x_{i,j}^{(h)} \cdot x_{i,j'}^{(h)}$; thus, the type (iii) term is also handled. ■

Lemma 4 leads to our main theorem here.

Theorem 5 *Let C' and T' denote the total weighted completion time and makespan of the integral solution. Then, $E[C'] \leq (3/2) \cdot (C+\epsilon)$ for any desired constant $\epsilon > 0$, and $T' \leq 2T$ with probability 1; this can be derandomized to deterministically yield the pair $(3C/2, 2T)$.*

Proof For simplicity, we ignore the factor of ϵ; in the full version, we will show how it can be dealt with in the same simple manner as in [119]. The fact that $T' \leq 2T$

with probability 1 easily follows by applying Lemma 3(ii) with constraints (3.5) and (3.6). Let us now bound $\mathbf{E}[C']$.

Recall that $X = \{X_{i,j}\}$ denotes the final random integral assignment. Lemma 2 shows that $\mathbf{E}[X_{i,j}] = x^*_{i,j}$. Also, Lemma 4 shows that $\mathbf{E}[\Phi_i(X)] \leq \Phi_i(x^*)$, for all i. These, combined with the linearity of expectation, yields the following:

$$\mathbf{E}[(\sum_j w_j \sum_i p_{i,j} X_{i,j}/2) + (\sum_i \Phi_i(X))] \quad \leq \tag{3.7}$$

$$(\sum_j w_j \sum_i p_{i,j} x_{i,j}/2) + (\sum_i \Phi_i(x)) \quad \leq z \tag{3.8}$$

where the second inequality follows from (3.3). Similarly, we have

$$\mathbf{E}[\sum_j w_j \sum_i X_{i,j} p_{i,j}] = \sum_j w_j \sum_i x_{i,j} p_{i,j} \leq z, \tag{3.9}$$

where the inequality follows from (3.4). As in [119], we get from (3.2) that $\mathbf{E}[C'] = (\sum_{i,j} w_j p_{i,j} \mathbf{E}[X_{i,j}]) + (\sum_i \mathbf{E}[\Phi_i(X)])$ $= \mathbf{E}[(\sum_j w_j \sum_i p_{i,j} X_{i,j}/2) + (\sum_i \Phi_i(X))] + \mathbf{E}[\sum_j w_j \sum_i X_{i,j} p_{i,j}/2$ $z + z/2 \leq C/2$. We can derandomize this algorithm using the method of conditional probabilities. ■

3.4 Minimizing the L_p Norm of Machine Loads

We now consider the problem of scheduling to minimize the L_p norm of the machine-loads, for some given $p > 1$. (The case $p = 1$ is trivial, and the case where $p < 1$ is not well-understood due to non-convexity.) We model this problem using a slightly different convex-programming formulation than Azar & Epstein [9]. Let $\{1, \ldots, n\}$ and $\{1, \ldots, m\}$ denote the set of jobs and machines respectively. Let T be a target value for the L_p norm objective. Any feasible integral assignment with an L_p norm

of at most T satisfies the following integer program.

$$\forall j \in \{1, \ldots, n\} \sum_{i=1}^{m} x_{i,j} \geq 1 \tag{3.10}$$

$$\forall i \in \{1, \ldots, m\} \sum_{j=1}^{n} x_{i,j} \cdot p_{i,j} - t_i \leq 0 \tag{3.11}$$

$$\sum_{i=1}^{m} t_i^p \leq T^p \tag{3.12}$$

$$\sum_{i=1}^{m} \sum_{j=1}^{n} x_{i,j} \cdot p_{i,j}^p \leq T^p \tag{3.13}$$

$$\forall (i,j) \in \{1, \ldots m\} \times \{1, \ldots n\} \; x_{i,j} \in \{0,1\} \tag{3.14}$$

$$\forall (i,j) \in \{(i,j) \mid p_{i,j} > T\} \; x_{i,j} = 0 \tag{3.15}$$

We let $x_{i,j} \geq 0$ for all (i,j) in the above integer program, to obtain a convex program. The feasibility of the convex program can be checked in polynomial time to within an additive error of ϵ (for an arbitrary constant $\epsilon > 0$): the nonlinear constraint (3.12) is not problematic since it defines a convex feasible region [9]. We obtain the minimum feasible value of the L_p norm, T^*, using bisection search in the range $[\min_{i,j}\{p_{i,j}\}, \max_{i,j}\{p_{i,j}\}]$. We ignore the additive error ϵ in the rest of our discussions since our all our randomized guarantees can be obtained deterministically using the method of conditional probabilities in such a way that ϵ is eliminated from the final cost. We also assume that T is set to T^* by a suitable bisection search. We start with two lemmas involving useful calculations.

Lemma 6 *Let $a \in [0,1]$ and $p, \lambda > 0$. Define $N(a,\lambda) = a \cdot (1+\lambda)^p + (1-a)$ and $D(a,\lambda) = (1+a\lambda)^p + a\lambda^p$. Let $\gamma(p) = \max_{(a,\lambda) \in [0,1] \times [0,\infty)} \frac{N(a,\lambda)}{D(a,\lambda)}$. Then, $\gamma(p)$ is at most: (i) 1, if $p \in (1,2]$; (ii) 2^{p-2}, if $p \in (2,\infty)$; and (iii) $O(\frac{2^p}{\sqrt{p}})$ if p sufficiently large. Further, for $p = 2.5, 3, 3.5, 4, 4.5, 5, 5.5$ and 6, $\gamma(p)$ is at most $1.12, 1.29, 1.55, 1.86, 2.34, 3.05, 4.0$ and 5.36 respectively.*

Proof Consider any $p \geq 1$. Let $N'(a,\lambda) = pa \cdot (1+\lambda)^{p-1}$ and $D'(a,\lambda) = pa \cdot (1+a\lambda)^{p-1} + pa\lambda^{p-1}$ be the derivatives of $N(a,\lambda)$ and $D(a,\lambda)$ respectively w.r.t. λ. Observe that $N(a,\lambda) = N(a,0) + \int_0^\lambda N'(a,\lambda)d\lambda$ and $N(a,\lambda) = D(a,0) + \int_0^\lambda D'(a,\lambda)d\lambda$. Hence, $\max_{a,\lambda} \frac{N(a,\lambda)}{D(a,\lambda)} \leq \max\{\frac{N(a,0)}{D(a,0)}, \max_{a,\lambda} \frac{N'(a,\lambda)}{D'(a,\lambda)}\}$. Consider the latter fraction. We have $\max_{a,\lambda} \frac{N'(a,\lambda)}{D'(a,\lambda)} \leq \frac{(1+\lambda)^{p-1}}{1+\lambda^{p-1}}$ which is maximized when $\lambda = 1$. Hence $\max_{a,\lambda} \frac{N(a,\lambda)}{D(a,\lambda)} \leq \max\{1, 2^{p-2}\}$. Thus the lemma holds for the first two cases.

Now suppose p is sufficiently large. Observe that $\frac{N(a,\lambda)}{D(a,\lambda)} \leq \frac{a(1+\lambda)^p+1-a}{1+pa\lambda+a\lambda^p}$. Both the numerator and denominator of the latter fraction are linear functions of a and hence the fraction is maximized when $a \in [0,1]$. When $a = 0$, the fraction evaluates to 1 and the claim holds. Hence, it is enough to show that $\frac{(1+\lambda)^p}{1+p\lambda+\lambda^p} \leq \gamma(p)$. We first note that this is indeed the case when $\lambda \in [0,1]$: if $\lambda \leq \frac{1}{2}$, then the ratio is at most $\left(\frac{3}{2}\right)^p$; else if $\lambda \in [\frac{1}{2}, 1]$, the denominator is at least $\frac{p}{2}$ and the numerator is at most 2^p. Hence the lemma holds. Assume that $\lambda \geq 1$. Let $\lambda = \frac{1+\epsilon}{1-\epsilon}$. The value of the ratio is seen to be at most $\frac{2^p}{p(1-\epsilon)^p+(1+\epsilon)^p}$. The denominator is minimized when $\frac{1+\epsilon}{1-\epsilon} = p^{\frac{1}{p-1}}$; so $\epsilon = \frac{\ln p}{2p} \pm O((\frac{\ln p}{p})^2)$. This implies that $1+\epsilon = 1 + \frac{\ln p}{2p} \pm O((\frac{\ln p}{p})^2)$ and $1-\epsilon = 1 - \frac{\ln p}{2p} \pm O((\frac{\ln p}{p})^2)$. Substituting back these values yields the lemma's claim for large p.

Next, suppose $p \in S = \{2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6\}$. We use numerical techniques to obtain tighter bounds on $\gamma(p)$. For each $p \in S$, it suffices to show the following for $(a,\lambda) \in [0,1] \times [0,\infty)$:

$$\begin{aligned} f(a,\lambda) &\doteq (1+a\lambda)^p \\ &\geq \frac{1}{\gamma(p)} + a \cdot \frac{(1+\lambda)^p - 1 - \gamma(p)\lambda^p}{\gamma(p)} \\ &\doteq g(a,\lambda) \end{aligned} \tag{3.16}$$

For any fixed λ, $f(a,\lambda)$ is a convex function of a while $g(a,\lambda)$ is linear. Assume $\gamma(p) > 1$. Hence, $f(0,\lambda) \geq g(0,\lambda)$ and $f(1,\lambda) \geq g(1,\lambda)$. Hence, if (3.16) is violated, then

the straight line $g(a, \lambda)$ intersects the convex function $f(a, \lambda)$ at two distinct values of $a \in (0, 1)$. In this case, by Lagrange's Mean Value Theorem, there exists an $a \in (0, 1)$ such that $f'(a, \lambda)$ (derivative w.r.t. a), i.e., $p\lambda(1 + a\lambda)^{p-1}$, equals $\frac{(1+\lambda)^p - 1 - \gamma(p)\lambda^p}{\gamma(p)}$. Let $a^*(\lambda)$ be this value which can be obtained by solving this equation for a. Since f is strictly convex as a function of a, $f(a^*(\lambda), \lambda) < g(a^*(\lambda), \lambda)$. The above arguments yield us the following strategy for choosing $\gamma(p)$. We choose $\gamma(p)$ such that one of the following conditions hold for $\lambda \in [0, 15]$.

1. $a^*(\lambda) < 0$. In this case both the intersection points of $g(a, \lambda)$ and $f(a, \lambda)$ are at values $a \leq 0$. Hence, $g(a, \lambda) \leq f(a, \lambda)$ for all values of $a \in [0, 1]$ and our claims hold.

2. $f(a^*(\lambda), \lambda) - g(a^*(\lambda), \lambda) \geq 0$. In this case the functions do not intersect within the ranges of a and λ that are of interest to us.

For the choices of $\gamma(p)$ in this lemma, one of these two conditions occurs and this can be verified numerically by plotting the above functions of λ in the range $\lambda \in [0, 15]$. We restrict ourselves to $\lambda \in [0, 15]$ since the fraction $\frac{N(a,\lambda)}{D(a,\lambda)}$ for the values of $\lambda > 15$ can be easily seen to be within $\gamma(p)$ for the values of p considered here. This completes the proof of the lemma. ■

Lemma 7 *Let a_1, a_2 be variables, each taking values in $[0, 1]$. Let $D \doteq (\lambda_0 + a_1 \cdot \lambda_1 + a_2 \cdot \lambda_2)^p + a_1\lambda_1^p + a_2\lambda_2^p$, where $p > 1$, $\lambda_0 \geq 0$ and $\lambda_1, \lambda_2 > 0$ are fixed constants. Define N as follows:*
if $a_1 + a_2 \leq 1$, then $N = (1 - a_1 - a_2) \cdot \lambda_0^p + a_1 \cdot (\lambda_0 + \lambda_1)^p + a_2 \cdot (\lambda_0 + \lambda_2)^p$; else if $a_1 + a_2 \in (1, 2]$, then $N = (1 - a_2) \cdot (\lambda_0 + \lambda_1)^p + (1 - a_1) \cdot (\lambda_0 + \lambda_2)^p + (a_1 + a_2 - 1) \cdot (\lambda_0 + \lambda_1 + \lambda_2)^p$.
Then, the ratio N/D is maximized when at least one of the variables a_1 and a_2 belongs to $\{0, 1\}$; also, the maximum value of N/D is at most $\gamma(p)$, the value from Lemma 6.

Proof We analyze the various cases. In all the cases below, we assume w.l.o.g. that $\lambda_1 \geq \lambda_2$.

Case 1: $a_1 + a_2 = 1$**.** We have

$$\begin{aligned} \frac{N}{D} &= \frac{a_1(\lambda_0+\lambda_1)^p + (1-a_1)(\lambda_0+\lambda_2)^p}{(\lambda_0+\lambda_2+a_1(\lambda_1-\lambda_2))^p + a_1\lambda_1^p + (1-a_1)\lambda_2^p} \\ &\leq \frac{a_1(\lambda_0+\lambda_2+(\lambda_1-\lambda_2))^p + (1-a_1)(\lambda_0+\lambda_2)^p}{(\lambda_0+\lambda_2+a_1(\lambda_1-\lambda_2))^p + a_1(\lambda_1-\lambda_2)^p} \end{aligned}$$

By scaling both the numerator and denominator of the latter fraction by $(\lambda_0+\lambda_2)^p > 0$, and by letting $\lambda = \lambda_1 - \lambda_2$, the fraction is seen to assume the form as in Lemma 6. Hence the lemma holds.

Case 2: $a_1 + a_2 \geq 1$**.** In this case,

$$\frac{N}{D} = \frac{(a_1+a_2-1)(\lambda_0+\lambda_2+\lambda_1)^p + (1-a_1)(\lambda_0+\lambda_2)^p + (1-a_2)(\lambda_0+\lambda_1)^p}{(\lambda_0+a_1\lambda_1+a_2\lambda_2)^p + a_1\lambda_1^p + a_2\lambda_2^p}$$

Let $\lambda_0 + a_1\lambda_1 + a_2\lambda_2 = \phi$. For any fixed ϕ, both the numerator and the denominator become linear functions of a_1 and a_2 and the ratio becomes a monotonic function of a_1 (since ϕ is fixed, if the ratio monotonically increases with a_1 then it monotonically decreases with a_2, and vice-versa). Hence, we now seek to maximize $\frac{N}{D}$ which is a monotonic function of a_1, subject to the linear constraints that (i) $0 \leq a_1 \leq 1$, (ii) $0 \leq a_2 \leq 1$, (iii) $a_1 + a_2 \geq 1$, and (iv) $\lambda_0 + a_1\lambda_1 + a_2\lambda_2 = \phi$, where ϕ is fixed. Clearly, the maximum occurs when two of these constraints are met with equality. If $a_1 + a_2 = 1$, then **Case 1** occurs and the lemma holds. Otherwise, either a_1 or a_2 is equal to 1. W.l.o.g., let $a_2 = 1$. We have

$$\begin{aligned} \frac{N}{D} &= \frac{a_1(\lambda_0+\lambda_2+\lambda_1)^p + (1-a_1)(\lambda_0+\lambda_2)^p}{(\lambda_0+\lambda_2+a_1\lambda_1)^p + a_1\lambda_1^p + \lambda_2^p} \\ &\leq \frac{a_1(\lambda_0+\lambda_2+\lambda_1)^p + (1-a_1)(\lambda_0+\lambda_2)^p}{(\lambda_0+\lambda_2+a_1\lambda_1)^p + a_1\lambda_1^p} \end{aligned}$$

If we scale both the numerator and denominator of the latter fraction by $(\lambda_0+\lambda_2)^p > 0$, it assumes the same form as in Lemma 6. Hence the lemma holds. Similar arguments apply when $a_1 = 1$.

Case 3: $a_1 + a_2 \leq 1$**.** This case can also be proven using arguments similar to those in case 2. ■

We once again round using our algorithm of Section 3.2, and analyze the rounding now. We let i be some fixed machine, and will adopt the following notation and conventions. X denotes the final rounded assignment, $\{x^*_{i,j}\}$ the fractional solution to the convex program, and $t^*_i = \sum_j p_{i,j} x^*_{i,j}$ denotes the load on i in the fractional solution. Let T_i denote the final (random) load on machine i. Let $U = \{U_{i,j}\}$ denote the random fractional assignment at the beginning of the first iteration in which i became unprotected. W.l.o.g., we assume that there are two distinct jobs j_1 and j_2 which are floating on machine i in assignment U. The cases where i became a singleton or i remains protected always, are handled by setting one or both of the variables $\{U_{i,j_1}, U_{i,j_2}\}$ to zero; we do not consider these cases in the rest of our arguments.

Lemma 8 *Let u denote an arbitrary fractional assignment. Then the following holds.*

Case 1: *If $u_{i,j_1} + u_{i,j_2} \in [0,1]$, then*

$$\begin{aligned}
\Pr[((X_{i,j_1} = 1) \bigwedge (X_{i,j_2} = 1)) \mid U = u] &= 0 \\
\Pr[((X_{i,j_1} = 1) \bigwedge (X_{i,j_2} = 0)) \mid U = u] &= u_{i,j_1} \\
\Pr[((X_{i,j_1} = 0) \bigwedge (X_{i,j_2} = 1)) \mid U = u] &= u_{i,j_2} \\
\Pr[((X_{i,j_1} = 0) \bigwedge (X_{i,j_2} = 0)) \mid U = u] &= 1 - u_{i,j_1} - u_{i,j_2}
\end{aligned}$$

Case 2: *If* $u_{i,j_1} + u_{i,j_2} \in (1,2]$, *then*

$$\begin{aligned}
\Pr[((X_{i,j_1}=1)\bigwedge(X_{i,j_2}=1)) \mid U=u] &= u_{i,j_1}+u_{i,j_2}-1 \\
\Pr[((X_{i,j_1}=1)\bigwedge(X_{i,j_2}=0)) \mid U=u] &= 1-u_{i,j_2} \\
\Pr[((X_{i,j_1}=0)\bigwedge(X_{i,j_2}=1)) \mid U=u] &= 1-u_{i,j_1} \\
\Pr[((X_{i,j_1}=0)\bigwedge(X_{i,j_2}=0)) \mid U=u] &= 0
\end{aligned}$$

Proof If i never became unprotected, then both u_{i,j_1} and u_{i,j_2} are zero; we have case 1 and the lemma holds trivially. If i became an unprotected singleton, then $u_{i,j_2} = 0$. Again, case 1 occurs and the above lemma can be easily seen to hold due to Lemma 2. Assume i become unprotected with both j_1 and j_2 fractionally assigned to it (i.e., $u_{i,j_1}, u_{i,j_2} \in (0,1)$). We now analyze case 1. Since $u_{i,j_1} + u_{i,j_2} \in [0,1]$, it follows from Lemma 1 that $\Pr[((X_{i,j_1}=1)\bigwedge(X_{i,j_2}=1)) \mid U=u] = 0$. This implies that $\Pr[((X_{i,j_1}=1)\bigwedge(X_{i,j_2}=0)) \mid U=u] = \Pr[(X_{i,j_1}=1) \mid U=u] = u_{i,j_1}$. The last equality follows from Lemma 2. By an identical argument, $\Pr[((X_{i,j_1}=0)\bigwedge(X_{i,j_2}=1)) \mid U=u] = u_{i,j_2}$. Finally, $\Pr[((X_{i,j_1}=0)\bigwedge(X_{i,j_2}=0)) \mid U=u]$ is the remaining value which is $1-u_{i,j_1}-u_{i,j_2}$. We note that the above arguments hold because the events considered above are mutually exclusive and exhaustive. Case 2 can be proved using very similar arguments; this concludes the proof of the lemma. ■

Theorem 9 *Given a fixed norm* $p > 1$ *and a fractional assignment whose fractional* L_p *norm is* T, *our algorithm produces an integral assignment whose value* C_p *satisfies* $\mathbf{E}[C_p] \leq \rho(p) \cdot T$. *Our algorithm can be derandomized in polynomial time to guarantee that* $C_p \leq \rho(p) \cdot T$. *The approximation factor* $\rho(p)$ *is at most the following: (i)* $2^{\frac{1}{p}}$, *for* $p \in (1,2]$; *(ii)* $2^{1-1/p}$, *for* $p \in [2,\infty)$; *and (iii)* $2 - \Theta(\log p/p)$ *for large* p. *Further, for any fixed value of* $p > 2$ *it is possible to achieve a better factor* $\rho(p)$ *using numerical*

techniques. In particular, the following table illustrates the achievable values of $\rho(p)$ for the corresponding values of p.

p	2.5	3	3.5	4
$\rho(p)$	1.381	1.372	1.382	1.389

p	4.5	5	5.5	6
$\rho(p)$	1.410	1.436	1.460	1.485

Proof Let $A(i) = \{j : U_{i,j} = 1\}$ and let $R_i = \sum_{j \in A(i)} p_{i,j}$ be the rounded load on i. By definition of a protected machine, $R_i + U_{i,j_1} \cdot p_{i,j_1} + U_{i,j_2} \cdot p_{i,j_2} = t_i^*$. By Lemma 8, we have $\mathbf{E}[T_i^p \mid U = u]$ equals the following:

$$(1 - u_{i,j_1} - u_{i,j_2}) \cdot R_i^p + u_{i,j_1} \cdot (R_i + p_{i,j_1})^p + u_{i,j_2} \cdot (R_i + p_{i,j_2})^p$$

if $u_{i,j_1} + u_{i,j_2} \in [0,1]$; and $(1 - u_{i,j_2}) \cdot (R_i + p_{i,j_1})^p + (1 - u_{i,j_1}) \cdot (R_i + p_{i,j_2})^p + (u_{i,j_1} + u_{i,j_2} - 1) \cdot (R_i + p_{i,j_1} + p_{i,j_2})^p$ if $u_{i,j_1} + u_{i,j_2} \in (1,2]$. Let $\mu(x,i) = \sum_j x_{i,j} p_{i,j}^p$ for any assignment-vector x. Irrespective of the value of $u_{i,j_1} + u_{i,j_2}$, we have

$$\begin{aligned} \frac{\mathbf{E}[T_i^p \mid U = u]}{t_i^{*p} + \mathbf{E}[\mu(X,i) \mid U = u]} &= \frac{\mathbf{E}[T_i^p \mid U = u]}{t_i^{*p} + \sum_j u_{i,j} p_{i,j}^p} \leq \\ \frac{\mathbf{E}[T_i^p \mid U = u]}{t_i^{*p} + u_{i,j_1} p_{i,j_1}^p + u_{i,j_2} p_{i,j_2}^p} &\leq \gamma(p) \end{aligned}$$

The last inequality follows from Lemma 7. By rearranging this expression after unconditioning on the value of U and by the linearity of expectation, we get

$$\begin{aligned} \mathbf{E}[T_i^p] &\leq \gamma(p)(t_i^{*p} + \mathbf{E}[\mu(X,i)]) \\ &\leq \gamma(p)(t_i^{*p} + \sum_j x_{i,j}^* p_{i,j}^p) \text{ (by Lemma 2)} \end{aligned}$$

So, $\sum_i \mathbf{E}[T_i^p] \leq 2\gamma(p) \cdot T^p$, by (3.12) and (3.13). The claims for $\rho(p)$ follow by noting that $\rho(p) \leq (2\gamma(p))^{\frac{1}{p}}$ and substituting $\gamma(p)$ from Lemma 6, and by the fact that for any non-negative random variable Z, $\mathbf{E}[Z] \leq (\mathbf{E}[Z^p])^{1/p}$. ■

3.5 Multi-criteria optimization for multiple L_p norms and weighted completion time

We now present our multicriteria optimization results for a given collection of L_p norms and weighted completion time. Let S be a set of integer norms and let $T(p)$ be a target value for each $p \in S$. Let W^* be a target total weighted completion time. Let x^* be a fractional assignment such that for each $p \in S$, its fractional L_p norm is at most $T(p)$ and its weighted completion time $W \leq \frac{3}{2}W^*$. Lemma 10 states that our algorithm of Section 3.2 yields a 2.56 multi-criteria optimization for any given set of integer norms, a 3.2 multi-criteria optimization for any given set of integer norms and the weighted completion time, and generalizes Theorem 5. The proof of Lemma 10 uses convex programming techniques to either show that no such feasible assignment exists or finds a feasible fractional assignment, if one exists.

Lemma 10 *Suppose we are given a set of integer norms S, a target L_p norm $T(p)$ for each $p \in S$, and a target W^* for the weighted completion time. Further, suppose we are given a fractional assignment x^* such that $W(x^*) \leq \frac{3W^*}{2}$ and such that the L_p norm of x^* is at most $T(p)$ for each $p \in S$. Then, our rounding algorithm of Section 3.2 can be derandomized in polynomial time such that any one of the following hold:*

1. *For every $p \in S$, the rounded norm $C(p) \leq 2.56 \cdot T(p)$;*

2. *The rounded completion time $W(X) \leq 3.2 \cdot W^*$ and for every $p \in S$, the rounded norm $C(p) \leq 3.2 \cdot T(p)$;*

3. *For any $\epsilon > 0$, $W(X) \leq \frac{3}{2} \cdot (1+\epsilon)W^*$ and for every $p \in S$, $C(p) \leq 2(e+\frac{2}{\epsilon}) \cdot T(p)$, where e is the base of natural logarithms;*

4. *There exists a constant K such that if $S = \{p\}$, then for any $\epsilon > 0$ and any $p \geq \frac{K}{\epsilon^2}$, $W(X) \leq \frac{3}{2}(1+\epsilon)$ and $C(p) \leq 2 \cdot T(p)$;*

Proof We show how to obtain each of the four guarantees claimed in the Lemma.

Guarantee 1: We first show a 2.56 multi-criteria approximation ratio for a given

collection of integer norms. Specifically, given a collection of integer norms S and a target L_p norm $T(p)$ for each $p \in S$, we either prove that no assignment exists which simultaneously satisfies all these targets or obtain an integral assignment where the final L_p norm for any $p \in S$ is at most $2.56T(p)$. We first formulate the multi-criteria convex program as follows:

$$\begin{aligned}
\sum_{i=1}^{m} x_{i,j} &= 1 && \forall j \in \{1, \ldots, n\} \\
\sum_{j=1}^{n} x_{i,j} \cdot p_{i,j} - t_i &\leq 0 && \forall i \in \{1, \ldots, m\} \\
\sum_{i=1}^{m} t_i^p &\leq T(p)^p && \forall p \in S \\
\sum_{i=1}^{m} \sum_{j=1}^{n} x_{i,j} \cdot p_{i,j}^p &\leq T(p)^p && \forall p \in S \\
x_{i,j} &\geq 0 && \forall (i,j) \\
x_{i,j} &= 0 && \forall (i,j) \in \{(i,j) \mid \exists p \in S \text{ s.t. } p_{i,j} > T(p)\}
\end{aligned}$$

If the above convex program is infeasible, then we declare that no valid assignment exists which respects the targets. Clearly, this is indeed the case. Assume that the convex program is feasible and x^* is the feasible fractional assignment. We will describe a derandomization of the algorithm in the section 3.2, in order to get the guarantee for all $p \in S$. We first recall a few definitions and define new ones. Let $X^{(h)}$ denotes the (fractional) assignment vector after iteration h in our derandomized rounding algorithm; $X^{(0)} \doteq x^*$; let t_i^* denote the fractional load imposed by assignment x^* on machine i. We let x denote an arbitrary assignment vector. For a fixed machine i, let $\mu_p(x, i) = \sum_j x_{i,j} p_{i,j}^p$. Let X denote the final integral assignment. Let T_i denote the final load on machine i. Let R_i, j_1, and j_2, denote the rounded load and the two floating jobs assigned to i respectively, immediately after i becomes unprotected. Define $\phi_p(x, i)$ as follows:

if $x_{i,j_1} + x_{i,j_2} \in [0,1]$, then

$$\phi_p(x,i) = (1 - x_{i,j_1} - x_{i,j_2}) \cdot R_i^p + x_{i,j_1} \cdot (R_i + p_{i,j_1})^p + x_{i,j_2} \cdot (R_i + p_{i,j_2})^p$$

else if $x_{i,j_1} + x_{i,j_2} \in (1,2]$, then

$$\phi_p(x,i) = (1-x_{i,j_2})\cdot(R_i+p_{i,j_1})^p+(1-x_{i,j_1})\cdot(R_i+p_{i,j_2})^p+(x_{i,j_1}+x_{i,j_2}-1)\cdot(R_i+p_{i,j_1}+p_{i,j_2})^p$$

The above definition is motivated by the fact that $\phi(X,i) = T_i^p$. We also define the quantity $\psi_p(x,i)$ as follows: if $p = 1$, then $\psi_p(x,i) = \phi_p(x,i)$; else if $p > 1$, then $\psi_p(x,i) = \frac{\phi_p(x,i)}{\gamma(p)} - t_i^{*p}$. It follows from Lemma 7 that for all $p > 1$ and $\forall i$, $\phi_p(x,i) \leq \gamma(p)(t_i^{*p} + \mu_p(x,i))$, and therefore we have:

$$\psi_p(x,i) \leq \mu_p(x,i) \tag{3.17}$$

Let $M_1^{(h)}$ and $M_2^{(h)}$ denote the set of protected and unprotected machines respectively immediately after iteration h. Let $Q_p(X^{(h)}) = \sum_{i \in M_1^{(h)}} \mu_p(X^{(h)},i) + \sum_{i \in M_2} \psi_p(X^{(h)},i)$. Let $Q(x) = \sum_{p \in S} \frac{Q_p(x)}{f(p) \cdot T(p)^p}$, where the constants $f(p)$ are chosen such that $\sum_{p \in S} \frac{1}{f(p)} \leq 1$. This implies that $Q(X^{(0)}) \leq 1$.

We are now ready to describe the derandomized version of our rounding algorithm. At iteration h, as in the randomized version, we have two choices of assignment vectors x_1 and x_2 and two scalars $\alpha_1, \alpha_2 \geq 0$ and $\alpha_1 + \alpha_2 = 1$ such that $\alpha_1 x_1 + \alpha_2 x_2 = X^{(h)}$. We choose $X^{(h+1)} \in \{x_1, x_2\}$ such that $Q(X^{(h+1)}) \leq Q(X^{(h)})$. This is always possible because $Q(x)$ is a linear function of the components in x - since we also have $Q(x) = \alpha_1 Q(x_1) + \alpha_2 Q(x_2)$, the minimum of these choices will not increase the value of Q; Next, if a machine i becomes unprotected during some iteration h, then for all $p \in S$, we replace $\mu_p(X^{(h+1)},i)$ by $\psi_p(X^{(h+1)},i)$ in the expression for Q. It follows from Equation (3.17) that this replacement does not increase the

value of Q.

Since $Q(X^{(h)})$ is a non-increasing function of h, $Q(X) \leq Q(X^{(0)}) \leq 1$. Hence, it follows that for each $p \in S$, $Q_p(X) \leq f(p)T(p)^p$. We now analyze the final L_p norms for each p. If $p = 1$, the final cost $C(1) = \sum_i \phi_1(X, i) = Q_1(X) \leq f(1)T(1)$. We now analyze the value $C(p)$ for norms $p > 1$. We first note that all machines become unprotected by the time when the algorithm terminates. Hence,

$$\begin{aligned} C(p)^p &= \sum_i \phi_p(X, i) \\ &= \sum_i \gamma(p)(\psi_p(X, i) + t_i^{*p}) \\ &= \gamma(p)(\sum_i t_i^{*p} + Q_p(X)) \\ &\leq \gamma(p)(T(p)^p + f(p)T(p)^p) \\ &\leq \gamma(p)(f(p) + 1)T(p)^p \end{aligned}$$

Hence, $C_p \leq (\gamma(p)(1+f(p)))^{\frac{1}{p}}T(p)$. We are now left to show the choice of values $f(p)$ such that $f(1) = 2.56$ and for all integer $p > 1$, $(\gamma(p)(1 + f(p)))^{\frac{1}{p}} \leq 2.56$ and $\sum_{p=1}^{\infty} \frac{1}{f(p)} \leq 1$, where the summation is over the set of positive integers. Let $k = 1.28$. We choose $f(p)$ as follows: $f(1) = 2k$; for $p \in \{2, 3, 4, 5, 6\}$, $f(p) = \frac{(2k)^p}{\gamma(p)} - 1$; for $p \geq 7$, $f(p) = 4k^p - 1$. By substituting the minimum achievable value $\gamma(p)$ for each p from Lemma 6, we have $(\gamma(p)(1 + f(p)))^{\frac{1}{p}} \leq 2.56$ for every integer p. Next, observe that $\sum_{p=7}^{\infty} \frac{1}{f(p)} \leq \int_6^{\infty} \frac{dr}{4k^r - 1} = \frac{1}{\log k} \cdot \log \frac{4k^6}{4k^6 - 1}$. By substituting the value $k = 1.28$, it follows that $\sum_{p=1}^{\infty} \frac{1}{f(p)} \leq 1$.

Guarantees 2, 3, and 4: We now consider the problem of simultaneously minimizing the total weighted completion time and L_p norms for a given collection of norms S. As in the analysis of guarantee 1, we assume that we are given a set of integer norms S and a target $T(p)$ for each norm $p \in S$; further, we are also given a target value W^* for the weighted completion time. The basic template for combining

completion times with multiple norms is the same as before. We first add the constraints 3.3 and 3.4 from Section 3.3, which correspond to the weighted completion time into the multi-criteria convex program. If this new convex program is infeasible (checkable in polynomial time), we can safely declare that there is no valid assignment which respects all targets. Assume that there is a feasible fractional assignment x^*. We note that the fractional weighted completion time $W(x^*) \leq \frac{3}{2} \cdot W^*$.

Recall the definitions of $Q_p()$ from the proof of guarantee 1 above. Also recall that the total weighted completion time objective for any assignment x is defined as follows: $W(x) = \sum_i w_{i,j} x_{i,j} (p_{i,j} + \sum_{j' \prec_i j} x_{i,j'} p_{i,j'})$. We now redefine our combined objective $Q(x)$ as follows: $Q(x) = \frac{2W(x)}{3gW^*} + \sum_{p \in S} \frac{Q_p(x)}{f(p) \cdot T(p)^p}$, where $\frac{1}{g} + \sum_{p \in S} \frac{1}{f(p)} \leq 1$. Since $X^{(0)} = x^*$, and x^* is a feasible assignment, $Q(X^{(0)}) \leq 1$. We now follow the same derandomization strategy as in guarantee 1 (i.e., choose from two possible choices for $X^{(h+1)}$ such that $Q(X^{(h+1)}) \leq Q(X^{(h)})$). Crucially, we remark that this is possible since, as shown in Lemma 4, at every iteration h, the two choices for $X^{(h+1)}$ namely x_1 and x_2, and the scalars $\alpha_1, \alpha_2 \geq 0$ and $\alpha_1 + \alpha_2 = 1$ are such that $\alpha_1 \cdot x_1 + \alpha_2 \cdot x_2 = X^{(h)}$ and $\alpha_1 W(x_1) + \alpha_2 W(x_2) \leq W(X^{(h)})$; this implies that at every iteration of the derandomized algorithm, there exists a choice of $X^{(h+1)}$ such that $Q(X^{(h+1)}) \leq Q(X^{(h)}) \leq 1$. Hence, $W(X) \leq \frac{3g}{2} W^*$. We are now left to show the choice of coefficients $f(p)$ and g such that the tradeoffs claimed in the lemma can be achieved. We show this below for each of the three claims.

1. We fix $k = 1.6$ and let $g = \frac{4k}{3}$. All the values of $f(p)$ remain the same function of k as in the proof guarantee 1: i.e., $f(1) = 2k$, $\forall p \in \{2, \ldots, 6\}$, $f(p) = \frac{(2k)^p}{\gamma(p)} - 1$, and for $p \geq 7$, $f(p) = 4k^p - 1$. It now follows from the arguments for guarantee 1 that $\frac{1}{g} + \sum_{p \in S} \frac{1}{f(p)} \leq 1$.

2. Fix $k = e + \frac{2}{\epsilon}$ and $g = 1 + \epsilon$. We let $f(1) = 2k$ and for $p \geq 2$, we let $f(p) = 4k^p - 1$ and $\gamma(p) = 2^{p-2}$ (from Lemma 7). This yields an approximation factor of $\frac{3(1+\epsilon)}{2}$ for the completion time and a factor of $2(e + \frac{2}{\epsilon})$ for each norm $p \in S$. We now have,

$\frac{1}{g} + \sum_{p \in S} \frac{1}{f(p)} \leq \frac{1}{1+\epsilon} + \sum_{p=1}^{\infty} \frac{1}{f(p)} \leq 1 - \frac{\epsilon}{2} + \frac{\epsilon}{4} + \frac{1}{\log k} \cdot \log(\frac{4k}{4k-1}) \leq 1 - \frac{\epsilon}{2} + \frac{\epsilon}{4} + \frac{1}{4k-1} \leq 1 - \frac{\epsilon}{2} + \frac{\epsilon}{4} + \frac{\epsilon}{8} \leq 1.$

3. Fix $g = (1+\epsilon)$. Let $f(p) = 1 + \frac{1}{\epsilon}$. We let $\gamma(p) = \Theta(\frac{2^p}{\sqrt{p}})$ from Lemma 7. Hence for all $p = \Omega(\frac{1}{\epsilon^2})$, $\gamma(p) \leq \frac{2^p}{\sqrt{p}} \leq \epsilon 2^{p-2}$. Hence, $W(X) \leq \frac{3(1+\epsilon)}{2} \cdot W^*$ and $C_p \leq (\gamma(p)(f(p)+1))^{\frac{1}{p}} T(p) \leq 2T(p)$. Further, $\frac{1}{g} + \frac{1}{f(p)} = 1$, which concludes the proof of the lemma. ■

Chapter 4

Social Network Modeling and Broadcast Scheduling

4.1 Introduction

Various combinatorial optimization problems include hard *capacity constraints*: e.g., a broadcast server may be able to broadcast at most one topic per time step. In Chapter 3, we developed our dependent rounding approach to accommodate such constraints in the context of assignment on unrelated parallel machines. In this chapter, we present two further applications of dependent rounding to social network modeling, and broadcast scheduling, both of which have been of wide interest to networking and algorithms researchers. We start by recalling the dependent rounding scheme in Section 4.2, and describe its use in social network modeling and broadcast scheduling in sections 4.3 and 4.4 respectively.

4.2 Dependent bipartite rounding

Suppose we are given a bipartite graph (A, B, E) with bipartition (A, B). We are also given a value $x_{i,j} \in [0, 1]$ for each edge $(i, j) \in E$. We are interested in a randomized

polynomial-time scheme that rounds each $x_{i,j}$ to a random variable $X_{i,j} \in \{0,1\}$, in such a way that the following properties hold.

(P1): Marginal distribution. For every edge (i,j), $\Pr[X_{i,j} = 1] = x_{i,j}$.

(P2): Degree-preservation. Consider any vertex $i \in A \cup B$. Define its fractional degree d_i to be $\sum_{j:(i,j)\in E} x_{i,j}$, and integral degree D_i to be the random variable $\sum_{j:(i,j)\in E} X_{i,j}$. Then, we have $D_i \in \{\lfloor d_i \rfloor, \lceil d_i \rceil\}$. Note in particular that if d_i is an integer, then $D_i = d_i$ with probability 1; this will often model the cardinality constraints in our applications.

(P3): Negative correlation. If $f = (i,j)$ is an edge, let X_f denote $X_{i,j}$. For any vertex i and any subset S of the edges incident on i:

$$\forall b \in \{0,1\}, \ \Pr[\bigwedge_{f\in S}(X_f = b)] \leq \prod_{f\in S} \Pr[X_f = b]. \tag{4.1}$$

We observe that the second phase of our dependent rounding algorithm presented in Section 3.2 of Chapter 3, guarantees precisely the above properties. We showed in Section 3.2 of Chapter 3 that the dependent rounding scheme satisfies **(P1)** and **(P2)**. Here, for the sake of completeness, we recall this scheme and prove that it satisfies **(P1)**, **(P2)**, and **(P3)**. Property **(P3)** is motivated by the fact that if we aggregate a large collection of independent *or* negatively correlated random variables, then the aggregate random variable is sharply concentrated around its mean [98]. Sharp concentration properties such as these are often very valuable in many load-balancing applications. While properties **(P1)** and **(P2)** suffice for the applications discussed in this Chapter, we demonstrate a low-congestion routing application for optical networks which utilizes property **(P3)** in Gandhi *et al.* [45].

We now present our rounding scheme. Suppose we are given a bipartite graph (A, B, E) with bipartition (A, B) and a value $x_{i,j} \in [0,1]$ for each edge $(i,j) \in E$. Initialize $y_{i,j} = x_{i,j}$ for each $(i,j) \in E$. We will probabilistically modify the $y_{i,j}$ in

several (at most $|E|$) iterations such that $y_{i,j} \in \{0,1\}$ at the end (at which point we will set $X_{i,j} = y_{i,j}$ for all $(i,j) \in E$). Our iterations will satisfy the following two invariants:

(I1) For all $(i,j) \in E$, $y_{i,j} \in [0,1]$.

(I2) Call $(i,j) \in E$ *rounded* if $y_{i,j} \in \{0,1\}$, and *floating* if $y_{i,j} \in (0,1)$. Once an edge (i,j) gets rounded, $y_{i,j}$ never changes.

An iteration proceeds as follows. Let $F \subseteq E$ be the current set of floating edges. If $F = \emptyset$, we are done. Otherwise, find in $O(|A| + |B|)$ steps via a depth-first-search (DFS), a simple cycle or *maximal* path P in the subgraph (A, B, F), and partition the edge-set of P into two matchings M_1 and M_2. The cycle/maximal path can actually be found in $O(|A| + |B|)$ time since the first back edge we encounter yields a cycle in the DFS.

Define

$$\begin{aligned}
\alpha &= \min\{\gamma > 0 : \ ((\exists (i,j) \in M_1 : y_{i,j} + \gamma = 1) \bigvee (\exists (i,j) \in M_2 : y_{i,j} - \gamma = 0))\}; \\
\beta &= \min\{\gamma > 0 : \ ((\exists (i,j) \in M_1 : y_{i,j} - \gamma = 0) \bigvee (\exists (i,j) \in M_2 : y_{i,j} + \gamma = 1))\}.
\end{aligned}$$

Since the edges in $M_1 \cup M_2$ are currently floating, it is easy to see that the positive reals α and β exist. Now, independent of all random choices made so far, we execute the following randomized step:

> With probability $\beta/(\alpha+\beta)$, set $y_{i,j} := y_{i,j} + \alpha$ for all $(i,j) \in M_1$, and $y_{i,j} := y_{i,j} - \alpha$ for all $(i,j) \in M_2$; with the complementary probability of $\alpha/(\alpha+\beta)$, set $y_{i,j} := y_{i,j} - \beta$ for all $(i,j) \in M_1$, and $y_{i,j} := y_{i,j} + \beta$ for all $(i,j) \in M_2$.

This completes the description of an iteration. A simple check shows that the invariants (I1) and (I2) are maintained, and that at least one floating edge gets rounded in

every iteration.

We now analyze the above randomized algorithm. First of all, since every iteration rounds at least one floating edge, we see from (I2) that we need at most $|E|$ iterations. So,

$$\text{the total running time is } O((|A| + |B|) \cdot |E|). \tag{4.2}$$

Discussion: We now discuss the connections between the bipartite rounding scheme and the linear-algebraic ideas behind Phase 1 of our dependent rounding scheme in Section 3.2 of Chapter 3. Suppose, we are currently at the beginning of some iteration i; let us remove all the rounded edges from the system, since their values do not change in future iterations. Let $y_{i,j}$ denote the current value for a floating edge $\{i, j\}$; further, define η_u to be the current floating degree for any vertex u: this is the sum of the values of all the currently floating edges that are incident on u. We say a vertex is singleton if it has exactly one floating edge incident on it; a vertex is non-singleton if it has two or more floating edges incident on it. Let $A_{ns} \subseteq A$ and $B_{ns} \subseteq B$ be the subsets of non-singleton vertices in A and B respectively. Consider the following linear system. We have,

$$\forall\ i \in A_{ns},\ \sum_{j \in B} y_{i,j} = \eta_i$$
$$\forall\ j \in B_{ns},\ \sum_{i \in A} y_{i,j} = \eta_j$$

Let Γ be the number of floating edges, and hence, the number of variables in the above linear system. Since each vertex in A_{ns} and B_{ns} is incident on at least two floating edges, we have $\Gamma \geq 2|A_{ns}|$, and $\Gamma \geq 2|B_{ns}|$; hence, $\Gamma \geq |A_{ns}| + |B_{ns}|$. We have two cases. First, assume $\Gamma > |A_{ns}| + |B_{ns}|$. In this case, the number of variables in the linear system is strictly greater than the number of equations, and hence the linear system is under-determined. In the second case, assume $\Gamma = |A_{ns}| + |B_{ns}|$; this is possible only if $\Gamma = 2|A_{ns}|$, and $\Gamma = 2|B_{ns}|$. In this case, the sub-graph induced by

the set of floating edges has to be a collection of disjoint even-cycles, and there can be no singleton vertices. Consider any even cycle of floating edges: a moment's reflection shows that in the absence of singleton vertices, the fractional degree of any vertex in this even cycle is uniquely determined by the fractional degrees of the rest of the vertices in the cycle. Generalizing this idea further, it is easy to see that the rank of the constraint matrix for the above linear system is strictly less than $|A_{ns}| + |B_{ns}| = \Gamma$. Hence, the linear system is under-determined in the second case as well. Thus, it is always possible to perturb the values of the currently floating edges such that at least one of them is rounded, and non-singleton machines experience no change in their fractional degrees. This is the essence of our bipartite rounding scheme. The preceding argument shows how property **(P2)** can be guaranteed; the specific set of edges we choose during each iteration, and the probabilistic choices we make yield the marginal distribution and negative correlation properties.

Properties: We now formally prove that properties **(P1)**, **(P2)**, and **(P3)** hold.

Lemma 11 *Property **(P1)** holds, i.e., For every edge* (i, j), $\Pr[X_{i,j} = 1] = x_{i,j}$.

Proof Fix an edge $(p, q) \in E$; let $Y_{p,q,k}$ be the random variable denoting the value of $y_{p,q}$ at the beginning of iteration k. We will show that

$$\forall k \geq 1, \mathbf{E}[Y_{p,q,k+1}] = \mathbf{E}[Y_{p,q,k}] \tag{4.3}$$

We will then have, $\Pr[X_{p,q} = 1] = \mathbf{E}[Y_{p,q,|E|+1}] = \mathbf{E}[Y_{p,q,1}] = x_{p,q}$ and (P1) will hold. We now prove equation (4.3) for a fixed k.

One of the following two events could occur for the edge (p, q) in iteration k.

Event A: The edge (p, q) is not part of the cycle or maximal path and its value is not modified. Hence we have $\mathbf{E}[Y_{p,q,k+1} | (Y_{p,q,k} = v) \wedge A] = v$.

Event B: The edge (p, q) is part of the cycle or maximal path during iteration k. In this case, w.l.o.g., let the edge (p, q) be part of the matching M_1. Then, there exist (α, β) such that $\alpha+\beta > 0$ and such that the edge value gets modified probabilistically as:

$$Y_{p,q,k+1} = \begin{cases} Y_{p,q,k} + \alpha & \text{with probability } \beta/(\alpha+\beta) \\ Y_{p,q,k} - \beta & \text{with probability } \alpha/(\alpha+\beta) \end{cases}$$

Let S be the set of all values of (α, β). We say that the event $B(\alpha_1, \beta_1)$ occurred if event B occurs and $(\alpha, \beta) = (\alpha_1, \beta_1)$ for a fixed $(\alpha_1, \beta_1) \in S$. We have

$$\mathbf{E}[Y_{p,q,k+1} | (Y_{p,q,k} = v) \wedge B(\alpha_1, \beta_1)] \;=\; (v + \alpha_1)\left(\frac{\beta_1}{\alpha_1+\beta_1}\right) + (v - \beta_1)\left(\frac{\alpha_1}{\alpha_1+\beta_1}\right) = v$$

Since the above equation holds for all values of (α, β), it also holds unconditionally. Thus, $\mathbf{E}[Y_{p,q,k+1} | (Y_{p,q,k} = v) \wedge B] = v$. Hence,

$$\begin{aligned} \mathbf{E}[Y_{p,q,k+1} | (Y_{p,q,k} = v)] &= \mathbf{E}[Y_{p,q,k+1} | (Y_{p,q,k} = v) \wedge B] \cdot \Pr[B] + \\ &\quad \mathbf{E}[Y_{p,q,k+1} | (Y_{p,q,k} = v) \wedge A] \cdot \Pr[A] \\ &= v(\Pr[A] + \Pr[B]) = v \end{aligned}$$

Let V be the set of all possible values of $Y_{p,q,k}$.

$$\begin{aligned} \mathbf{E}[Y_{p,q,k+1}] &= \sum_{v \in V} \mathbf{E}[Y_{p,q,k+1} | Y_{p,q,k} = v] \cdot \Pr[Y_{p,q,k} = v] \\ &= (\sum_{v \in V} v \cdot \Pr[Y_{p,q,k} = v]) = \mathbf{E}[Y_{p,q,k}] \end{aligned}$$

This completes our proof for Property (P1). ■

Lemma 12 *Property **(P2)** holds.*

Proof Fix any vertex i, with fractional degree d_i. If i has at most one floating edge incident on it at the beginning of our dependent rounding, it is easy to verify

that (P2) holds; so suppose i initially had at least two floating edges incident on it. We claim that as long as i has at least two floating edges incident on it, the value $D_i^{(y)} \doteq \sum_{j:(i,j)\in E} y_{i,j}$ remains at its initial value of d_i. To see this, first note that if i is not in the cycle/maximal path P chosen in an iteration, then $D_i^{(y)}$ is not altered in that iteration. Next, consider an iteration in which i had at least two floating edges incident on it, and in which i was in the cycle/path P. Then, i must have degree two in P, and so, it must have one edge in M_1 and one in M_2. Then, since edges $(i,j) \in M_1$ have their $y_{i,j}$ value increased/decreased by the same amount as edges in M_2 have their $y_{\cdot,\cdot}$ value decreased/increased, we see that $D_i^{(y)}$ does not change in this iteration. Now consider the last iteration at the beginning of which i had at least two floating edges incident on it. At the end of this iteration, we will have $D_i^{(y)} = d_i$, and i will have at most one floating edge incident on it. It is now easy to see that (P2) holds. ■

Lemma 13 *Property **(P3)** holds.*

Proof Fix a vertex i, and a subset S of edges incident on i, as in (4.1). Let $b = 1$ (the proof for the case where $b = 0$ is identical). If $f = (i,j)$, we let $Y_{f,k} \doteq Y_{i,j,k}$, where $Y_{i,j,k}$ denotes the value of $y_{i,j}$ at the beginning of iteration k. We will show that

$$\forall k, \mathbf{E}\left[\prod_{f\in S} Y_{f,k}\right] \leq \mathbf{E}\left[\prod_{f\in S} Y_{f,k-1}\right] \tag{4.4}$$

Thus, we will have $\Pr[\bigwedge_{f\in S}(X_f = 1)] = \mathbf{E}\left[\prod_{f\in S} Y_{f,|E|+1}\right] \leq \mathbf{E}[\prod_{f\in S} Y_{f,1}] = \prod_{f\in S} x_{f,1} = \prod_{f\in S} \Pr[X_f = 1]$ and (P3) will hold.

Let us now prove (4.4) for a fixed k. In iteration k, exactly one of the following three events occur:

Event A: Two edges in S have their values modified. Specifically, let $A(f_1, f_2, \alpha, \beta)$ denote the event that edges $\{f_1, f_2\} \subseteq S$ have their values changed in the following

probabilistic way:

$$(Y_{f_1,k}, Y_{f_2,k}) = \begin{cases} (Y_{f_1,k-1} + \alpha, Y_{f_2,k-1} - \alpha) & \text{with probability } \beta/(\alpha+\beta) \\ (Y_{f_1,k-1} - \beta, Y_{f_2,k-1} + \beta) & \text{with probability } \alpha/(\alpha+\beta) \end{cases}$$

Suppose, for each $f \in S$, $Y_{f,k-1}$ equals some fixed a_f. Let $S_1 = S - \{f_1, f_2\}$. Then,

$$\mathbf{E}[\prod_{f \in S} Y_{f,k} | (\forall f \in S,\, Y_{f,k-1} = a_f) \wedge A(f_1, f_2, \alpha, \beta)] \;\; = \;\;$$
$$\mathbf{E}[Y_{f_1,k} \cdot Y_{f_2,k} | (\forall f \in S,\, Y_{f,k-1} = a_f) \wedge A(f_1, f_2, \alpha, \beta)] \prod_{f \in S_1} a_f$$

The above expectation can be written as $(\psi + \Phi) \prod_{f \in S_1} a_f$, where

$$\begin{aligned} \psi &= (\beta/(\alpha+\beta)) \cdot (a_{f_1} + \alpha) \cdot (a_{f_2} - \alpha), \text{ and} \\ \Phi &= (\alpha/(\alpha+\beta)) \cdot (a_{f_1} - \beta) \cdot (a_{f_2} + \beta). \end{aligned}$$

It is easy to show that $\psi + \Phi \leq a_{f_1} a_{f_2}$. Thus, for any fixed $\{f_1, f_2\} \subseteq S$ and for any fixed (α, β), and for fixed values a_f, the following holds:

$$\mathbf{E}[\prod_{f \in S} Y_{f,k} | (\forall f \in S,\, Y_{f,k-1} = a_f) \wedge A(f_1, f_2, \alpha, \beta)] \leq \prod_{f \in S} a_f$$

Hence, $\mathbf{E}[\prod_{f \in S} Y_{f,k} | A] \leq \mathbf{E}[\prod_{f \in S} Y_{f,k-1} | A]$.

Event B: Exactly one edge in the set S has its value modified. Let $B(f_1, \alpha, \beta)$ denote the event that edge $f_1 \in S$ has its value changed in the following probabilistic way:

$$Y_{f_1,k} = \begin{cases} Y_{f_1,k-1} + \alpha & \text{with probability } \beta/(\alpha+\beta) \\ Y_{f_1,k-1} - \beta & \text{with probability } \alpha/(\alpha+\beta) \end{cases}$$

Thus, $\mathbf{E}[Y_{f_1,k} | (\forall f \in S,\, Y_{f,k-1} = a_f) \wedge B(f_1, \alpha, \beta)] = a_{f_1}$. Letting $S_1 = S - \{f_1\}$, we

get that $\mathbf{E}[\prod_{f\in S} Y_{f,k}|(\forall f \in S,\, Y_{f,k-1} = a_f) \wedge B(f_1, \alpha, \beta)]$ equals

$$\mathbf{E}[Y_{f_1,k}|(\forall f \in S,\, Y_{f,k-1} = a_f) \wedge B(f_1, \alpha, \beta)] \prod_{f\in S_1} a_f = \prod_{f\in S} a_f.$$

Since this equation holds for any $f_1 \in S$, for any values a_f, and for any (α, β), we have $\mathbf{E}[\prod_{f\in S} Y_{f,k}|B] = \mathbf{E}[\prod_{f\in S} Y_{f,k-1}]$.

Event C: No edge has its value modified. Hence, $\mathbf{E}[\prod_{f\in S} Y_{f,k}|C] = \mathbf{E}[\prod_{f\in S} Y_{f,k-1}]$. Thus by the above case-analysis that considers which of events A, B and C occurs, we get that $\mathbf{E}[\prod_{f\in S} Y_{f,k}] \leq \mathbf{E}[\prod_{f\in S} Y_{f,k-1}]$. This completes the proof. ■

Properties (P1) and (P2) will both play a key role in the applications described in the subsequent sections of this chapter.

4.3 Social Network Modeling

Recent years have seen a growing interest in modeling the underlying graph of the Internet, WWW, and other such massive networks; see, e.g., [129, 42]. If we can model such graphs using appropriate random graphs, then we can sample multiple times from such a model and test candidate algorithms, such as Web-crawlers [35]. A particularly successful outcome of the study of such graphs has been the uncovering of the *power-law* behavior of the vertex-degrees of many such graphs (see, e.g., [36]). Hence, there has been much interest in generating (and studying) random graphs with a given degree-sequence (see, e.g., [50]). Web/Internet measurements capture a lot of connectivity information in the graph, in addition to the distribution of the degrees of the nodes. In particular, through repeated sampling, these models capture the probability with which a node of a certain degree d_1 might share an edge with a node of degree d_2. Our question here is: since a network is much more than its degree sequence, can we model connectivity in addition to the degree-sequence? Concretely,

given n, values $\{x_{i,j} \in [0,1] : i < j\}$, and a degree-sequence $d_1, d_2, \ldots, d_n$ (realized by the values $x_{i,j}$), we wish to generate an n-vertex random graph $G = (\{1, 2, \ldots, n\}, E)$ in which: **(A1)** vertex i has degree d_i with probability 1, and **(A2)** the probability of edge (i, j) occurring is $x_{i,j}$. (Note that we must have $d_i = \sum_j x_{i,j}$.) This is the problem we focus on, in order to take a step beyond degree-sequences.

Our dependent rounding scheme solves this problem when G is bipartite. However, can we get such a result for general graphs? Unfortunately, the answer is no: the reader can verify that no such distribution (i.e., random graph model) exists for the triangle with $x_{1,2} = x_{2,3} = x_{1,3} = 1/2$ (and hence with $d_1 = d_2 = d_3 = 1$). This example has $d_1 + d_2 + d_3$ being odd, which violates the basic property that the sum of the vertex-degrees should be even. However, even if the vertex-degrees add up to an even number, there are simple cases of non-bipartite graphs where there is no space of random graphs which satisfies (A1) and (A2). (Consider two vertex-disjoint triangles with all $x_{i,j}$ values being $1/2$, and connect the two triangles by and edge whose $x_{i,j}$ value is 1.) Thus, we need to compromise – hopefully just a little – for general graphs. One method in this context is to construct a random graph where each edge (i, j) is put in *independently*, with probability $x_{i,j}$. This preserves (A2), but does not do well with (A1): the only (high-probability) guarantee we get is that for each i, $|D_i - d_i| \leq O(\max\{\sqrt{d_i \log n}, (\log n)^{1-o(1)}\})$. We now show that we can do much better than this:

Theorem 14 *Given a degree-sequence $d_1, d_2, \ldots, d_n$, and values $\{x_{i,j} \in [0,1] : i < j\}$, we can efficiently generate an n-vertex random graph for which (A2) holds, and where the following relaxation of (A1) holds: with probability one for each vertex i, its (random) degree D_i satisfies $|D_i - d_i| \leq 2$. Letting m denote the number of nonzero $x_{i,j}$, the running time of our algorithm is $O(n + m^2)$.*

Thus, we get an essentially best-possible result. Recall that, in the bipartite rounding algorithm, if we encounter an even cycle, we "break" this cycle by prob-

abilistically rounding (at least) one of the edges in the cycle. Our algorithm for non-bipartite graphs also proceeds by probabilistically breaking cycles in the graph. We now describe the details of the algorithm.

We start with a graph with vertices $1, 2, \ldots, n$; for each nonzero value $x_{i,j}$, we put an edge between i and j that has a value or label $x_{i,j}$. We will closely follow our algorithm of Section 4.2, and borrow notation such as "floating edges" from there. In the following description, we use the terms *simple cycle* and *linked odd cycles* in the following sense: each vertex in a simple cycle has degree two; a pair of linked odd cycles is a pair of odd cycles sharing a common vertex that has degree four. The algorithm proceeds in four phases as follows. Throughout the execution of the algorithm, G will denote the subgraph given by the *currently-floating* edges of F.

Phase 1: While there exists a simple even cycle in G, do:
Pick a simple cycle C from G. Partition the edges in C into matchings M_1 and M_2. Probabilistically modify the edge values of M_1 and M_2 as in the bipartite rounding algorithm.

Phase 2: While there exists a pair of linked odd cycles in G, do:
Pick a pair of linked odd cycles C from G. Partition the edges in C into two sets M_1 and M_2 such that for any given vertex, the number of edges incident upon it in M_1 is the same as that in M_2. (It is easy to see that such a partition exists since C is a linked pair of odd cycles). Probabilistically modify the edge values of M_1 and M_2 as in the bipartite rounding algorithm (M_1 and M_2 were matchings in the case of bipartite graphs.)

Phase 3: While there exists an odd cycle in G, do:
Pick an odd cycle C from G and pick an arbitrary edge e in C. Let Y_e be the random variable which denotes the value of e's edge-label. Let the current value of Y_e be y_e. Round Y_e to one with probability y_e and to zero with the complementary probability.

Phase 4: All cycles in G have been broken by the previous phases and G is now a forest. Apply the bipartite rounding algorithm on G.

We now argue that the two properties claimed by Theorem 14 hold. For any fixed edge, the expected value of the edge-label does not change in any of the phases. Hence we see that (A1) holds, by using the same simple argument as in the proof of Lemma 11. Phases **1** and **2** do not change the fractional degree of any vertex. Crucially, each vertex belongs to *at most one* odd cycle at the beginning of Phase **3**. Thus, Phases **3** and **4** change the degree of any vertex by at most one each. Hence, at the end of our algorithm, the integral degree of any vertex differs from its fractional degree by at most two.

We now discuss how to implement this algorithm. We first decompose the graph into its biconnected components [41]. Some biconnected components are *trivial*, if they consist of a single edge. Other biconnected components always contain a cycle. The following proposition shows that it is easy to find an even cycle in a non-trivial biconnected component. Before we understand the proof, we need to define the concept of *bridges* of a graph $G = (V, E)$ with respect to a cycle C. A trivial bridge is an edge of the graph that connects two nodes on C that are not adjacent in C. These are simply chords on the cycle. Consider the graph induced by the vertices in $V \setminus C$. Let $B_1, \ldots, B_k$ be the connected components in this graph. Let E_i be the set of edges that connect vertices in B_i to vertices on C. The edges E_i together with the component B_i form a bridge in the graph [41]. If an edge $(u, v) \in E_i$ with $u \in B_i$ and $v \in C$, then v is an attachment point of the bridge.

Proposition 15 *A non-trivial biconnected simple graph is either exactly an odd cycle, or must contain an even cycle.*

Proof Assume that the biconnected component is not exactly an odd cycle. Find a cycle C in the biconnected component. Assume that the cycle is odd. Consider

the bridges of the graph with respect to the cycle C. Since the graph is biconnected, each bridge has at least two distinct attachment points on C. This yields a path in the graph that is disjoint from C that connects two nodes u and v on C. Since C has odd length, the two paths between u and v using edges of C are of opposite parity. Using one of them along with the path that avoids C we obtain a simple cycle of even length. ■

Phase 1 is implemented as follows. If a biconnected component is trivial, or an odd cycle, we do not process it for now. We process each remaining component to identify even cycles (the proof of Proposition 15 suggests how to do this algorithmically in linear time). Once we remove an edge of the even cycle, we further decompose the graph into its biconnected components in linear time. We repeat this until each biconnected component is either trivial, or exactly an odd cycle.

In Phase 2 we find linked odd cycles. If two components are non-trivial and share a common cut vertex, they form a pair of linked odd cycles. We can perform the rounding as described above, and delete one edge to break a cycle. Eventually, all odd cycles are disjoint and we can perform the rounding as in Phase 3. Finally, when the graph is acyclic, it is bipartite and the rounding can be done as described in Section 4.2. The total running time of the algorithm is $O(n + m^2)$. Phase 1 is the most expensive since each time we delete one edge, we have to reconstruct the biconnected components, which takes time linear in the size of the component.

4.4 Broadcast Scheduling

In this section, we study a scheduling problem in the *broadcasting* model. Traditional scheduling problems require each job to receive its own chunk of processing time. The growth of (multimedia) broadcast technologies has led to situations where certain jobs can be *batched* and processed together: e.g., users waiting to receive the same topic

in a broadcast setting. For example, all waiting users get satisfied when that topic is broadcast [21, 65, 39, 44, 18, 19]. The basic features of the model are as follows. There is a set of pages or topics, $P = \{1, 2, \ldots, n\}$, that can be broadcast by a broadcast server. We assume that time is discrete; for an integer t, the time-slot (or simply *time*) t is the window of time $(t-1, t]$. Any subset of the pages can be requested at time t. All users receive every page that is broadcast; our main problem is to construct a good broadcast-schedule. The default assumption is that the server can broadcast at most one page at any time; we will also consider resource-augmented solutions where the server is allowed to broadcast up to two pages per time-slot. We work in the offline setting in which the server is aware of all future requests.

Our objective is to minimize the average (equivalently, total) response time for the requests. Let (p, t) denote a request for page p arriving at time t; our goal is to schedule the broadcast of pages in a way that minimizes the total response time of all requests. The total response time is $\sum_{(p,t)} r_t^p (S_t^p - t)$, where for a request (p, t), S_t^p is the first time instance *after* t when page p is broadcast. As before, r_t^p denotes the number of requests for page p that arrive at time t. An *α-speed* broadcast schedule is one in which at most α pages are broadcast at any time instance. Let an (α, β)-algorithm stand for an algorithm that constructs an α-speed schedule whose expected cost is at most β times the cost of an optimal 1-speed solution. Gandhi *et al.* provided approximation algorithms for this problem which achieved the bounds of $(2, 2), (3, 1.5)$, and $(4, 1)$ [44]. We now improve all these by providing a bound of $(2, 1)$ using the dependent rounding technique.

An IP formulation for the problem is given below. The binary variable $y_{t'}^p = 1$ iff page p is broadcast at time slot t'. The binary variable $x_{tt'}^p = 1$ iff a request (p, t) is satisfied at time $t' > t$; i.e., if $y_{t'}^p = 1$ and $y_{t''}^p = 0, t < t'' < t'$. Also, T is the time of last request for any page. It is easy to check that this is a valid IP formulation.

The first set of constraints ensure that whenever a request (p, t) is satisfied at

time t', page p is broadcast at t'. The second set of constraints ensure that every request (p,t) is satisfied at some time $t' > t$. The third set of constraints ensure that at most one page is broadcast at any given time. The last two set of constraints ensure that the variables assume integral values. By letting the domain of $x^p_{tt'}$ and $y^p_{t'}$ be $0 \leq x^p_{tt'}, y^p_{t'} \leq 1$, we obtain the LP relaxation for this problem.

$$\begin{aligned}
&\text{Minimize} \sum_p \sum_t \sum_{t'=t+1}^{T+n} (t'-t) \cdot r^p_t \cdot x^p_{tt'} \\
&y^p_{t'} - x^p_{tt'} \geq 0 && \forall p, t, t' > t \\
&\sum_{t'=t+1}^{T+n} x^p_{tt'} \geq 1 && \forall p, t \\
&\sum_p y^p_{t'} \leq 1 && \forall t' \\
&x^p_{tt'} \in \{0,1\} && \forall p, t, t' \\
&y^p_{t'} \in \{0,1\} && \forall p, t'
\end{aligned} \tag{4.5}$$

The Algorithm Our scheduling algorithm starts with a 2-speed fractional solution $\mathcal{S}$ which is obtained as follows. We first solve the LP relaxation optimally. $\mathcal{S}$ is obtained by doubling the fraction of each page broadcast at each time slot by the LP solution. $\mathcal{S}$ is then rounded as follows. Recall that $\{y^p_t\}$ denotes non-negative values in $\mathcal{S}$ where p indexes pages and t indexes time-slots; in our 2-speed setting, $\sum_p y^p_t \leq 2$ for all t.

Given $\mathcal{S}$, the scheduling algorithm proceeds in two steps as described below.

Step 1. Construct a bipartite graph $G = (U, V, E)$ as follows. U consists of vertices that represent time slots. Let u_t denote the vertex in U corresponding to time t. Consider a page p and the time instances during which page p is broadcast fractionally in $\mathcal{S}$. Let these time slots be $\{t_1, t_2, \ldots, t_k\}$ such that $t_i < t_{i+1}$. We will group these time slots into some number $m = m(p)$ of windows, $W^p_j, 1 \leq j \leq m$, such that in each window except the first and the last, *exactly* one page p is broadcast fractionally. More formally, we will define non-negative values $b^p_{t,j}$ for each time-slot t and window

W_j^p, such that for each j: $b_{t,j}^p$ is derived from y_t^p in a natural way, the values t for which $b_{t,j}^p \neq 0$ form an interval, and $\sum_t b_{t,j}^p = 1$ for $2 \leq j \leq m-1$. (The first and last windows, W_1^p and W_m^p, may broadcast a full page or less.) The grouping of time slots into windows is done as follows. Choose $z \in (0, 1]$ uniformly at random; z represents the amount of service provided by the first window. (It suffices to use the same z for all pages p.) Intuitively, the windows represent contiguous chunks of page p broadcast in $\mathcal{S}$. The first chunk is of size z, the last chunk is of size at most one, and all other intermediate chunks are of size exactly one. Formally, for each time instance t_h, we will associate a fraction $b_{t_h,j}^p$ that represents the amount of contribution made by time slot t_h toward the fractional broadcast of page p in W_j^p. For all h, define $b_{t_h,1}^p$ and $b_{t_h,j}^p$, for $j \geq 2$ as follows. If $\sum_{i=1}^{h-1} y_{t_i}^p < z$ then $b_{t_h,1}^p = \min\{y_{t_h}^p, z - \sum_{t'<t_h, t' \in W_1^p} b_{t',1}^p\}$, and 0 otherwise. For all $j \geq 2$, if $\sum_{i=1}^{h-1} y_{t_i}^p < j - 1 + z$ then $b_{t_h,j}^p = \min\{y_{t_h}^p - b_{t_h,j-1}^p, 1 - \sum_{t'<t_h, t' \in W_j^p} b_{t',j}^p\}$ and 0 otherwise.

A time slot t_h belongs to W_j^p iff $b_{t_h,j}^p > 0$. This implies that a window W_j^p consists of consecutive time slots and that the total number of windows $m_p \in \{\lceil \sum_{t'} y_{t'}^p \rceil, \lceil \sum_{t'} y_{t'}^p \rceil + 1\}$. The vertex set V consists of vertices that represent pages. For each page p, we have vertices $v_1^p, v_2^p, \ldots, v_{m_p}^p$ in V. For all p and j, v_j^p is connected to vertices corresponding to timeslots in W_j^p. The value of an edge (v_j^p, u_{t_h}) is equal to $b_{t_h,j}^p$. The above is repeated for all pages p, using the same random value z. This construction is illustrated in Figure 4.1, in which a subgraph of G that is induced on the vertices and edges relevant to a particular page p are shown. For this example we choose $z = 1$ and $y_j^p, t_1 \leq j \leq t_7$, values are $0.3, 0.3, 0.5, 0.5, 0.2, 0.9, 0.8$.

Step 2. Perform dependent rounding in G. If an edge (v_k^p, u_{t_h}) gets chosen in the rounded solution, then we broadcast page p at time t_h.

This concludes the description of the scheduling algorithm. As we shall see, the use of the "random offset" z is critical in guaranteeing the performance of the algorithms that follow.

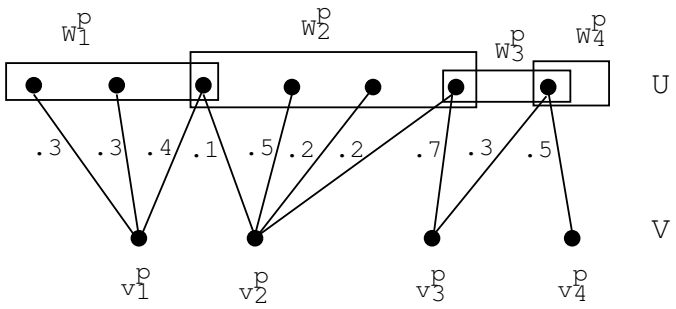


Figure 4.1: Subgraph of G relevant to page p whose $y^p_j, t_1 \leq j \leq t_7$, values are $0.3, 0.3, 0.5, 0.5, 0.2, 0.9, 0.8$. We set $z = 1$ here.

Analysis Consider a request r for page p. We assume w.l.o.g. that this request arrives at time 0. Let the cost incurred by the LP solution to satisfy this request be $O_r = \sum_{i=1}^{l} x_i t_i$, where $x_i > 0$ is the fractional amount of service r receives at time t_i and $\sum_{i=1}^{l} x_i = 1$. Let $1 \leq t_1 \leq t_2 \leq \cdots \leq t_l$. We also assume w.l.o.g. that there exists a $\lambda \in \{1, \ldots, l\}$ such that $\sum_{i=1}^{\lambda} x_i = 1/2$ (otherwise we can "split" an appropriate x_i into two fractions x_λ and $x_{\lambda'}$ to achieve this). In general, the time slots $t_1, \ldots, t_l$ will span three consecutive windows in $\mathcal{S}$; since r arrives at time 0, these are the first three windows in $\mathcal{S}$. Let the random variable Y denote the fraction of service which r receives from the first window in $\mathcal{S}$. Note that Y is distributed uniformly in the interval $(0, 1]$. Let $A_r(Y)$ and $B_r(Y)$ be the set of time-slots which belong to the first and second windows respectively that serve r. Define X_i to be the indicator random variable which is one iff p is broadcast by the rounded solution at time t_i. Let C_r be the random variable which denotes the cost incurred by the request r in the rounded solution. Define $s_i = \sum_{j=1}^{i} x_j$, with $s_0 = 0$. The variables relevant to request r are illustrated in Figure 4.2.

Our Approach. Our main goal now is to prove Lemma 20, which shows that $\mathbf{E}[C_r] \leq O_r$; as we shall see, property (P2) will play a critical role. To prove Lemma 20, we bound $\mathbf{E}[C_r]$ in two different ways. First, Lemma 16 uses the property that whatever

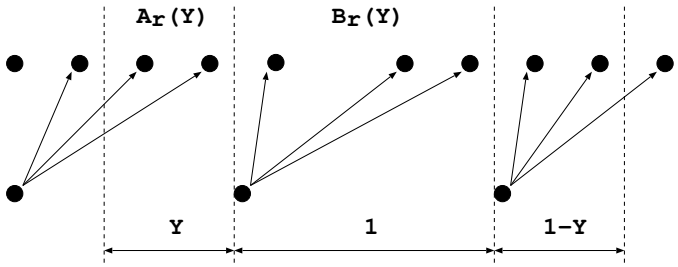


Figure 4.2: Variables relevant to request r: vertices on top represent time-slots, and vertices below represent windows. Values Y and 1 are the amount of service r receives from time slots in $A_r(Y)$ and $B_r(Y)$ respectively.

the value of Y is, the set of time-slots $B_r(Y)$ broadcast page p with probability 1; thus, it suffices to bound this cost. This bound alone does not suffice for our purposes; we can only show that $\mathbf{E}[C_r] \leq (4 - 2\sqrt{2})O_r$ in this manner. So, as a second approach to bound $\mathbf{E}[C_r]$, Lemma 17 starts with the observation that r needs to wait for a broadcast of p from $B_r(Y)$ only if the event

$$\mathcal{E} \equiv \text{ (page } p \text{ was not broadcast in } A_r(Y))$$

happens. Now, conditional on $\mathcal{E}$, the distribution of broadcasts in $B_r(Y)$ could be quite arbitrary, but (P2) still ensures that there will be a broadcast of p in $B_r(Y)$! Thus, the worst case is that conditional on $\mathcal{E}$, p is broadcast in the *last* time-slot of $B_r(Y)$. Lemma 17 bounds $\mathbf{E}[C_r]$ using this idea. The average of these two bounds is also an upper-bound on $\mathbf{E}[C_r]$; Lemma 19 then shows that in the resulting optimization problem with the x_i as variables, the maximum possible value of $\mathbf{E}[C_r]/O_r$ is 1.

Lemma 16 $\mathbf{E}[C_r] \leq 2\sum_{i=1}^{\lambda}(2s_{i-1} + x_i)x_i t_i + 2\sum_{i=\lambda+1}^{l}(2 - 2s_i + x_i)x_i t_i$.

Proof Let $f(y) = \sum_{t_i \in B_r(y)} \mathbf{E}[X_i \mid (Y = y)]t_i$. Since page p will be transmitted in at least one of the slots in $B_r(Y)$ by (P2), we have $C_r \leq \sum_{t_i \in B_r(Y)} X_i t_i$ with probability 1. Hence $\mathbf{E}[C_r \mid (Y = y)] \leq f(y)$ and $\mathbf{E}[C_r] \leq \int_0^1 f(y)\,dy$. We now

calculate the contribution of each t_i to this integral. There are two cases:

Case 1: $i \leq \lambda$**.** If $Y \leq 2s_{i-1}$, then t_i fractionally broadcasts $2x_i$ units of p in the second window. If $2s_{i-1} < Y \leq 2s_i$, then t_i fractionally broadcasts $(2s_i - Y)$ units of p in the second window. If $Y > 2s_i$, then t_i does not belong to the second window. All three scenarios are illustrated in the Figure 4.3.

Case 2: $\lambda < i \leq l$**.** If $Y \geq 2s_i - 1$, then t_i fractionally broadcasts $2x_i$ units of p in the second window. If $2s_{i-1} - 1 \leq Y < 2s_i - 1$, then t_i fractionally broadcasts $(Y + 1 - 2s_{i-1})$ units of p in the second window. Otherwise, t_i does not belong to the second window.

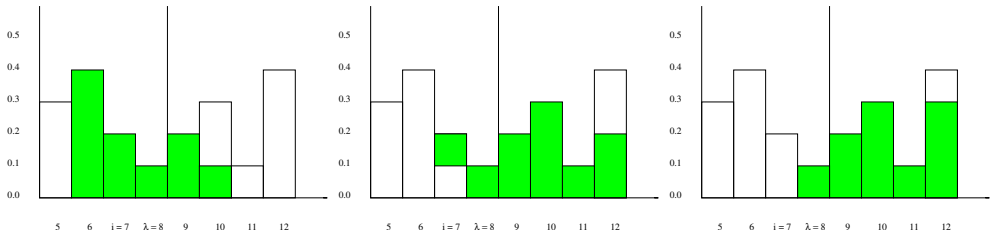


Figure 4.3: Contribution of slot i to the integral $\int_0^1 f(y)\,dy$: slots $5, \ldots, 12$ fractionally serve request r in the LP solution. For each slot, the height of the bar indicates the fraction of page p broadcast in that slot in the 2-speed fractional solution. The request receives exactly one unit of page p from slots $5, \ldots, \lambda = 8$. The fractional broadcasts of page p within the time window $B_r(Y)$ is denoted by the shaded regions. The contribution of slot i to the integral depends on its position relative the start of the shaded region Y. (a) $Y \leq 2s_{i-1}$; slot i is completely included in the shaded region. (b) $2s_{i-1} < Y \leq 2s_i$; slot i is partially included in the shaded region. (c) $Y > 2s_i$; slot i is completely omitted from the shaded region.

By property (P1) of dependent rounding, if t_i fractionally broadcasts some a units of p in the second window, then the probability that page p is broadcast at time

t_i by our generic algorithm, is exactly a. Thus,

$$\begin{aligned}
\mathbf{E}[C_r] &\leq \int_0^1 f(y)\,dy \\
&= \sum_{i=1}^{\lambda}\left(2s_{i-1}2x_it_i + \int_{2s_{i-1}}^{2s_i}((2s_i-y)t_i)\,dy\right) \\
&+ \sum_{i=\lambda+1}^{l}\left((2-2s_i)2x_it_i + \int_{2s_{i-1}-1}^{2s_i-1}(y+1-2s_{i-1})\,dy\right) \\
&= 2\sum_{i=1}^{\lambda}(2s_i+x_i)x_it_i + 2\sum_{i=\lambda+1}^{l}(2-2s_i+x_i)x_it_i.
\end{aligned}$$

■

Lemma 17 $\mathbf{E}[C_r] \leq 2\sum_{i=1}^{\lambda}(1-2s_i+x_i)x_it_i + 2\sum_{i=\lambda+1}^{l}(2-2s_i+x_i)x_it_i$.

Proof Suppose we condition on the event "$Y = y$". Let $Z_r(y)$ be the indicator random variable which is one iff $A_r(y)$ does not serve r in the rounded solution. Let $T_r(y)$ be the random variable denoting the last time slot in $B_r(y)$. Since r is served by either the first or the second window in the rounded solution, the following holds with probability 1:

$$C_r \leq \left(\sum_{t_i \in A_r(y)} X_it_i\right) + Z_r(y)T_r(y).$$

So we have

$$\mathbf{E}[C_r \mid (Y=y)] \leq \left(\sum_{t_i \in A_r(y)} \mathbf{E}[X_i \mid (Y=y)]t_i\right) + \mathbf{E}[Z_r(y)]\mathbf{E}[T_r(y)].$$

Since $\mathbf{E}[Z_r(y)] = (1-y)$,

$$\mathbf{E}[C_r] \leq \int_{y=0}^{y=1}\left(\sum_{t_i \in A_r(y)} \mathbf{E}[X_i \mid (Y=y)]t_i\right)\,dy + \int_{y=0}^{y=1}(1-y)T_r(y)\,dy.$$

We now compute the contribution of each t_i to each of the two integrals. There are

two cases:

Case 1 $i \leq \lambda$: t_i never contributes to the second integral since it is never the last time slot in the second window. If $2s_i < y \leq 1$ then t_i contributes $2x_i$ to the first integral. If $2s_{i-1} < y \leq 2s_i$ then t_i contributes $2s_i - y$ to the first integral. If $0 < y \leq 2s_{i-1}$ it contributes nothing to either of the two integrals.

Case 2 $i \geq \lambda + 1$: t_i never contributes to the first integral since it is never part of the first window. $T_r(y) = t_i$ iff $2s_{i-1} - 1 < y \leq 2s_i - 1$.

Thus, we have

$$\mathbf{E}[C_r] \;=\; \sum_{i=1}^{\lambda} \left((1 - 2s_i) 2x_i t_i + \int_{y=2s_{i-1}}^{2s_i} (2s_i - y) t_i \, dy \right) + \sum_{i=\lambda+1}^{l} \int_{y=2s_{i-1}-1}^{2s_i - 1} (1 - y) t_i$$

Simplifying the above expression yields the lemma. ■

Lemma 18 $\mathbf{E}[C_r] \leq \sum_{i=1}^{\lambda} x_i t_i + 2 \sum_{i=\lambda+1}^{l} (2 - 2s_i + x_i) x_i t_i$.

Proof The lemma follows by averaging the bounds given by Lemmas 16 and 17. ■

The term "$2 \sum_{i=\lambda+1}^{l} (2 - 2s_i + x_i) x_i t_i$" is next upper-bounded by Lemma 19. For convenience, Lemma 19 relabels the values $t_{\lambda+1}, t_{\lambda+2}, \ldots t_l$ as $v_1, v_2, \ldots, v_j$, and the values $x_{\lambda+1}, x_{\lambda+2}, \ldots x_l$ as $z_1, z_2, \ldots, z_j$.

Lemma 19 *Let values* $1 \leq v_1 \leq v_2 \leq \cdots \leq v_j$ *be given, and let* $z_1, \ldots, z_j$ *be real-valued variables. Consider the problem of maximizing the value* f *subject to the fol-*

lowing constraints:

$$\begin{aligned} f &= \frac{2\sum_{i=1}^{j}(2-2s_i+z_i)z_iv_i}{\sum_{i=1}^{j} z_iv_i} \\ s_i &= 1/2 + \sum_{u=1}^{i} z_u \qquad \forall i \\ \textstyle\sum_{i=1}^{j} z_i &= 1/2 \\ z_i &\geq 0 \qquad \forall i \end{aligned}$$

The maximum value of f subject to these constraints is at most 1.

Proof The problem has a maximum, since we have a continuous objective function defined on a compact domain. Let f^* be the maximum value, and let the values of the variables in a maximizing solution be z_i^* and s_i^*. We start by making some observations about the z_i^* values, which hold w.l.o.g. Note first that those z_i^* that are zero can be eliminated from the problem. Next, if $v_i = v_{i+1}$ for some i, it is easy to see that the objective function does not change if we increment z_{i+1}^* by z_i^*, and reset z_i^* to 0; so, we can assume that $1 \leq v_1 < v_2 < \cdots < v_j$. If exactly one of the z_i^* values is non-zero, then $f^* = 1$. Hence, assume w.l.o.g. that all z_i^* values are non-zero, that there are at least two of these, and that $1 \leq v_1 < v_2 < \cdots < v_j$. Let $N = 2\sum_{i=1}^{j}(2-2s_i^*+z_i^*)z_i^*v_i$ and $D = \sum_{i=1}^{j} z_iv_i$, so that $f^* = N/D$. We now examine the structure of this solution by perturbing z_1^* and z_2^*. Specifically, increase and decrease the values of z_1^* and z_2^* respectively by an infinitesimal value ϵ. Clearly, the new solution is still feasible. The value of N changes by $\Delta N + O(\epsilon^2)$, where $\Delta N = 2\epsilon(2-2s_1)(v_1-v_2)$; the value of D changes by $\Delta D = \epsilon(v_1-v_2)$. Observe that for f^* to be the maximum value of the optimization problem, the following is a necessary condition: $f^* = N/D = \Delta N/\Delta D = 2(2-2s_1)$.

Repeating the above arguments for different z_i^* leads to the following: $f^* = 2(2-2s_1) = 2(2-2s_2) = \cdots = 2(2-2s_{j-1})$ and hence, $s_1 = s_2 = \ldots = s_{j-1}$. So, there

are at most two non-zero z_i^* values, which we take to be $z_1^* = 1/2 - z$ and $z_2^* = z$. We now have

$$\begin{aligned} f^* &= \frac{2(2-2s_1+z_1^*)z_1^*v_1 + 2(2-2s_2+z_2^*)z_2^*v_2}{z_1^*v_1 + z_2^*v_2} \\ &= \frac{2(1/4-z^2)v_1 + 2z^2v_2}{v_1/2 + z(v_2-v_1)} \\ &= \frac{v_1/2 + 2z^2(v_2-v_1)}{v_1/2 + z(v_2-v_1)} \\ &\leq \max\{1, 2z\} \\ &\leq 1. \end{aligned}$$

■

Recall that O_r denotes the cost incurred by the LP solution to serve r. Then, our key lemma is:

Lemma 20 $\mathbf{E}[C_r] \leq O_r$.

Proof Lemma 18 implies that

$$\begin{aligned} \frac{E[C_r]}{O_r} &\leq \frac{\sum_{i=1}^{\lambda} x_i t_i + 2\sum_{i=\lambda+1}^{l}(2-2s_i+x_i)x_i t_i}{\sum_{i=1}^{l} x_i t_i} \\ &\leq \max\left\{1, \frac{2\sum_{i=\lambda+1}^{l}(2-2s_i+x_i)x_i t_i}{\sum_{i=\lambda+1}^{l} x_i t_i}\right\}. \end{aligned}$$

The lemma now follows from Lemma 19. ■

Theorem 21 *Our rounding scheme yields a* 2*-speed* 1*-approximate solution. Furthermore, it leads to the following per-user guarantee. Suppose each user-request* $r = (p, t)$ *comes with a delay (response-time) bound* D_r*. Then, there is an efficient algorithm that does the following: (i) it either proves that there is no* 1*-speed solution that satisfies each request within its response-time bound, or (ii) it constructs a randomized* 2*-speed schedule such that for each request* r*,*

- *the expected response time of r is at most D_r, and*
- *with probability 1, the response time of r is at most $2 \cdot D_r$.*

Proof Our algorithm constructs a 2-speed solution since the fractional value of edges incident on a vertex in U is at most 2: by property (P2) of dependent rounding, at most two edges will be incident on any vertex in U in the rounded solution. The claim that we have an 1-approximate solution in expectation, follows from Lemma 20 and the linearity of expectation.

As for the per-user guarantee, we proceed as follows. Given a delay bound D_r for each user r, we start by modifying our IP. We add the extra constraint

$$\forall r = (p,t), \quad \sum_{t'=t+1}^{T+n} (t'-t) \cdot r_t^p \cdot x_{tt'}^p \leq D_r.$$

We also remove the objective function, and simply ask for a feasible solution. We next solve the LP relaxation. If it has no feasible solution, then we halt, declaring that there is no 1-speed solution that satisfies each request within its response-time bound. Otherwise, suppose the LP-solver returns a feasible solution. Then, we proceed as above (doubling the y_t^p values and running the generic algorithm). Consider any request $r = (p,t)$. Lemma 20 shows that the expected response time of r in the randomized schedule constructed, is at most D_r. Finally, since at least one of the time-slots in $B_r(Y)$ will transmit page p with probability 1, it is easily seen that with probability 1, the response time of r is at most $2 \cdot D_r$. ■

Chapter 5

Sweep Scheduling Algorithms

5.1 Introduction

High-performance computing has evolved as the main enabling technology for scalable simulation and analysis of several important physical and biological processes. In this Chapter, we consider the sweep scheduling problem, which is motivated by the fast simulation of *radiation transport methods* using massively parallel machines. Radiation transport methods are commonly used in the simulation and analysis of a wide variety of physical phenomena. In its generality, this process involves computing the propagation of radiation flux across a physical region. The physical region is modeled as a collection of spatial cells, and the various cells constitute an unstructured mesh or a graph; the radiation propagation across the cells is modeled as a *sweep* through the vertices of the graph. Radiation transport methods underlie the dynamics of several physical applications such as medical imaging, nuclear reactor design, weapons effect, and the spread of forest fires [99, 104, 102].

A single sweep involves solving a large collection of equations associated with each mesh element locally; these local computations in the mesh proceed in a specified order for each direction in which the sweep occurs. Figure 5.1 shows an instance of

a mesh partitioned into cells, and a direction i. The computation on a cell, say cell 4, requires information from either the boundary conditions (along edge $\bar{ab}$) or from other cells "upstream" of this cell (along edge $\bar{bc}$); no information is needed from cells "downstream" of this cell, e.g. cells 5 and 7, in this example. These terms (i.e., "upstream" and "downstream") can be made precise, by looking at the angle between the direction i and the normals to these edges, but we refer the reader to [99, 104] for technical details about this.

From a computational standpoint, this means that each direction induces a set of precedence constraints captured by a dependency graph; for instance, in Figure 5.1, cell 4 depends on cell 2 for information, and so there is a precedence $(2, i) \rightarrow (4, i)$ in the corresponding DAG. In each direction, the dependency graph is different, but is induced on the same set of mesh elements - this is captured by the notation (v, i), which stands for the copy of cell v in direction i. As in Pautz [99, 104], there could be cyclic dependencies in the case of unstructured meshes, but there are known algorithms for detecting and removing cycles (see, e.g. [103]). Therefore, without loss of generality, we will assume that the dependency graph in each direction is a directed acyclic graph (DAG). The objective of the sweep scheduling problem is to assign the mesh cells to processors, and construct a schedule of the *smallest length* (or *makespan*), that respects all the precedence constraints. Messages that need to be sent from one processor to another incur communication delays, which could increase the makespan - in the models of Rayward-Smith [110] and Hwang et al. [60], for each edge $((v, i), (w, i))$ in the precedence graph, task (w, i) can start only certain time T after task (v, i) is completed if (v, i) and (w, i) are assigned to different processors - here, T corresponds to the time it takes for the information about (v, i) to be sent to the processor that is going to process (w, i).

We develop the first known algorithm for the sweep scheduling problem with provable performance guarantees. Our algorithm does not require any geometric

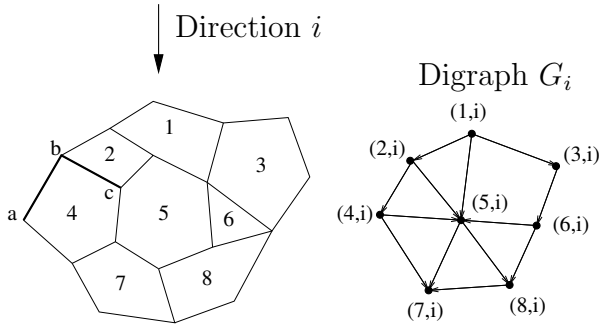


Figure 5.1: Example of a mesh and the digraph induced by direction i. The heavy edges of cell 4 show the edges from which information is needed before processing it. The edge $\bar{ab}$ is a boundary edge, whereas the edge $\bar{bc}$ shares an edge with cell 2, which is upstream to cell 4, w.r.t. this direction. The figure on the right shows the DAG corresponding to the mesh for direction i, and the nodes in the DAG are labeled as tuples (v, i), for each mesh cell v. There is an edge from (v, i) to (w, i) in the DAG if v is upstream of w w.r.t. this direction.

assumptions about the precedence constraints, and therefore, works in more general settings. For the sake of analytical tractability, we focus mainly on the problem which ignores the communication costs between processors - even this simplified version generalizes the well known precedence constrained scheduling problem [51], and is $\mathcal{NP}$-complete. In contrast, none of the known heuristics for sweep scheduling [99, 104] have provable performance guarantees, and it is not clear how to use them in the presence of more general, non-geometric precedence constraints.

We design the *Random Delay* algorithm, and analyze it rigorously to show that it gives an $O(\log^2 n)$ approximation (n is the number of mesh elements). We then show that a modification of this algorithm, coupled with an improved analysis leads to an $O(\log m \log\log\log m)$–approximation, where m is the number of processors. To our knowledge, these are the first algorithms with provable performance guarantees; in contrast, for all the heuristics studied by Pautz [99] there are worst case instances, admittedly not mesh like, where the schedule length could be $\Omega(m)$ times the optimal. While the running time is usually not crucial, it is interesting to note that our algorithms run in time almost linear in the length of the schedule. The algorithms can also be extended for certain communication models proposed by Rayward-Smith [110]

and Hwang et al. [60]. The extended algorithm yields a $O(C_{\max} \log m \log\log\log m)$ bound on the makespan in the communication latency model of [110, 60], where $C_{\max}$ denotes the maximum interprocessor communication latency.

Related Work. Because of its general applicability, sweep scheduling has been an active area of research. When the mesh is very regular, the KBA algorithm [71] is known to be essentially optimal. However, when the mesh is irregular, or unstructured, it is not easy to solve. There has been a lot of research in developing efficient algorithms for sweep scheduling, by exploiting the geometric structure, for instance by Pautz [99] and Plimpton et al. [104, 103]. The results of Amato et al. [3] and Mathis et al. [91] develops a theoretical model for finding good decomposition techniques that can be used along with other sweep scheduling algorithms. However, *none* of these heuristics has been analyzed rigorously, and *worst case* guarantees on their performance (relative to the optimal schedule) are not known. The S_n-sweeps application has a very high symmetry, arising from the directions being spread out evenly, but in other applications where this problem is relevant (such as in the parallel implementation of *EpiSims* [40] and *Transims* simulators), such symmetry does not exist. In such scenarios, it is unclear how the heuristics of [99, 104, 91] would work.

Scheduling problems in general have a rich history and much work has gone into theoretical algorithmic and hardness results, as well as the design of heuristics tailor-made for specific settings; see Karger et al. [67] for a comprehensive survey of the theoretical work on scheduling. The precedence constrained scheduling problem was one of the first problems for which provable approximation algorithms were designed, and is denoted as $P|prec|C_{max}$ in the notation of Graham et al. [51] who also give a simple $2 - \frac{1}{m}$ approximation algorithm; there has been an enormous amount of subsequent research on other variants of this problem (see [67]). However, there has been very little work that includes communication cost - the only model studied in this context was introduced by Rayward-smith [110], and is denoted $P|prec, p, c|C_{max}$.

This model involves communication latencies of the following form: for any edge (v, w), if v is assigned to processor i and w is assigned to processor $i' \neq i$, then w can be processed only $c_{ii'}$ time units after v has been completed, with $c_{ii'}$ denoting the time needed to send a message from processor i to processor i'; the goal is to minimize the makespan, with this communication latency incorporated. All the known results are about the special case of uniform communication delays, i.e., $c_{ii'} = c, \forall i, i'$, and usually, $c = 1$. Hoogeven, Lenstra and Veltman [59] showed that it is $\mathcal{NP}$-complete to get an approximation better than $\frac{5}{4}$. Rayward-Smith [110] and Hwang et al. [60] give constant factor approximation algorithms in this model, which was improved by Munier and Hanen [96] to $\frac{7}{3} - \frac{4}{3m}$. A generalization of Rayward-Smith's model is proposed by Hwang et al. [60]; our results hold for this model.

5.2 Preliminaries

We are given an unstructured mesh $\mathcal{M}$ consisting of a collection of n cells, a set of k directions and m processors. The mesh induces a natural graph $G(V, E)$: cells of the mesh correspond to the vertices and edges between vertices correspond to adjacency between mesh elements. A direction i induces a directed graph with the vertex set being identical to V, and a directed edge from u to v is present if and only if u and v are adjacent in G *and* the sweep in direction i requires u to be done before v. Figure 5.1 illustrates how a digraph is induced in an irregular, 2-dimensional mesh: for example, vertex 5 cannot be solved before its upstream neighbor 2 is solved, which induces a directed edge from 2 to 5 in the corresponding digraph. We assume in the following that the induced digraphs are acyclic (otherwise we break the cycles using the algorithm of [103]) and call them *directed acyclic graphs (DAG)*.

Thus, there is a copy of each vertex v for each direction; we will denote the copy of vertex v in direction i by (v, i) and call this (cell,direction) pair a *task*. The

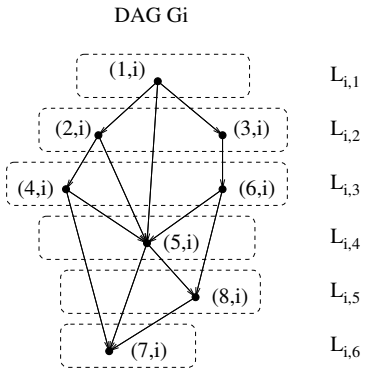


Figure 5.2: Levels of the digraph shown in Figure 5.1.

DAG in direction i will be denoted by $G_i(V_i, E_i)$, where $V_i = \{(v, i) \mid v \in V\}$.

An *instance* of a sweep scheduling problem is given by a vertex set V (the cells), k DAGs $G_i(V_i, E_i), i = 1, \ldots, k$ (the precedence constraints), and m processors. A *feasible solution* to the sweep scheduling problem is a schedule that processes all the DAGs, so that the following constraints are satisfied.

1. The precedence constraints for each DAG $G_i(V_i, E_i)$ must be satisfied. That is, if $((u, i), (v, i)) \in E_i$, then task (u, i) must be processed before task (v, i) can be started.

2. Each processor can process one task at a time and a task cannot be preempted-empted.

3. Every copy of vertex v must be processed on the same processor for each direction i.

Our overall *objective* is to minimize the computation time of all sweeps subject to the above constraints. We will assume that each task takes uniform time p to be processed, and there exists a communication cost of uniform time $C_{\max}$ between processors. In reality, interprocessor communication will increase the makespan in a way that is hard to model. We will consider the following two objectives for ease of analytical tractability: (i) the *makespan* of the schedule assuming no communication

cost, that is, the time it takes to process all tasks on m processors according to a certain schedule without taking communication cost into account, and, (ii) makespan with communication delays; we assume a communication delay of $C_{\max}$ is incurred after each step of computation, when all the processors exchange the messages needed to finish all the communication.

Levels. Given k DAGs $G_i(V_i, E_i), i = 1, \ldots, k$, we can form levels (also called layers) as follows: for DAG $G_i(V_i, E_i)$, layer $L_{i,j}$ is the set of vertices with no predecessors after vertices $L_{i,1} \cup \cdots \cup L_{i,j-1}$ have been deleted. We define D as the maximum number of layers in any direction. In Figure 5.2 we show how levels are formed for the example in Figure 5.1. Note that if we completely process all the cells in one level in arbitrary order before we start processing cells in the next level, we have processed the cells in an order that satisfies the precedence constraints. We will sometimes call a vertex (u, i) a leaf (or a sink) if the out-degree is 0. Similarly a node with in-degree 0 is called root (or a source).

List Scheduling. Throughout this chapter, we will use list scheduling at various places. In list scheduling, we may assign a priority to each task. If no priorities are assigned to the tasks, all tasks are assumed to have the same priority.

A task is said to be *ready*, if it has not been processed yet, but all its ancestors in the dependence graph have been processed. At each timestep t, let us denote by $R(t) \subset V \times \{1, \ldots, k\}$ the subset of tasks that are ready. We further denote by $R_P(t) \subset R(t)$ the subset of tasks that are ready and allowed to be processed by processor P. The list scheduling algorithm now proceeds such that for each timestep t, it assigns to each processor P the task of highest (or lowest) priority in $R_P(t)$. Ties are broken arbitrarily. If $R_P(t)$ is empty, processor P will be idle at time t.

5.3 Provable Approximation Algorithms

In this section, we will start by assuming that all processing costs are uniform and there are no communication costs (i.e., $p = 1$ and $C_{\max} = 0$). We first present two randomized approximation algorithms, both with an approximation guarantee of $O(\log^2 n)$. The underlying intuition behind both these algorithms is simple and is as follows. We first combine all the DAGs G_i into a single DAG G using the "random delays" technique. Next, we assign each vertex to a random processor. Each randomization serves to do contention resolution: the random assignment ensures that each processor roughly gets the same number of mesh elements, the random delay ensures that at each layer of the combined DAG, we do not have too many tasks corresponding to any given cell. Thus the two randomized steps taken together ensure the following property: at a particular level l of the combined DAG G, there are "relatively few" tasks to be scheduled in a particular processor. We now expand each level into appropriate time slots to obtain a valid sub-schedule for this level. The final schedule can be constructed by merging the sub-schedules for each of the levels. Note, however, that both the above randomized steps are likely to lead to huge communication costs. This can be improved significantly by first doing a decomposition into blocks and then doing the random assignment on the blocks. In Section 5.3.3 we present a slightly modified algorithm and a much more careful analysis, which gives an approximation guarantee of $O(\log m \log\log\log m)$. In Section 5.3.4, we outline an approach for bounding the makespan in the presence of communication costs.

5.3.1 Random Delays Algorithm

We now present our first algorithm for the sweep scheduling problem, called "Random Delay" (see Algorithm 1). In the first step, we choose a random delay X_i for each DAG G_i. In the second step, we combine all the DAGs G_i into a single DAG G

Algorithm 1 Random Delay

1: For all $i \in [1, \ldots, k]$, choose $X_i \in \{0, \ldots, k-1\}$ uniformly at random.

2: Form a combined DAG G as follows: $\forall r \in \{1, \ldots, D+k-1\}$, define $L_r = \bigcup_{\{i:X_i<r\}} L_{i,r-X_i}$. The edge $((u,i),(v,i)))$ is present in G, if and only if there exists an edge $((u,i),(v,i))$ in G_i.

3: For each vertex $v \in V$, choose a processor uniformly at random from $\{1, \ldots, m\}$.

4: Construct a schedule by processing layers $L_1, L_2, \ldots$ sequentially in that order:

- Layer L_{r+1} is processed only after all tasks in L_r have been processed.
- Within each layer L_r, process the tasks assigned to each processor in any arbitrary order.

using the random delays chosen in first step. Recall that $L_{i,j}$ denotes the set of tasks which belong to the level j of the DAG G_i. Specifically, for any i and j, the tasks in $L_{i,j}$ belong to the level r in G, where $r = j + X_i$. The edges in G between two tasks are induced by the edges in the original DAGs G_i: if the edge $((u,i),(v,i))$ exists in G_i then it also exists in the combined DAG G. It is easy to see that all the edges in G are between successive levels, and all the original precedence constraints are captured by the new DAG G. The third step involves assigning a processor chosen uniformly at random for each vertex v (and hence for all its copies in G). The fourth and the final step involves computing the schedule. This is done by computing a sub-schedule for each of the layers separately and merging these schedules. Within each layer, the tasks are scheduled using a greedy approach: tasks assigned a particular processor are scheduled in an arbitrary sequence. In the final schedule, all tasks in level L_r are processed before any task in level L_{r+1} is processed.

We now analyze the performance of the above algorithm. We first state the following basic facts from probability theory.

Lemma 22 ***(The Chernoff-Hoeffding Bound and its variants [30, 58])*** *Given independent r.v.s $X_1, \ldots, X_t \in [0,1]$, let $X = \sum_{i=1}^{t} X_i$ and $\mu = \mathbf{E}[X]$.*

a. For any $\delta > 0$, $\Pr[X \geq \mu(1+\delta)] \leq G(\mu, \delta)$, where $G(\mu,\delta) = \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu}$. In particular, for any sufficiently large $c \geq 0$,

$$\Pr[X > c \log n(\max\{\mu, 1\})] < \frac{1}{n^{O(1)}} . \tag{5.1}$$

b. There exists a constant $a > 0$ such that the following holds. Given $\mu > 0$ and $p \in (0,1)$, suppose a function $F(\mu,p) \geq \mu$ is defined as follows:

$$F(\mu,p) = \begin{cases} a \cdot \frac{\ln(p^{-1})}{\ln(\ln(p^{-1})/\mu)} & \text{if } \mu \leq \ln(p^{-1})/e \\ \mu + a \cdot \sqrt{\frac{\ln(p^{-1})}{\mu}} & \text{otherwise} \end{cases} \tag{5.2}$$

Then, defining $\delta = F(\mu,p)/\mu - 1 \geq 0$, we have $G(\mu,\delta) \leq p$; in particular, $\Pr[X \geq F(\mu,p)] \leq p$.

The following corollary follows.

Corollary 23 *Let $X_1, \ldots, X_n \in [0,1]$ be independent random variables and let $X = \sum_{i=1}^{n} X_i$. Let $\mathbf{E}[X] \leq \mu$. Then, for any sufficiently large $c \geq 0$,*

$$\Pr[X > c \log n(\max\{\mu, 1\})] < \frac{1}{n^{c}} . \tag{5.3}$$

Proof It can be checked that for $\delta \geq 4$, $\left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu} \leq e^{-\delta\mu}$. Now let $\delta = c \log n$. We then get that $\Pr[X > c \log n \mu] < e^{-c \log n \mu}$. For $\mu \geq 1$ we further get $e^{-c \log n \mu} = 1/n^{c\mu} \leq 1/n^{c}$. ■

Let $\mathcal{S}$ be the schedule produced by our algorithm. In the following analysis, unless otherwise specified, level L_r refers to level r of DAG G.

Lemma 24 *For all $v \in V$, and for each layer L_r, with high probability, the number of copies of v in L_r is at most $\alpha \log n$ with high probability, where $\alpha > 0$ is a constant. Specifically, this probability is at least $1 - \frac{1}{n^\beta}$, where β is a constant which can be made suitably large by choosing α appropriately.*

Proof Let $Y_{r,v,i}$ be the indicator random variable which is 1 if task (v, i) is in layer L_r and 0 otherwise. Since we choose X_i randomly from $\{0, \ldots, k-1\}$, we have $\Pr[Y_{r,v,i} = 1] \leq \frac{1}{k}$. Let $N_{r,v} = \sum_i^k Y_{r,v,i}$ be the random variable that denotes the number of copies of v in layer L_r. By linearity of expectation, we have $\mathbf{E}[N_{r,v}] = \sum_i^k \mathbf{E}[Y_{r,v,i}] = \sum_i^k \Pr[Y_{r,v,i} = 1] \leq \frac{k}{k} = 1$. Applying Lemma 22(a), we have $\Pr[N_{r,v} > c \log n] < \frac{1}{n^{O(1)}}$. Let $\mathcal{E}$ denote the event that there exists a vertex u and a layer l such that the number of copies of u in l is $> c \log n$. By the union bound, we have $\Pr[\mathcal{E}] \leq \sum_{v,r} \Pr[N_{r,v} > c \log n] < \sum_{v,r} \frac{1}{n^{O(1)}} \leq \frac{n^2}{n^{O(1)}} \leq \frac{1}{n^\beta}$, by choosing c suitably large. ■

For each layer L_r, define the set $V_r = \{v \mid \exists i \text{ such that } (v, i) \in L_r\}$. The following lemma holds.

Lemma 25 *For any level L_r and any processor P, the number of tasks that are assigned to P from L_r is at most $\alpha' \max\{\frac{|V_r|}{m}, 1\} \log^2 n$ with high probability where $\alpha' > 0$ is a constant. Specifically, this probability is at least $1 - \frac{1}{n^{\beta'}}$, where β' is a constant which can be made suitably large by choosing α' appropriately.*

Proof Consider any level L_r and a processor P. Let $Y_{P,v}$ be the indicator variable which is one if vertex v is assigned to processor P and zero otherwise. Due to the random assignment, we have $\Pr[Y_{P,v} = 1] = \frac{1}{m}$. Let $N_{P,r} = \sum_{v \in V_r} Y_{P,v}$ be the random variable which denotes the number of vertices in V_r that are assigned to P. By linearity of expectation, we have $\mathbf{E}[N_{P,r}] = \frac{|V_r|}{m}$. By Lemma 22(a), we have $\Pr[N_{P,r} > c \log n(\max\{\frac{|V_r|}{m}, 1\})] < \frac{1}{n^{O(1)}}$, for a sufficiently large c. By Lemma 24, with

high probability, there are at most $\alpha \log n$ copies of any vertex v in L_r, where α is a constant.

Let $\mathcal{F}_{P,r}$ denote the event that the total number of tasks assigned to processor P from level L_r is greater than $c' \cdot \max\{\frac{|V_r|}{m}, 1\} \cdot \log^2 n$, where c' is a constant. The above two arguments imply that

$$\Pr[\mathcal{F}_{P,r} > c' \max\{\frac{|V_r|}{m}, 1\} \log^2 n] < \frac{1}{n^\gamma},$$

where γ is a constant which can be made sufficiently large by choosing the value of c' appropriately. Let $\mathcal{F}$ denote the event that there exists a processor P and level L_r such that event $\mathcal{F}_{P,r}$ holds. By the union bound, we have

$$\Pr[\mathcal{F}] = \sum_{P,r} \Pr[\mathcal{F}_{P,r}] \leq \sum_{P,r} \frac{1}{n^\gamma} \leq \frac{n^2}{n^\gamma} \leq \frac{1}{n^{\gamma-2}},$$

where γ can be made suitably large. Hence, the lemma follows. ■

Lemma 26 *Let OPT denote the length of the optimal schedule. Schedule $\mathcal{S}$ has length $O(OPT \log^2 n)$ with high probability.*

Proof Let R be the number of levels in G. Lemma 25 implies that any level L_r has a processing time of $O(\max\{\frac{|V_r|}{m}, 1\} \log^2 n)$ with high probability. Hence, the total length of schedule S is at most

$$\sum_{r=1}^{R} O((\max\{\frac{|V_r|}{m}, 1\} \log^2 n)) \leq \sum_{r=1}^{R} O((\frac{|L_r|}{m} + 1) \log^2 n),$$

which is $O((\frac{nk}{m}+R) \log^2 n)$, where $R \leq k+D$. We observe that $OPT \geq \max\{\frac{nk}{m}, k, D\}$. Hence the length of schedule $\mathcal{S}$ is $O(OPT \log^2 n)$. ■

Theorem 27 *Algorithm 1 computes in polynomial time a schedule $\mathcal{S}$ which has an approximation guarantee of $O(\log^2 n)$ with high probability.*

Proof The approximation guarantee follows from Lemma 26. It is easy to see that the algorithm runs in time $O(k + kn^2 + n + mnk)$. Since $k = O(n)$ and $m = O(n)$, the algorithm runs in polynomial time of the input size n. ■

In a schedule produced by Algorithm 1, each layer in G is processed sequentially. This might result in the following scenario: there may be time instants t during which a processor P remains idle, even though there are ready tasks assigned to processor P. Clearly, idle times needlessly increase the makespan of the schedule. One way to eliminate idle times is to "compact" the schedule obtained through Algorithm 1. We now describe this approach in detail.

5.3.2 Random Delays with Compaction: A Priority based List Schedule

Motivated by the need to eliminate idle times from the schedule, we present Algorithm 2, which is called "Random Delays with Priorities". Algorithm 2 first defines a priority $\Gamma(v, i)$ for each task (v, i) and uses these priorities to create a schedule by list scheduling, as follows: at any given time t, for any processor P, among the set of all yet to be processed tasks which are ready and which are assigned to P, Algorithm 2 schedules the task with the least Γ value. It is easy to see that this algorithm results in a schedule such that there are no idle times. Let $\mathcal{S}'$ denote the schedule produced by this algorithm. The following theorem gives the performance guarantee and the running time of Algorithm 2.

Theorem 28 *Let $G(V, E)$ be an unstructured mesh, with $|V| = n$ and $D_1, \ldots D_k$ be the sweep directions. Let OPT be the length of the optimal schedule and m be the total*

Algorithm 2 Random Delays with Priorities

1: For all $i \in [1, \ldots, k]$, choose $X_i \in \{0, \ldots, k-1\}$ uniformly at random.

2: For each task (v, i), if it lies in level r in G_i, define $\Gamma(v, i) = r + X_i$. $\Gamma(v, i)$ is the priority for task (v, i).

3: For each vertex $v \in V$, choose a processor uniformly at random from $\{1, \ldots, m\}$.

4: $t = 1$.

5: **while** not all tasks have been processed **do**

6: **for all** processors $P = 1, \ldots, m$ **do**

7: (i) Let (v, i) be the task with lowest priority assigned to P (i.e., $\Gamma(v, i)$ is the smallest) that is ready to be processed (with ties broken arbitrarily).

8: (ii) Schedule (v, i) on P at time t.

9: **end for**

10: $t \leftarrow t + 1$.

11: **end while**

number of processors. Algorithm 2 runs in time $O((mk+nk)\log nk)$ *and produces an assignment of mesh elements to the* m *processors and a schedule* $\mathcal{S}'$ *whose makespan is at most* $O(OPT \log^2 n)$ *with high probability.*

Proof

Running time: To prove the claimed running time, we use a priority queue data structure which supports the operations: (i) Build Priority Queue, in $O(N \log N)$ time, where N is the total number of items, (ii) Find Min, in $O(1)$ time, (iii) Delete Min, in $O(1)$ time and (iv) Update Priority, in $O(\log N)$ time. Each item has a key, and the items are ordered based on this key. The Find Min operation returns the item with the smallest key value. In our case $N = mk$, since we have at most m edges in each DAG and a total of k distinct DAGs. For meshes arising in practice, $m = O(n)$.

To improve the efficiency, we define the key value of task (v, i) as $key(v, i) = \Gamma(v, i) + W \cdot indegree(v, i)$. Here, W is a large number (e.g. $10nk$), and $indegree(v, i)$ is the (current) number of immediate predecessors of task (v, i). As the schedule proceeds, $indegree(v, i)$ will reduce, and the key values will reduce. After the random delays are determined, $\Gamma(v, i)$, and therefore $key(v, i)$ is fixed for each task (v, i); we use these key values to construct a separate priority queue for each processor.

At each step, each processor looks at the task with smallest key value in its heap. If its indegree is 0, it performs it in the current step, and for each child w of this task, it reduces the indegree of w by 1; the key values of such tasks also need to be updated, using the Update Priority operation. Thus, whenever a task w is completed, it requires $O(outdegree(w) \log nk)$ time, which gives the bound in the lemma. Also, because of the definition of the key value, it follows that nodes of indegree i will have priorities much lower than nodes of priority $i + 1$, for any i. This ensures that only ready tasks are picked at any time.

Performance analysis: Recall the sets L_r defined in Algorithm 1. Let $\mathcal{S}$ and $\mathcal{S}'$ be the schedules produces by Algorithms 1 and 2 respectively. Let $t(v, i)$ and $t'(v, i)$ be the times at which task (v, i) got completed in the schedules $\mathcal{S}$ and $\mathcal{S}'$, respectively. We will show that for each r, $\max_{(v,i)\in L_r}\{t'(v, i)\} \leq \max_{(v,i)\in L_r}\{t(v, i)\}$. The claim then follows. Observe that for each $(v, i) \in L_r$, the priority $\Gamma(v, i) = r$. We now prove the above statement by induction on r.

Base Case: For $r = 1$, all nodes in L_1 have the lowest priority of 1, which is lower than the priority of any other node. Therefore, each processor P will schedule tasks in L_1, as long as there are any tasks in L_1 assigned to it. So, we have $\max_{(v,i)\in L_1}\{t'(v, i)\} \leq \max_{(v,i)\in L_1}\{t(v, i)\}$.

Induction hypothesis: Assume that the claim holds for all $r \leq l$.

Induction step: We now show the claim for $r = l + 1$. In schedule $\mathcal{S}$, the tasks in

L_{l+1} are started only after those in L_l are completed; let t denote the time when the last task in L_l got completed in $\mathcal{S}$. By the induction hypothesis, applied to L_l, all tasks in L_l are already completed in $\mathcal{S}'$, by this time. Also, the priority of any task in L_{l+1} is lower than that of any task in L_j for $j > l+1$. Therefore, in $\mathcal{S}'$, each processor P will first complete the tasks assigned to it in L_{l+1}, and only then would it pick tasks with higher $\Gamma()$ value. Therefore, $\max_{(v,i)\in L_r}\{t'(v,i)\} \leq \max_{(v,i)\in L_r}\{t(v,i)\}$. ■

5.3.3 An Improved $O(\log m \log\log\log m)$-Approximation

We now show that a slight modification of the earlier algorithm, along with a more careful analysis leads to a $O(\log m \log\log\log m)$-approximation of the makespan. The new algorithm is called "Improved Random Delay" and is presented in Algorithm 3. In contrast with Theorem 27, which shows a high probability bound, we will only bound the expected length of the schedule. The basic intuition for the improved analysis comes from corollary 31 below: if we consider the standard "balls-in-bins" experiment, the maximum number of balls in any bin is at most the average, plus a logarithmic quantity. The idea now is to consider the scheduling of each layer in the combined DAG as such an experiment. One complication comes from the dependencies - the events that tasks (v,i) and (w,i) end up in the same layer in the combined DAG are not independent, as a result of which a lot of tasks from some direction could be assigned to the same layer of the combined DAG. The new algorithm handles this problem by the pre-processing step, which is the essential difference between this and the previous random delay algorithms. The pre-processing step transforms the original instance, so that there are at most m tasks in each layer in each direction, and also guarantees that, in expectation, at most m tasks are assigned to each layer of the combined DAG.

Analysis. For the tighter analysis, we need to look at the time taken to process all the tasks in any layer L''_t. Let Y_t denote the time required to process the tasks in L''_t.

Algorithm 3 Improved Random Delay

1: **Preprocessing:** Construct a new set of levels L'_i for each direction i in the following manner.

- First construct a new DAG $H(\cup_i V_i, \cup_i E_i)$ by combining all the G_i's, and viewing all the copies (v, i) of a vertex v as distinct.
- Run the standard greedy list scheduling algorithm on H with m identical parallel machines [51]; let T be the makespan of this schedule.
- Let $L'_{ij} = \{(v, i) \in V_i | (v, i)$ done at step j of above schedule$\}$.

2: For all $i \in [1, \ldots, k]$, choose $X_i \in \{0, \ldots, k-1\}$ uniformly at random.

3: Form a combined DAG G'' as follows: $\forall r \in \{1, \ldots, T + k - 1\}$, define $L''_r = \bigcup_{\{i:X_i<r\}} L'_{i,r-X_i}$. The edge $((u, i), (v, i)))$ is present in G'', if and only if there exists an edge $((u, i), (v, i))$ in G_i.

4: For each vertex $v \in V$, choose a processor uniformly at random from $\{1, \ldots, m\}$.

5: Construct a schedule by processing layers $L''_1, L''_2, \ldots$ sequentially in that order:

- Layer L''_{r+1} is processed only after all tasks in L''_r have been processed.
- Within each layer L''_r, process the tasks assigned to each processor in any arbitrary order.

Our main result will be the following.

Theorem 29 *For any t, $\mathbf{E}[Y_t] \leq O(\mu_t/m + (\log m)\log\log\log m)$, where $\mu_t = E[|L''_t|]$.*

Let $\rho = (\log m)\log\log\log m$. Theorem 29 implies that we get an $O(\rho)$–approximation in expectation, by observing that the makespan T after the preprocessing step is within a small factor of the optimal.

Corollary 30 *Algorithm 3 gives a schedule of expected length $O(\rho)$ times the optimal.*

Proof From the analysis in [51], it follows that $T \leq 2OPT$; therefore, $OPT = \Omega(nk/m + T + k)$. Next, $\sum_t |L''_t| = nk$ and thus, $\sum_t \mu_t = nk$. Summing the bound of Theorem 29 over all t, we get that the expected final makespan is $O(nk/m + (T+k)\rho)$, which gives an $O(\rho)$–approximation. ■

We start with some observations on the expected maximum load in a balls–in–bins experiment. Motivated by Lemma 22, we define a function $H(\mu, p)$, for $\mu > 0$ and $p \in (0,1)$ as follows; the constant C will be chosen large enough.

$$H(\mu,p) = \begin{cases} C \cdot \frac{\ln(p^{-1})}{\ln(\ln(p^{-1})/\mu)} & \text{if } \mu \leq \ln(p^{-1})/e; \\ Ce\mu & \text{otherwise.} \end{cases} \tag{5.4}$$

Note that for any fixed p, H is continuous and has a valid first derivative for all values of μ – to see this, we just need to check these conditions for $\mu = \ln(p^{-1})/e$.

Corollary 31 *(a) If we fix p, then $H(\mu, p)$ is a concave function of μ. (b) Suppose the constant C in the definition of H is chosen large enough. Then, if we assign some number t of objects at random to m bins, the expected maximum load on any bin is at most $H(t/m, 1/m^2) + t/m$.*

Proof (a). Fix p. The second derivative of $H(\mu, p)$ w.r.t. μ, can be seen to be non-positive when $\mu \leq \ln(p^{-1})/e$; thus, H is concave in this region. Since H is linear

for larger μ, it is trivially concave in this region also. We see that H is concave in general, by noting as above that H is continuous and differentiable at $\mu = \ln(p^{-1})/e$. (b) Consider any machine i; the load X_i on it is a sum of i.i.d. indicator random variables, and $\mathbf{E}[X_i] = t/m$. Now, it is easy to verify that if C is large enough, then $F(\mu, p) \leq H(\mu, p)$. Thus, letting $\mathcal{E}_i$ be the event "$X_i \geq H(t/m, 1/m^2)$", Lemma 22(b) shows that $\Pr[\mathcal{E}_i] \leq 1/m^2$; so, $\Pr[\exists i : \mathcal{E}_i] \leq 1/m$. If the event "$\exists i : \mathcal{E}_i$" does not hold, then the maximum load on any machine is at most $H(t/m, 1/m^2)$ with probability 1; else if "$\exists i : \mathcal{E}_i$" is true, then the maximum load on any machine is at most t with probability 1. Therefore, the expected maximum load is at most $H(t/m, 1/m^2) + (1/m) \cdot t$. ■

Lemma 32 *For any constant $a \geq 3$, the function $\phi_a(x) = x^a e^{-x}$ is convex in the range $0 \leq x \leq 1$.*

Proof It can be verified that the second derivative of ϕ_a satisfies $\phi_a''(x) = x^{a-2}e^{-x}((a-x)^2 - a)$, which in turn is at least $x^{a-2}e^{-x}((a-1)^2 - a)$, for the given range of x and a. Since $(a-1)^2 - a \geq 0$ for $a \geq 3$, the lemma follows. ■

Proof of Theorem 29: Fix t arbitrarily. For $j \geq 0$, let $Z_j = \{v| \ |\{(v,i) \in L_t''\}| \in [2^j, 2^{j+1})\}$, i.e., Z_j is the set of nodes v such that the number of copies of v that end up in layer L_t'' lies in the range $[2^j, 2^{j+1})$. We first present some useful bounds on $\mathbf{E}[|Z_j|]$ and on μ_t.

Lemma 33 *(a) $\sum_{j \geq 0} 2^j \mathbf{E}[|Z_j|] \leq \mu_t$; and (b) $\mu_t \leq m$.*

Proof Part (a) follows by the definitions of Z_j and μ_t, and from the linearity of expectation. For part (b), note first that a node $(v,i) \in L_{ij}'$ can get assigned to layer L_t'' only if $t - k + 1 \leq j \leq t$. By the preprocessing step, $|\cup_i L_{ij}'| \leq m$, for each j, and therefore, the number of such nodes (v,i) assigned to L_t'' is at most mk. For

each such node (v, i), the probability of getting assigned to layer L''_t is $1/k$, since X_i is chosen uniformly random in the range $0, \ldots, k-1$. Thus, $\mu_t \leq m$. ■

Lemma 34 *For $j \geq 2$, $\mathbf{E}[Z_j] \leq (e/2^j)^{2^j} \cdot \mu_t$.*

Proof Fix $j \geq 2$, and let $a = 2^j$. Let N_v be the random variable denoting the number of copies of job $v \in V$ that get assigned to layer L''_t; letting $b_v = \mathbf{E}[N_v]$, we also have $b_v \leq 1$. Furthermore, $\mu_t = \sum_v b_v$ and $\mathbf{E}[Z_j] \leq \sum_v \Pr[N_v \geq a]$.

Now, Lemma 22(a) yields $\Pr[N_v \geq 2^j] \leq (e/a)^a \cdot b_v^a e^{-b_v}$, and so

$$\mathbf{E}[Z_j] \leq \sum_v (e/a)^a \cdot b_v^a e^{-b_v}. \tag{5.5}$$

Now, $(e/a)^a \cdot b_v^a e^{-b_v}$ is a convex function function of b_v (for fixed j), by Lemma 32. Thus we get, for any fixed value of μ_t:

- if $\mu_t < 1$, then the r.h.s. of (5.5) is maximized when $b_v = \mu_t$ for some v, and $b_w = 0$ for all other w; so, in this case, $\mathbf{E}[Z_j] \leq (e/a)^a \cdot {\mu_t}^a e^{-\mu_t}$.
- if $\mu_t \geq 1$, then the r.h.s. of (5.5) is at most what it would be, if we had $\lceil \mu_t \rceil$ indices v with $b_v = 1$, with $b_w = 0$ for all other w; so, in this case, $\mathbf{E}[Z_j] \leq (2\mu_t) \cdot (e/a)^a \cdot e^{-1}$.

This yields the lemma. ■

Now, consider step (3) of Algorithm 3, and fix Z_j for some time t. Next, schedule the jobs in Z_j in the following manner in step (5) of the algorithm: we first run all nodes in Z_0 to completion, then run all nodes in Z_1 to completion, then run all nodes in Z_2 to completion, and so on. Clearly, our actual algorithm does no worse than this. Recall that we condition on some given values Z_j. We now bound the expected time to process all jobs in Z_j, in two different ways (this expectation is only w.r.t. the random choices made by P_1): (a) first, by Corollary 31, this expectation

is at most $2^{j+1} \cdot (H(|Z_j|/m, 1/m^2) + |Z_j|/m)$; and (b) trivially, this expectation is at most $2^{j+1} \cdot |Z_j|$. Thus, conditional on the values Z_j, the expected makespan for level t is:

$$\begin{aligned}
\mathbf{E}[Y_t \mid (Z_0, Z_1, \ldots)] &\leq [\sum_{j=0}^{\ln\ln m} 2^{j+1} \cdot (H(|Z_j|/m, 1/m^2) \\
&+ |Z_j|/m)] + [\sum_{j>\ln\ln m} 2^{j+1} \cdot |Z_j|] \\
&\leq [\sum_{j=0}^{\ln\ln m} 2^{j+1} \cdot (\mathbf{E}[H(|Z_j|/m, 1/m^2)] + \mathbf{E}[|Z_j|]/m)] \\
&+ [\sum_{j>\ln\ln m} 2^{j+1} \cdot \mathbf{E}[|Z_j|]] \\
&\leq [\sum_{j=0}^{\ln\ln m} 2^{j+1} \cdot (H(\mathbf{E}[|Z_j|]/m, 1/m^2) + \mathbf{E}[|Z_j|]/m)] \\
&+ [\sum_{j>\ln\ln m} 2^{j+1} \cdot \mathbf{E}[|Z_j|]]
\end{aligned}$$

This follows since H is concave by Corollary 31(a). (We are using Jensen's inequality: for any concave function f of a random variable T, $\mathbf{E}[f(T)] \leq f(\mathbf{E}[T])$.) Consider the first sum in the last inequality above. By Lemma 33(a), the term "$\sum_{j=0}^{\ln\ln m} 2^{j+1} \cdot \mathbf{E}[|Z_j|]/m$" is $O(\mu_t/m)$. Next, we can see from (5.4) that if p is fixed, then $H(\mu, p)$ is a non-decreasing function of μ. So, Lemmas 33 and 34 show that there is a value $\alpha \leq O((\ln m)/\ln\ln m)$ such that $H(\mathbf{E}[|Z_j|]/m, 1/m^2) \leq \alpha$ for $j = 0, 1$.

Hence,

$$\begin{aligned}
\textstyle\sum_{j=0}^{\ln\ln m} & \; 2^{j+1} \cdot H(\mathbf{E}[|Z_j|]/m, 1/m^2) \\
\leq & \quad O(\alpha) + \sum_{j=2}^{\ln\ln m} 2^{j+1} \cdot H(\mathbf{E}[|Z_j|]/m, 1/m^2) \\
\leq & \quad O(\alpha) + \sum_{j=2}^{\ln\ln m} 2^{j+1} \cdot H((e/2^j)^{2^j}, 1/m^2) \\
& \quad \text{(by Lemmas 33(b) and 34)} \\
\leq & \quad O(\alpha) + O\left(\sum_{j=2}^{\ln\ln m} 2^{j+1} \cdot \frac{\ln m}{\ln\ln m + j2^j} \right). \qquad (5.6)
\end{aligned}$$

The second inequality above follows from Lemmas 33(b) and 34. We split this sum into two parts. As long as $2^j \leq \ln\ln m / \ln\ln\ln m$, the term "$\frac{\ln m}{\ln\ln m + j2^j}$" above is $\Theta(\frac{\ln m}{\ln\ln m})$; for larger j, it is $\Theta(\frac{\ln m}{j2^j})$. Thus, the sum in the first part is dominated by its last term, and hence equals $O((\log m)/\log\log\log m)$. The sum in the second part is bounded by

$$O\left(\sum_{j=2}^{\ln\ln m} (\ln m)/j \right) = O((\log m) \cdot \log\log\log m).$$

Summarizing, the first sum above is $O(\mu_t/m + (\log m)\log\log\log m)$. Now consider the second sum above. Recalling Lemma 34, we get

$$\sum_{j > \ln\ln m} 2^{j+1} \cdot \mathbf{E}[|Z_j|] \leq \mu_t \cdot \sum_{j > \ln\ln m} 2^{j+1} \cdot (e/2^j)^{2^j} = O(\mu_t/m), \qquad (5.7)$$

since the second sum in the earlier expression for $E[Y_t]$ is basically dominated by its first term. This completes the proof of Theorem 29.

5.3.4 Communication cost

We briefly discuss the problem of bounding the makespan for the GSS problem subject to inter-processor communication latencies. We call this problem GSS-ICD - the GSS

problem with inter-processor communication delays. As mentioned earlier, we use an extension of the model proposed by Hwang et al. [60]. Their model is an extension of the well known model of Rayward-Smith [110]. In the extended model, we are given an instance of GSS as before. The jobs are required to be processed on a parallel machine with m processors as before. The crucial difference now is that we now have two additional costs: each edge (v, w) of the DAG G has an associated weight $\eta(v, w)$ denoting the size of message sent by job v to w upon completion of V_i. Secondly, the m processors are joined together in form of a network and there is parameter $\tau(p_i, p_j)$ denoting the delay in sending a message from p_i to p_j. Thus if v is assigned to processor p_l and w is assigned to processor p_k and w is an immediate successor of v, then the w has to wait an additional $\eta(v, w) \times \tau(p_l, p_k)$ time after v is completed before it can be considered for processing.

We now briefly describe how we can extend the $O(\log m \log\log\log m)$ bound of Section 5.3.3 to the communication latency model of [110, 96, 60]. Let $C_{\max} = \max_{v,w,l,k}\{\eta(v, w) \times \tau(p_l, p_k)\}$ denote the maximum communication latency. Note that the schedule $\mathcal{S}$ computed by Algorithm 3 processes the layers $L''_1, L''_2, \ldots$ sequentially. We dilate the schedule $\mathcal{S}$ by an $O(C_{\max})$ factor in the following manner: for each t, after layer L''_t has been completely processed, we wait for $C_{\max}$ steps for all communication to be completed, and then start layer L''_{t+1}. Denote the modified algorithm as Algorithm 4. By Corollary 30, we get the following result.

Corollary 35 *Let OPT denote the length of the optimal schedule for the GSS-ICD problem. Then Algorithm 4 yields a schedule of expected length $O(C_{\max} \cdot \log m \log\log\log m \cdot OPT)$.*

Chapter 6

Tree Scheduling Algorithms

6.1 Introduction

A very general type of scheduling problem involves unrelated parallel machines and precedence constraints, i.e., we are given: (i) a set of n jobs with precedence constraints that induce a partial order on the jobs; (ii) a set of m machines, each of which can process at most one job at any time, and (iii) an arbitrary set of integer values $\{p_{i,j}\}$, where $p_{i,j}$ denotes the time to process job j on machine i. Thus, we need to decide which machine to schedule each job on, and then run the jobs in some order consistent with the precedence constraints. Let C_j denote the completion time of job j. Subject to the above constraints, two commonly studied versions are (i) minimize the *makespan*, or the maximum time any job takes, i.e. $\max_j\{C_j\}$ - this is denoted by $R|prec|C_{max}$, and (ii) minimize the weighted completion time - this is denoted by $R|prec|\sum_j w_jC_j$. Numerous other variants, involving release dates or other objectives have been studied (see e.g. [55]); most such variants are NP-hard.

Almost-optimal upper and lower bounds on the approximation ratio are known for the versions of the above problems without precedence constraints (i.e., the $R||C_{max}$ and $R||\sum_j w_jC_j$ problems) [29, 84, 119]. In Chapter 3, we presented multi-

criteria algorithms for unrelated parallel scheduling for simultaneously optimizing makespan, weighted completion time, and ℓ_p norms of the machine loads – in the absence of precedences. However, very little is known in the presence of precedence constraints. The only case of the general $R|prec|C_{max}$ problem for which non-trivial approximations are known is the case where the precedence constraints are a collection of node-disjoint chains - this is the job shop scheduling problem [118], which itself has a long history. The first result for job shop scheduling was the breakthrough work of Leighton et al. [83, 82] for packet scheduling, which implied a logarithmic approximation for the case of unit processing costs. Leighton et al. [83, 82] introduced the "random delays" technique, and almost all the results on the job shop scheduling problem are based on variants of this technique. The result of [83, 82] was generalized to nonuniform processing costs by Shmoys et al. [118], who obtained an approximation factor of $O(\log(m\mu)\log(\min\{m\mu, p_{\max}\})/\log\log(m\mu))$, where $p_{\max}$ is the maximum processing time of any job, and μ is the maximum length of any chain in the given precedence constraints. These bounds were improved by an additional $\log\log(m\mu)$ factor by Goldberg et al. [49]; see [43] for additional relevant work. Shmoys et al. [118] also generalize job-shop scheduling to DAG-shop scheduling, where the operations of each job form a DAG, instead of a chain, with the additional constraint that *the operations within a job can be done only one at a time.* They show how the results for the case of a chain extend to this case also.

The only results known for the case of arbitrary number of processors (i.e., machines) with more general precedence constraints are for identical parallel machines (denoted by $P|prec|C_{max}$) [55], or for uniformly-related parallel machines (denoted by $Q|prec|C_{max}$) [31, 28]. The weighted completion time objective has also been studied for these variants [29, 31, 56]. When the number of machines is constant, polynomial-time approximation schemes are known [62, 64]. Note that all the discussion here relates to *non-preemptive* schedules, i.e., once the processing of a job is

started, it cannot be stopped until it is completely processed; preemptive variants of these problems have also been well studied (see e.g. [115]). Less is known for the weighted completion time objective in the same setting, as compared to the makespan. The known approximations are either for the case of no precedence constraints [119], or for precedence constraints with parallel/related processors [31, 56, 105]. To the best of our knowledge, no non-trivial bound is known on the weighted completion time on unrelated machines, in the presence of precedence constraints of *any kind*.

Here, motivated by applications such as evaluating large expression-trees and tree-shaped parallel processes, we consider the special case of the $R|prec|C_{max}$ and $R|prec|\sum_j w_j C_j$ problems, where the precedences form a forest, i.e., the undirected graph underlying the precedences is a forest. Thus, this naturally generalizes the job shop scheduling problem, where the precedence constraints form a collection of disjoint chains.

Summary of results. We present the first polylogarithmic approximation algorithms for the $R|prec|C_{max}$ and $R|prec|\sum_j w_j C_j$ problems, under "treelike" precedences. Since most of our results hold in the cases where the precedences form a forest (i.e., the undirected graph underlying the DAG is a forest), we will denote the problems by $R|forest|C_{max}$, and $R|forest|\sum_j w_j C_j$, respectively, to simplify the description - this generalizes the notation used by [63] for the case of chains.

(a). The $R|forest|C_{max}$ problem. We obtain a polylogarithmic approximation for this problem. We employ the same lower bound LB (described shortly) used in [83, 82, 118, 49, 43], except that we are dealing with the more general situation where jobs have not yet been assigned to machines. Given an assignment of jobs to machines, let $P_{\max}$ denote the maximum processing time along any directed path, and $\Pi_{\max}$ be the maximum processing time needed on any machine. It is immediate that given such an assignment, $\max\{P_{\max}, \Pi_{\max}\}$ is a lower bound on the makespan of any schedule. Let LB denote the minimum possible value of $\max\{P_{\max}, \Pi_{\max}\}$,

taken over all possible legal assignments of jobs to machines; LB is thus a lower bound on the makespan. Let $p_{max} = \max_{i,j} p_{i,j}$ be the maximum processing time of any job on any machine. We obtain an $O(\frac{\log^2 n}{\log\log n}\lceil\frac{\log\min(p_{\max},n)}{\log\log n}\rceil)$ approximation to the $R|forest|C_{max}$ problem. When the forests are out-trees or in-trees, we show that this polylogarithmic factor can be improved to $O(\log n \cdot \lceil\log(\min\{p_{\max}, n\})/\log\log n\rceil)$; for the special case of unit processing times, this actually becomes $O(\log n)$. We also show that the lower-bound LB cannot be put to much better use, even in the case of trees - for unit processing costs, we show instances whose optimal schedule is $\Omega(LB \cdot \log n)$.

Our algorithm for solving $R|forest|C_{max}$ follows the overall approach used to solve the job shop scheduling problem (see, e.g. [118]) and involves two steps: (i) we show how to compute a processor assignment whose LB value is within a $(\frac{3+\sqrt{5}}{2})$–factor of optimal, by extending the approach of [84], and (ii) design a poly-logarithmic approximation algorithm for the resulting variant of the $R|prec|C_{max}$ problem with pre-specified processor assignment and forest-shaped precedences.

We call the variant of the $R|prec|C_{max}$ problem arising in step (ii) above (i.e., when the processor assignment is pre-specified), the *Generalized DAG-Shop Scheduling* or the GDSS problem, for brevity. Note that the job shop scheduling problem is a special case of GDSS, and this problem is different from the Dagshop scheduling problem defined by [118].[1] Our algorithm for treelike instances of GDSS is similar to one used in [83, 118, 49], namely injecting random delays to the start times of the jobs; this allows for contention resolution. However, unlike [83, 118, 49], it is not adequate to insert random delays only at the head of the trees - we actually insert random delays throughout the schedule. Our algorithm partitions the forest into blocks of chains suitably, and the problem restricted to a block of chains is simply a job shop problem; also, the decomposition guarantees that the solutions to these job

[1] In the Dagshop problem [118], the input is a collection of DAGs, but in each DAG, at most one operation can be done at a time.

shop problems can be pasted together to get a complete schedule - this immediately gives us a reduction from the $R|forest|C_{max}$ problem to the job shop problem, with the quality depending on the number of blocks. We can remove a logarithmic factor when the DAG is an in-/out-tree, by a different analysis, which does not reduce this problem to a collection of job shop problems. As in the original approach of [83], we bound the contention by a Chernoff bound. However, the events we need to consider are not independent, and we need to exploit the variant of this bound from [98] that works in the presence of correlations.

(b). The $R|forest|\sum_j w_jC_j$ problem. We show a reduction from $R|prec|\sum_j w_jC_j$ to $R|prec|C_{\max}$ of the following form: if there is a schedule of makespan $(P_{\max}+\Pi_{\max})\cdot\rho$ for the latter, then there is an $O(\rho)$-approximation algorithm for the former. We exploit this, along with the fact that our approximation guarantee for $R|forest|C_{max}$ is of the form "$(P_{\max}+\Pi_{\max})$ times polylog", to get a polylogarithmic approximation for the $R|forest|\sum_j w_jC_j$ problem. Our reduction is similar in spirit to that of [31, 105]: using geometric time windows and appropriate linear constraints. We employ additional ideas here in order to handle our specific situation (e.g., the reduction in [105] is meant for *identical* parallel machines while ours is for *unrelated* machines).

(c). Minimizing weighted flow time on chains. Given a "release time" for each job (the time at which it enters the system) and a schedule, the *flow time* of a job is the time elapsed from its release time to its completion time. Minimizing the weighted flow time of the jobs is a notoriously hard problem, and no reasonable approximation algorithm is known even for the special case of job-shop scheduling. We consider the case of this problem where (i) the forest is a collection of node-disjoint chains, (ii) for each machine i and operation v, $p_{i,v} \in \{p_v, \infty\}$ (i.e., the *restricted-assignment variant*), and (iii) all processing times p_v are polynomially-bounded in the input length N. (Note that job shop scheduling, where we have a collection of node-disjoint chains and where the jobs are pre-assigned to machines, is

a special case of what we consider; however, we also assume that the processing times are polynomially-bounded.) We describe a natural LP-relaxation and a dependent randomized rounding scheme for this problem. Our rounding ensures that (i) the precedence constraints are satisfied with *probability* 1, and (ii) for any (v, t), the probability of starting v at time t equals its fractional (LP) value $z_{v,t}$. This result also leads to a bicriteria $(1 + o(1))$–approximation for the weighted flow time, using $O(\log N / \log \log N)$ copies of each machine.

6.2 The $R|forest|C_{max}$ problem

We now present approximation algorithms for the $R|forest|C_{max}$ problem, and also study the limitations of our approach. In the description below, we will use the terms "node" and "job" interchangeably; we will not use the term "operation" to refer to nodes of a DAG, because we do not have the job shop or dag shop constraints that at most one node in a DAG can be processed at a time. Our algorithm for the $R|forest|C_{max}$ problem constructs a schedule whose makespan is to within a guaranteed factor times the lower bound $\max\{P_{\max}, \Pi_{\max}\}$; we then show in Section 6.2.3 that this lower bound is not very good for general (i.e., non-forest-shaped) DAGs.

Our algorithm for the $R|forest|C_{max}$ problem involves the following two steps:

Step 1: Construct a processor assignment for which the value of $\max\{P_{\max}, \Pi_{\max}\}$ is within a constant factor $((3 + \sqrt{5})/2)$ of the smallest-possible. This is described in Section 6.2.1.

Step 2: Solve the GDSS problem we get from the previous step to get a schedule of length polylogarithmically more than $\max\{P_{\max}, \Pi_{\max}\}$. This is described in Section 6.2.2.

6.2.1 Step 1: A constant-factor processor assignment

We now describe the algorithm for processor assignment, using some of the ideas from [84]. Let T be our "guess" for the optimal value of $LB = \max\{P_{\max}, \Pi_{\max}\}$. Let J and M denote the set of jobs and machines, respectively. Let x denote any fractional processor assignment, i.e., the non-negative value $x_{i,j}$ is the fraction of job j assigned to machine i; we have for all j that $\sum_i x_{i,j} = 1$. As mentioned before, P_{max} denotes the maximum processing time along any directed path, i.e., $P_{max} = \max_{path\ P}\{\sum_{j\in P}\sum_i x_{i,j}p_{i,j}\}$. Also, Π_{max} denotes the maximum load on any machine, i.e., $\Pi_{max} = \max_i\{\sum_j x_{i,j}p_{i,j}\}$. We now define a family of linear programs $LP(T)$, one for each value of $T \in \mathbf{Z}^+$, as follows:

$$\begin{aligned}
\forall j \in J,\ \sum_i x_{ij} &= 1 && (6.1)\\
\forall i \in M,\ \sum_j x_{ij}p_{ij} &\le T && (6.2)\\
\forall j \in J,\ z_j &= \sum_i p_{ij}x_{ij} && (6.3)\\
\forall (j' \prec j)\ c_j &\ge c_{j'} + z_j && (6.4)\\
\forall j \in J,\ c_j &\le T && (6.5)\\
\forall (i,j),\ (p_{i,j} > T) &\implies x_{i,j} = 0 && (6.6)\\
\forall (i,j),\ x_{i,j} &\ge 0\\
\forall j,\ c_j &\ge 0
\end{aligned}$$

The constraints (6.1) ensure that each job is assigned a machine, and (6.2) ensures that the maximum fractional load on any machine ($\Pi_{\max}$) is at most T. Constraints (6.3) define the fractional processing time z_j for a job j, and (6.4) captures the precedence constraints amongst jobs (c_j denotes the fractional completion of time of job j). We note that $\max_j c_j$ is the fractional P_{max}. Constraints (6.5) state that the fractional $P_{\max}$ value is at most T, and those of (6.6) are the valid constraints that

if it takes more than T steps to process job j on machine i, then j should not be scheduled on i.

Let T^* be the smallest value of T for which $LP(T)$ has a feasible solution. It is easy to see that T^* is a lower bound on LB. We now present a rounding scheme which rounds a feasible fractional solution to $LP(T^*)$ to an integral solution. Let X_{ij} denote the indicator variable which denotes if job j was assigned to machine i in the integral solution, and let C_j be the integer analog of c_j. We first modify the x_{ij} values using *filtering* [86]. Let $K_1 = \frac{3+\sqrt{5}}{2}$. For any (i,j), if $p_{ij} > K_1 z_j$, then set x_{ij} to zero. This step could result in a situation where, for a job j, the fractional assignment $\sum_i x_{ij}$ drops to a value r such that $r \in [1 - \frac{1}{K_1}, 1)$. So, we scale the (modified) values of x_{ij} by a factor of at most $K_2 = \frac{K_1}{K_1 - 1}$. Let $\mathcal{A}$ denote this fractional solution. Crucially, we note that any rounding of $\mathcal{A}$, which ensures that only non-zero variables in $\mathcal{A}$ are set to non-zero values in the integral solution, has an integral $P_{\max}$ value which is at most $K_1 T^*$. This follows from the fact that if $X_{ij} = 1$ in the rounded solution, then $p_{ij} \leq K_1 z_j$. Hence, it is easy to see that by induction, for any job j, C_j is at most $K_1 c_j \leq K_1 T^*$.

We now show how to round $\mathcal{A}$. Recall that [84] presents a rounding algorithm in the "unrelated parallel machines and *no* precedence constraints" context with the following guarantee: if the input fractional solution has a fractional $\Pi_{\max}$ value of α, then the output integral solution has an integral $\Pi_{\max}$ value of at most $\alpha + \max_{(i,j):\ x_{ij}>0} p_{ij}$. We use $\mathcal{A}$ as the input instance for the rounding algorithm in [84]. Note that $\mathcal{A}$ has a fractional $\Pi_{\max}$ value of at most $K_2 T^*$. Further, $\max_{(i,j):\ x_{ij}>0} p_{ij} \leq T^*$ by (6.6). Thus, the algorithm of [84] yields an integral solution I whose $P_{\max}$ value is at most $K_1 T^*$, and whose $\Pi_{\max}$ value is at most $(K_2 + 1)T^*$. Observe that setting $K_1 = \frac{3+\sqrt{5}}{2}$ results in $K_1 = K_2 + 1$. Finally, we note that the optimal value of T can be arrived at by a bisection search in the range $[0, np_{\max}]$, where $n = |J|$ and $p_{\max} = \max_{i,j} p_{ij}$. Since T^* is a lower bound on LB, we have the following result.

Theorem 36 *Given an arbitrary (not necessarily forest-shaped) DAG, the above algorithm computes a processor assignment for each job in which the value of* $\max\{P_{\max}, \Pi_{\max}\}$ *is within a* $(\frac{3+\sqrt{5}}{2})$*–factor of the optimal.*

6.2.2 Step 2: Solving the GDSS problem under treelike precedences

We can now assume that the assignment of jobs to machines is given. We first consider the case when the precedences are a collection of directed in-trees or out-trees in Section 6.2.2. We then extend this to the case where the precedences form an arbitrary forest (i.e., the underlying undirected graph is a forest) in Section 6.2.2. We will use the notation $m(v)$ to denote the machine to which node v is assigned, and the processing time for node v will be denoted by p_v.

GDSS on Out-/In-Arborescences

An out-tree is a tree rooted at some node, say r, with all edges directed away from r; an in-tree is a tree obtained by reversing the directions of all the arcs in an out-tree. In the discussion below in Section 6.2.2, we only focus on out-trees; the same results can be obtained similarly for in-trees.

We will need Fact 37, a generalization of the Chernoff bound from [98]. Note that the *ordering* of the X_i is important in Fact 37; we make a careful choice of such an ordering in the proof of Lemma 39.

Fact 37 ***([98])*** *Let* $X_1, X_2, \ldots, X_l \in \{0,1\}$ *be random variables such that for all* i*, and for any* $S \subseteq \{X_1, \ldots, X_{i-1}\}$*,* $\Pr[X_i = 1 | \bigwedge_{j \in S} X_j = 1] \leq q_i$*. (In particular,* $\Pr[X_i = 1] \leq q_i$*.) Let* $X \doteq \sum_i X_i$*; note that* $\mathbf{E}[X] \leq \sum_i q_i$*. Then for any* $\delta > 0$*,* $\Pr[X \geq (1+\delta) \cdot \sum_i q_i] \leq (e^{\delta}/(1+\delta)^{1+\delta})^{\sum_i q_i}$.

Our algorithm for out-trees requires a careful partitioning of the tree into blocks

of chains, and giving random delays at the start of each chain in each of the blocks - thus the delays are spread all over the tree. The head of the chain waits for all its ancestors to finish running, after which it waits for an amount of time equal to its random delay. After this, the entire chain is allowed to run without interruption. Of course, this may result in an infeasible schedule where multiple jobs simultaneously contend for the same machine (at the same time). We show that this contention is low and can be resolved by expanding the infeasible schedule produced above.

Chain Decomposition. We define the notions of *chain decomposition* of a graph and its *chain width*. Given a DAG $G(V, E)$, let $d_{in}(u)$ and $d_{out}(u)$ denote the in-degree and out-degree, respectively, of u in G. A *chain decomposition* of $G(V, E)$ is a partition of its vertex set into subsets $B_1, \ldots, B_\lambda$ (called blocks) such that the following properties hold:

(P1) The subgraph induced by each block B_i is a collection of vertex-disjoint directed chains, i.e., the in-degree and out-degree of each node in the induced subgraph is at most one (and there are of course no cycles); and

(P2) for any $u, v \in V$, let $u \in B_i$ be an ancestor of $v \in B_j$. Then, either $i < j$, or $i = j$ and u and v belong to the same directed chain of B_i.

The *chain-width* of a DAG is the minimum value λ such that there is a chain decomposition of the DAG into λ blocks. (Such a decomposition always exists: trivially, we could take each block to be a singleton vertex. We also note that the notions of chain decomposition and chain-width are similar to those of caterpillar decomposition and caterpillar dimension for trees [87]. However, in general, a caterpillar decomposition need not be a chain-decomposition and vice-versa.)

Well-structured schedules. We now state some definitions motivated by those in [49]. Given a GDSS instance with a DAG $G(V, E)$ and given a chain decomposition of G into λ blocks, we construct a *B-delayed schedule* for it as follows; B is an integer that will be chosen later. Each job v which is the head of a chain in a block is assigned

a delay $d(v)$ in $\{0, 1, \ldots, B-1\}$. Let v belong to the chain C_i. Job v waits for $d(v)$ amount of time after all its predecessors have finished running, after which the jobs of C_i are scheduled consecutively (of course, the resulting schedule might be infeasible). A *random B-delayed schedule* is a B-delayed schedule in which all the delays have been chosen independently and uniformly at random from $\{0, 1, \ldots, B-1\}$. For a B-delayed schedule S, the *contention* $C(M_i, t)$ is the number of jobs scheduled on machine M_i in the time interval $[t, t+1)$. As in [49, 118], we assume w.l.o.g. that all job lengths are powers of two. This can be achieved by multiplying each job length by at most a factor of two (which affects our approximation ratios only by a constant factor). A delayed scheduled S is *well-structured* if for each k, all jobs with length 2^k begin in S at a time instant that is an integral multiple of 2^k. Such schedules can be constructed from randomly delayed schedules as follows. First create a new GDSS instance by replacing each job $v = (m(v), p_v)$ by the job $v = (m(v), 2p_v)$. Let S be a random B-delayed schedule for this modified instance, for some B; we call S a *padded random B-delayed schedule*. From S, we can construct a well-structured delayed schedule, S', for the original GDSS instance as follows: insert v with the correct boundary in the slot assigned to $\hat{v}$ by S. S' will be called a *well-structured random B-delayed schedule* for the original GDSS instance.

Our algorithm. We now describe our algorithm; for the sake of clarity, we occasionally omit floor and ceiling symbols (e.g., "$B = \lceil 2\Pi_{\max}/\log(np_{\max})\rceil$" is written as "$B = 2\Pi_{\max}/\log(np_{\max})$"). As before let $p_{\max} = \max_v p_v$.

1. Construct a chain decomposition of the DAG $G(V, E)$ and let λ be its chain width.

2. Let $B = 2\Pi_{\max}/\log(np_{\max})$. Construct a padded random B-delayed schedule S by first increasing the processing time of each job v by a factor of 2 (as described above), and then choosing a delay $d(v) \in \{0, \ldots, B-1\}$ independently and uniformly at random for each job v which is the head of its chain in a block.

3. Construct a well-structured random B-delayed schedule S' as described above.

4. Construct a valid schedule S'' using the technique from [49] as follows:

   (a) Let the makespan of S' be L.

   (b) Partition the schedule S' into *frames* of length $p_{\max}$; i.e., into the set of time-intervals $\{[ip_{\max}, (i+1)p_{\max}),\ i = 0, 1, \ldots, \lceil L/p_{\max} \rceil - 1\}$.

   (c) For each frame, use the frame-scheduling technique from [49] to produce a feasible schedule for that frame. Concatenate the schedules of all frames to obtain the final schedule.

The following theorem shows the performance guarantee of the above algorithm, when given a chain decomposition.

Theorem 38 *Given an instance of treelike GDSS and a chain decomposition of its DAG $G(V, E)$ into λ blocks, the schedule S'' produced by the above algorithm has makespan $O(\rho \cdot (P_{\max} + \Pi_{\max}))$ with high probability, where $\rho = \max\{\lambda, \log n\} \cdot \lceil \log(\min\{p_{\max}, n\}) / \log\log n \rceil$. Furthermore, the algorithm can be derandomized in polynomial time.*

Proof We only analyze the above randomized algorithm. The delays can then be easily seen to be computable deterministically by the method of conditional probabilities.

First, observe that S has a makespan of at most $L \doteq 2(P_{\max} + \lambda\Pi_{\max}/\log(np_{\max}))$: this is because the maximum processing time along any directed path is at most $2P_{\max}$, and since there are λ points along any path which have been delayed, the additional delay is at most $2\lambda\Pi_{\max}/\log(np_{\max})$. Clearly, S' has no larger makespan. Let $C(M_i, t)$ be the contention of machine M_i at time t under S. The contention on any machine at any time under S' is no more than under S.

The following key lemma bounds the contentions:

Lemma 39 *There exists a constant $c_1 > 0$ such that $\forall i \in \{1, \ldots, m\}$, $\forall t \in \{1, \ldots, L\}$, $C(M_i, t) \leq \alpha$ with high probability, where $\alpha = c_1 \log(np_{\max})$.*

Proof For any job v, define the random variable $X(v, i, t)$ to be 1 if v is scheduled on M_i during the time interval $[t, t+1)$ by S, and 0 otherwise. Note that $C(M_i, t) = \sum_{v:m(v)=M_i} X(v, i, t)$. Let $d(v)$ be the random delay given to the chain to which v belongs. Conditioning on all other delays, $d(v)$ can take at most p_v values in the range $\{0, 1, \ldots, B-1\}$ that will lead to v being scheduled on M_i during $[t, t+1)$. Hence, $\mathbf{E}[X(v, i, t)] = \Pr[X(v, i, t) = 1] \leq \frac{p_v}{B}$. Hence $\mathbf{E}[C(M_i, t)] \leq \frac{\Pi_{\max}}{B} \leq \log(np_{\max})$. Although the random variables $X(v, i, t)$ are not independent, we will now present an upper-tail bound for $C(M_i, t)$.

Let $B_1, \ldots, B_\lambda$ be the blocks in the chain decomposition. Consider the following ordering of nodes in V: nodes within each B_i are ordered so that if v is an ancestor of w, then v precedes w, and nodes in B_i are ordered before nodes in B_{i+1}, for each i. Let $\pi(1), \ldots, \pi(n)$ be the resulting ordering of nodes. For any node v, and for any subset $W \subset V$ such that $\forall v' \in W$, $\pi(v') < \pi(v)$, we will argue that $\Pr[X(v, i, t) = 1 \mid \bigwedge_{v' \in W} X(v', i, t) = 1] \leq p_v/B$ in such a case. First, observe that if there is a node $v' \in W$ such that v' is an ancestor or descendant of v, then $X(v, i, t) = 0$, since the schedule S' preserves precedences. Therefore, assume that for each $v' \in W$, it is neither an ancestor nor a descendant of v. Let A be the chain containing v in the chain decomposition. Then, the random delay given at the start node of A does not affect any of the nodes in W, and conditioned on all other delays, $\Pr[X(v, i, t) = 1 \mid \bigwedge_{v' \in W} X(v', i, t) = 1] \leq p_v/B$ continues to hold. Thus, Fact 37 can now be applied to bound $\Pr[C(M_i, t) \geq \alpha]$, with $\sum_i q_i = \log(np_{\max})$ and $\delta = c_1 - 1$. Since $e^\delta/(1+\delta)^{1+\delta}$ decreases with δ for $\delta \geq 0$ and tends to 0 as $\delta \to \infty$, we thus get $\Pr[C(M_i, t) \geq \alpha] \leq 1/(np_{\max})^c$, where the constant c can be made arbitrarily large by taking c_1 large enough. Since the number of events "$C(M_i, t) \geq \alpha \log(np_{\max})$" is $O((np_{\max})^{c'})$ for a constant c', the lemma now follows via a union bound. ■

The above lemma implies that schedule S' has a low contention for each machine at each time instant, with high probability. Our final task is to verify that Step 4 of our algorithm gives the desired bounds. From the observation earlier, S' has a makespan at most L. By the definition of $p_{\max}$ and the fact that S' is well-structured, no job crosses over a frame. Given such a well-structured frame of length $p_{\max}$ where the maximum contention on any machine is at most α, the frame scheduling algorithm of [49] gives a feasible schedule with the following bounds.

Fact 40 *Given a well-structured frame of length $p_{\max}$ where the maximum contention on any machine is at most α, there exists a deterministic algorithm which delivers a schedule for this frame with makespan $O(p_{\max}\alpha\lceil\log p_{\max}/\log\log\alpha\rceil)$. Hence, concatenating the frames yields a schedule of length $O(\rho'(P_{\max}+\Pi_{\max}))$, where $\rho' = \max\{\lambda, \log(np_{\max})\}\lceil\frac{\log p_{\max}}{\log\log(np_{\max})}\rceil$.*

Note that if $p_{\max}$ is polynomially bounded in n, then Theorem 38 holds immediately. We now propose a simple reduction for the case where $p_{\max} \gg n$ to the case where $p_{\max}$ is polynomial in n. Assume that in the given instance I, $p_{\max} \geq n^{10}$. Create a new instance I' which retains only those vertices in I whose processing times are greater than $p_{\max}/n^2$. Vertices in the new instance I' inherit the same precedence constraints amongst themselves which they were subject to in I. However, all these vertices have processing times in the range $[p_{\max}/n^2, p_{\max}]$. Equivalently, all processing times can be scaled down such that they are in the range $[1, n^2]$. Hence, Fact 40 implies that we can obtain a schedule $\mathcal{S}'$ for instance I' whose length is $\rho(P_{\max}+\Pi_{\max})$, where $\rho = \max\{\lambda, \log n\} \cdot (\log n/\log\log n)$. We note that the total processing time of all the vertices in $I \setminus I'$ is at most $n\frac{P_{\max}}{n^2} = P_{\max}/n$. Hence, these vertices can be inserted into $\mathcal{S}'$ valid schedule $\mathcal{S}$ for I such the makespan increases by at most $P_{\max}/n$, and hence schedule $\mathcal{S}$ is also of length $\rho(P_{\max}+\Pi_{\max})$.

This completes the proof of Theorem 38. ■

Theorem 41 demonstrates a chain decomposition of width $O(\log n)$ for any out-tree: this completes the algorithm for an out-tree. An identical argument works for the case of a directed in-tree.

Theorem 41 *Given an out-tree, we can construct a chain decomposition of it with chain-width at most* $\lceil \lg n \rceil + 1$*, in deterministic polynomial-time.*

Proof The construction proceeds in iterations, each of which creates a block of the decomposition. Define $T_1(V_1, E_1) = T(V, E)$. Let $T_i(V_i, E_i)$ be the subtree at the beginning the i^{th} iteration. Let $S_i \subseteq V_i$ be the set of vertices u such that: (i) the subtree rooted at u in T_i is a directed chain, and (ii) the parent (if any) of u in T_i has out-degree at least two. During the i^{th} iteration, we create a block $B_{\lambda+1-i}$ which contains each $u \in S_i$ along with its subtree; we then remove all vertices of this block from T_i. It is easy to see that the graph induced by V_{i+1} is an out-tree T_{i+1}, and this procedure can be run on T_{i+1}; therefore, we do obtain a valid chain decomposition.

Claim 42 *Let* $\beta_{\lambda+1-i}$ *denote the number of chains induced by* $B_{\lambda+1-i}$*, in the* i*th iteration. Then for all* i*,* $\beta_{\lambda+1-i} \geq 2\beta_{\lambda-i}$*.*

Proof Consider a leaf vertex u in T_{i+1} (and hence belonging to $B_{\lambda-i}$). Vertex u has out-degree zero in T_{i+1} and out-degree of at least two in T_i (otherwise, u would have belonged to $B_{\lambda+1-i}$ leading to a contradiction). Hence, there are at least two chains induced by $B_{\lambda+1-i}$ for which u is an ancestor. Further, each chain in $B_{\lambda+1-i}$ has at most one ancestor in $B_{\lambda-i}$ which is a leaf. Since any directed chain has a unique leaf vertex, the claim follows. ■

Claim 42 implies that $\beta_\lambda \geq 2^{\lambda-1}$. Since $\beta_\lambda \leq n$, Theorem 41 follows. ■

Thus we get:

Theorem 43 *There is a deterministic polynomial-time approximation algorithm for solving the GDSS problem when the underlying DAG is restricted to be an in/out tree. The algorithm computes a schedule with makespan* $O((P_{\max} + \Pi_{max}) \cdot \rho)$*, where* $\rho = \log n \cdot \lceil \log(\min\{p_{\max}, n\}) / \log\log n \rceil$*. In particular, we get an* $O(\log n)$*–approximation in the case of unit-length jobs.*

GDSS on arbitrary forest-shaped DAGs

We now consider the case where the undirected graph underlying the DAG is a forest. The chain decomposition algorithm described in Theorem 41 does not work for arbitrary forests, and it is not clear how to make the Lemma 39 work with chain decompositions of arbitrary forests. Instead of following the approach of Section 6.2.2, we observe that once we have a chain decomposition, the problem restricted to a block of chains is precisely the job shop scheduling problem. This allows us to reduce the $R|forest|C_{max}$ problem to a set of job shop problems, for which we use the algorithm of [49]. While this is simpler than the algorithm in Section 6.2.2 for in-/out-trees, we incur another logarithmic factor in the approximation guarantee.

We now show how a good decomposition can be computed for forest-shaped DAGs:

Lemma 44 *Given an arbitrary DAG* T *whose underlying undirected graph is a forest, we can efficiently construct a chain decomposition of it into* γ *blocks, where* $\gamma \leq 2(\lceil \lg n \rceil + 1)$*.*

Proof Add an artificial "root" r to T and add an arc from r to some nodes in T, so that the underlying undirected graph becomes a tree. If we imagine T hanging down from r, some edges in T will be pointing away from the root (down) and others will be pointing toward the root (up). Imagine that T is an out-tree and perform a chain decomposition as in the proof of Theorem 41 which will result in a decomposition $\mathcal{B}$

with *blocks* $B_1, \ldots, B_\lambda$ and intermediate trees $T_1, \ldots, T_\lambda$. We now re-partition these blocks into *partitions* $P_1, \ldots, P_{2\lambda}$ of the chain decomposition $\mathcal{P}$ (refer to Figure 6.1 for an illustration; note that we use the term *blocks* for the intermediate decomposition and *partitions* for the final decomposition). Consider a "chain" $\mathcal{C}$ in B_i. In general, some edges of $\mathcal{C}$ will point down and others will point up. For instance, in Figure 6.1, nodes $a, b, \ldots, f$ form a chain in B_i and so do nodes $g, \ldots, m$; the edges (a, b) and (c, b) point down and up respectively. Each node $u \in B_i$ can be classified into the two following types and is put into P_i or $P_{2\lambda+1-i}$ accordingly. Imagine that T is undirected, and consider the sequence of edges which connect node u to the tree T_{i+1}. If the first edge e in this sequence (i.e., the edge e which has u as one of its end-points) points up, then u is a *type 1* node and put into P_i. Otherwise, if e points down, then u is a *type 2* node and put into $P_{2\lambda+1-i}$. This classification is motivated by the following observation: no node in T_{i+1} can be the ancestor of a type 1 node or a descendant of a type 2 node; since type 1 nodes belong to P_i, type 2 nodes belong to $P_{2\lambda+1-i}$, and nodes in T_{i+1} belong to partitions P_j where $i < j < 2\lambda + 1 - i$, the precedence conditions in the chain decomposition (as required by property **(P2)**) are satisfied. We now formally argue that our construction results in a valid chain decomposition.

We first show that Property **(P1)** of the chain decomposition holds, i.e., in the induced subgraph of a partition, each node has in and out-degrees of at most one, and there are no cycles. Since T has a tree structure, the cycle-free property follows immediately. Consider stage i of the decomposition. Let H_i be the induced subgraph of the nodes in block B_i. The total degree (in-degree + out-degree) of any node in H_i is at most 2. If a node u has two out-neighbors in H_i, then u must be of type 1 and at least one of its neighbors must be of type 2 (see node d in Figure 6.1 for instance). Hence, node u is in partition P_i and one of its out-neighbors is in partition $P_{2\lambda+1-i}$. It follows from a similar argument that if u has two in-neighbors in H_i, then u will

be in $P_{2\lambda+1-i}$ and one of its neighbors will be in P_i. All other nodes nodes have an in-degree and out-degree of at most one in H_i. Hence, in the induced subgraph of P_i and $P_{2\lambda+1-i}$, each node has an in-degree and out-degree of at most one, and property **(P1)** holds.

We now show that property **(P2)** holds. Consider stage i of the decomposition. Let node v be a descendant of node u. We consider the following cases:

Case 1: Both u and v belong to the same chain C in block B_i. If u and v are of the same type, then it is easy to see that all the nodes in the directed path from u to v are also of the same type as u and v. Hence, all these nodes will be put in the same partition, and property **(P2)** holds. If u and v are of different types, the only possibility is that u is of type 1 and v is of type 2. In this case, $u \in P_i$ and $v \in P_{2\lambda+1-i}$; since $i < 2\lambda + 1 - i$, property **(P2)** follows.

Case 2: Nodes u and v belong to different chains in block B_i. In this case, there exists a directed path from u to a node x in T_{i+1}, a directed path from node y in T_{i+1} to v, and a directed path from x to y in T_{i+1}. Clearly, node u will be of type 1 and node v will be of type 2 and property **(P2)** follows due to the same argument as in Case 1.

Case 3: Node $u \in B_i$ and $v \in T_{i+1}$. In this case, there is a directed path from $u \in B_i$ to $v \in T_{i+1}$, and hence u is of type 1 and $u \in P_i$. Further, since $v \in T_{i+1}$, v can only be in a partition P_j such that $i + 1 \leq j \leq 2\lambda - i$; thus $i < j$, and **(P2)** is satisfied.

Case 4: Node $v \in B_i$ and $u \in T_{i+1}$. Property **(P2)** is satisfied due to similar arguments as in Case 3.

This completes the proof of the Lemma. ■

Theorem 45 *Given a GDSS instance and a chain decomposition of its DAG $G(V, E)$ into γ blocks, there is a deterministic polynomial-time algorithm which delivers a schedule of makespan $O((P_{\max} + \Pi_{\max}) \cdot \rho)$, where $\rho = \frac{\gamma \log n}{\log \log n} \cdot \left\lceil \frac{\log \min(p_{\max}, n)}{\log \log n} \right\rceil$. Thus,*

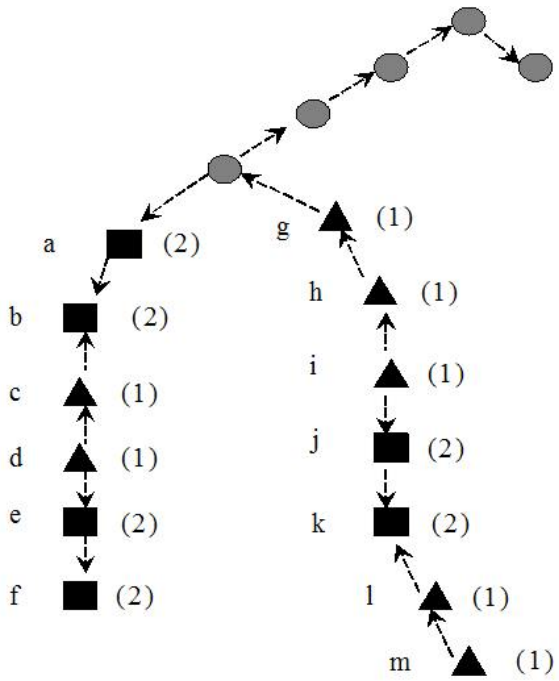


Figure 6.1: Chain-decomposition of a DAG whose underlying structure is a tree: the triangular and rectangular nodes are in block B_i while the circular nodes are in the subtree T_{i+1}. The triangular nodes are of type 1 and belong to partition P_i while the rectangular nodes are of type 2 and belong to partition $P_{2\lambda+1-i}$. The letters $a, \ldots, m$ to the left of the nodes denote their labels and the numbers to their right in parentheses denote their types.

Lemma 44 implies that $\rho = O\left(\frac{\log^2 n}{\log\log n} \cdot \left\lceil \frac{\log \min(p_{\max}, n)}{\log\log n} \right\rceil\right)$ *is achievable for forest-shaped DAGs.*

Proof Consider the chain decomposition of the DAG with γ blocks $P_1, P_2, \ldots, P_\gamma$. Each of these blocks P_i is an instance of job-shop scheduling, since it only consists of chains. These can be solved using the algorithm of [49] which, given a job-shop instance, produces a schedule with makespan at most $O(\frac{(P_{\max}+\Pi_{\max})\log n}{\log\log n}\lceil \log p_{\max}/\log\log n \rceil)$. Also, by the properties of the chain decomposition, there are no precedence constraints from P_j to P_i, for $j > i$. Therefore, we can concatenate the schedules for each block, and this yields a schedule for the GDSS instance with the desired makespan (since, as argued in the proof of Theorem 38, we may assume without loss of generality that $p_{\max}$ is polynomially bounded in n). ■

6.2.3 The Limits of our Lower Bound

Any attempt to improve our approximation guarantees must address the issue of how good the $P_{\max} + \Pi_{\max}$ lower bounds are. We show that in general DAGs, the $LB = \max\{P_{\max}, \Pi_{\max}\}$ lower bound is very weak: there are instances where the optimal makespan is $\Omega(P_{\max}\Pi_{\max})$. This leaves the question for forests- we show that even in this case, there are instances where the optimal makespan is $\Omega(LB \cdot \log n / \log \Pi_{\max})$.

We construct a rooted in-tree T for which the optimal makespan is $\Omega(LB \cdot \log n / \log \Pi_{\max})$, for any value of $\Pi_{\max}$ that is $\Omega(\log n / \log \log n)$. All nodes (jobs) are of unit length. At level 0, we have the root which is assigned to a processor that is not used for any other nodes. Once the level-i nodes are fixed, level-$(i+1)$ nodes are fixed in the following manner. For each node v at level i, there are $C = \Pi_{\max}$ nodes in level $i+1$ that are immediate predecessors of v. All these C nodes are assigned to the same machine that is never used again. Since there are n nodes in T, it is clear that there are $\log n / \log C$ levels, and about n/C machines are used.

Lemma 46 *The optimal makespan for the above instance is* $\Omega((\Pi_{\max}+P_{\max}) \log n / \log \Pi_{\max})$.

Proof We have $C = \Pi_{\max}$. Let V_i denote the set of nodes in level i. Note that $D \doteq P_{\max} = \log n / \log C + 1$ is the number of levels in T. We will show by backward induction on i that the earliest time that nodes in V_i can start is $(D-i)C$. From this the lemma follows, since $C \geq \Omega(\log n / \log \log n)$. The base case $i = D$ is obvious. Now assume this claim is true for levels $j \geq i$. Consider $v \in V_{i-1}$. Let $P(v)$ denote the immediate predecessors of v in level i. By construction, $|P(v)| = C$, and by the induction hypothesis, the earliest time any node in $P(v)$ can start is $(D-i)C$. All the nodes in $P(v)$ are assigned to the same processor. Therefore, the earliest time all nodes in $P(v)$ are done is $(D-i)C + C = (D-i+1)C$. Note that v can start only after all of $P(v)$ is completed. This completes the proof for forest-shaped instances.

■

An $\Omega(\sqrt{n})$ gap for general DAGs

In the above instance, the optimal makespan is also $\Omega(P_{\max}\Pi_{\max})$, but the ratio of this to $P_{\max} + \Pi_{\max}$ is only $O(\log n / \log \Pi_{\max})$, because $P_{\max} = \log n / \log \Pi_{\max}$. We now show an instance of the general GDSS problem where the optimal makespan is $\Omega(P_{\max}\Pi_{\max})$ and this quantity is $\Omega(\sqrt{n})$ times larger than $\Pi_{\max} + P_{\max}$. This instance has $m = \sqrt{n}$ machines and m layers; each layer contains m nodes, each to be processed on a distinct machine with unit processing time. Let these layers be denoted by $V_1, \ldots, V_m$. For each $i = 1, \ldots, m-1$, all edges in $V_i \times V_{i+1}$ are present. It is easy to see that $P_{\max} = \Pi_{\max} = m$ in this instance, but the optimal makespan is $n = P_{\max}\Pi_{\max}$. We show in Section 6.4 that the natural time-indexed integer program considered therein, also has an $\Omega(m)$ gap between the integral and fractional optima for this instance.

6.3 The $R|forest|\sum_j w_j C_j$ problem

We consider next the objective of minimizing weighted completion time, where the given weight for each job j is $w_j \geq 0$. Given an instance of $R|prec|\sum_j w_j C_j$ where the jobs have not been assigned their processors, we now reduce it to instances of $R|prec|C_{\max}$ with processor assignment. More precisely, we show the following: let $P_{\max}$ and $\Pi_{\max}$ denote the "dilation" and "congestion" as usual; if there exists a schedule of makespan $\rho \cdot (P_{\max} + \Pi_{\max})$ for $R|prec|C_{\max}$, then there is a $O(\rho)$-approximation algorithm for $R|prec|\sum_j w_j C_j$. We adapt an approach of [31, 105] for this. Let the machines and jobs be indexed respectively by i and j; $p_{i,j}$ is the (integral) time for processing job j on machine i, if we choose to process j on i. We now present an LP-formulation for $R|prec|\sum_j w_j C_j$ which has the following variables: for $\ell = 0, 1, \ldots$, variable $x_{i,j,\ell}$ is the indicator variable which denotes if "job j is processed on machine i, and completes in the time interval $(2^{\ell-1}, 2^{\ell}]$"; for job j, C_j is its completion time,

and z_j is the time spent on processing it. The LP is to minimize $\sum_j w_j C_j$ subject to:

$$\begin{aligned}
\forall j,\ \sum_{i,\ell} x_{i,j,\ell} &= 1 && (6.7)\\
\forall j,\ z_j &= \sum_i p_{i,j} \sum_\ell x_{i,j,\ell} &&\\
\forall (j \prec k),\ C_k &\geq C_j + z_j &&\\
\forall j,\ \sum_{i,\ell} 2^{\ell-1} x_{i,j,\ell} &\leq C_j \leq \sum_{i,\ell} 2^\ell x_{i,j,\ell} && (6.8)\\
\forall (i,\ell),\ \sum_j p_{i,j} \sum_{t \leq \ell} x_{i,j,t} &\leq 2^\ell && (6.9)\\
\forall \ell\ \forall \text{maximal chains } \mathcal{P},\ \sum_{j \in \mathcal{P}} \sum_i p_{i,j} \sum_{t \leq \ell} x_{i,j,t} &\leq 2^\ell && (6.10)\\
\forall (i,j,\ell),\ (p_{i,j} > 2^\ell) &\implies (x_{i,j,\ell} = 0) && (6.11)\\
\forall (i,j,\ell),\ x_{i,j,\ell} &\geq 0 &&
\end{aligned}$$

Note that (6.9) and (6.10) are "congestion" and "dilation" constraints respectively. Our reduction proceeds as follows. Solve the LP, and let the optimal fractional solution be denoted by variables $x^*_{i,j,\ell}$, C^*_j, and z^*_j. We do the following filtering, followed by an assignment of jobs to (machine, time-frame) pairs.

Filtering: For each job j, note from the first inequality in (6.8) that the total "mass" (sum of $x_{i,j,\ell}$ values) for the values ℓ such that $2^\ell \geq 4C^*_j$, is at most 1/2. We first set $x_{i,j,\ell} = 0$ if $2^\ell \geq 4C^*_j$, and scale each $x_{i,j,\ell}$ to $x_{i,j,\ell}/(1 - \sum_{\ell' \geq 4C^*_j} \sum_i x_{i,j,\ell'})$, if ℓ is such that $2^\ell < 4C^*_j$ - this ensures that equation (6.7) still holds. After the filtering, each non-zero variable increases by at most a factor if 2. Additionally, for any fixed j, the following property is satisfied: if ℓ' is the largest integer such that $x_{i,j,\ell'}$ is non-zero, then $2^{\ell'} = O(C^*_j)$. The right-hand-sides of (6.9) and (6.10) become at most $2^{\ell+1}$ in the process and the C_j values increase by at most a factor of two.

Assigning jobs to machines and frames: For each j, set $F(j)$ to be the frame $(2^{\ell-1}, 2^\ell]$, where ℓ is the index such that $4C^*_j \in F(j)$. Let $G[\ell]$ denote the sub-

problem which is restricted to the jobs in this frame. Let $P_{\max}(\ell)$ and $\Pi_{\max}(\ell)$ be the fractional congestion and dilation, respectively, for the sub-problem restricted to $G[\ell]$. From constraints (6.9) and (6.10), and due to our filtering step, which at most doubles any non-zero variable, it follows that both $P_{\max}(\ell)$ and $\Pi_{\max}(\ell)$ are $O(2^\ell)$. We now perform a processor assignment as follows: for each $G[\ell]$, we use the processor assignment scheme in Section 6.2.1 to assign processors to jobs. This ensures that the integral $P_{\max}(\ell)$ and $\Pi_{\max}(\ell)$ values are still at most $O(2^\ell)$.

Scheduling: First schedule all jobs in $G[1]$; then schedule all jobs in $G[2]$, and so on. We can use any approximation algorithm for makespan-minimization, for each of these scheduling steps. It is easy to see that we get a feasible solution: for any two jobs j_1, j_2, if $j_1 \prec j_2$, then $C^*_{j_1} \leq C^*_{j_2}$ – frame $F(j_1)$ occurs before $F(j_2)$ and hence gets scheduled first.

Theorem 47 *Consider any family $\mathcal{F}$ of precedence constraints that is closed under taking subsets: i.e., if a partial order σ is in $\mathcal{F}$, then any partial order obtained by removing arcs from σ is also in $\mathcal{F}$. If there exists an approximation algorithm for $R|prec|C_{\max}$ for all precedence constraints $\sigma \in \mathcal{F}$ that yields a schedule whose makespan is $O((P_{\max} + \Pi_{\max}) \cdot \rho)$, then there is also an $O(\rho)$–approximation algorithm for $R|prec|\sum_j w_j C_j$ for all $\sigma \in \mathcal{F}$. Thus, Theorem 45 implies that an $O(\frac{\log^2 n}{\log\log n}\lceil\frac{\log \min(p_{\max},n)}{\log\log n}\rceil)$-approximation is achievable for $R|forest|\sum_j w_j C_j$.*

Proof Consider any job j which belongs to $G[\ell]$; since both $P_{\max}(\ell)$ and $\Pi_{\max}(\ell)$ are $O(2^\ell)$, the jobs in $G[\ell]$ take a total of $O(\rho 2^\ell)$ to complete. Thus, even given the wait for all jobs in $G[\ell']$ for $\ell' < \ell$, the completion time of job j is $O(\rho \cdot \sum_{\ell' \leq \ell} 2^{\ell'}) = O(\rho 2^\ell)$. Since $2^\ell = O(C^*_j)$, the theorem follows. ■

6.4 Minimizing the weighted flow time under precedence-chains

We now study the flow-time objective. We consider the *restricted-assignment variant* where for every job v, there is a value p_v such that for all machines i, $p_{i,v} \in \{p_v, \infty\}$. We focus on the case where the precedence DAG is a disjoint union of chains with all p_u being polynomially-bounded positive integers in the input-size N. We present a bicriteria approximation algorithm for the weighted flow time.

Let us recall the flow-time objective. In addition to the chain-shaped precedence constraints and the machine assignment constraints, each job v also has a "release-time" r_v and a deadline l_v. The release time specifies the time at which the job v was created and hence it can be started only at times which are $\geq r_v$. The deadline l_v specifies the last time slot at which this job can be started, say. Our goal is to find a schedule which minimizes the weighted flow time $\sum_v w_v(C_v - r_v)$ subject to the above constraints. In general, minimizing the weighted flow time appears very hard to approximate. Leonardi & Raz [85] provide a $O(\sqrt{n}\log n)$-approximation algorithm for this problem and show that it cannot be approximated within a factor of $O(n^{\frac{1}{3}-\delta})$ for any constant $\delta > 0$, unless $P = NP$. One way of dealing with such intractability is through *resource augmentation*, where we allow our solution to use ψ copies of a machine. In this case, we say that the solution is an ψ-speed solution. Let OPT be the cost (total weighted flow time) of the optimal schedule. We say that a solution is (ψ, γ)-approximate, if its speed is ψ and the cost of the solution is at most $\gamma \cdot OPT$. Note that we compare the cost of our ψ-speed solution with that of a 1-speed optimal solution. We now present an algorithm, which either proves that input instance is has no feasible solution, or outputs a (ψ, γ)-approximate solution where $\psi = O(\frac{\log N}{\log\log N})$ and $\gamma = 1 + o(1)$ with high probability. (Recall that N here denotes the input size.)

Let $T = \sum_u p_u$. The values r_v and l_v could trivially be 1 and ∞, respectively; if $l_v > T$, we reset l_v without loss of generality to T. Let $\prec$ denote the immediate-predecessor relation, i.e., if $u \prec v$, then they both belong to the same chain and u is an immediate predecessor of v in this chain. Note that if v is the first job in its chain, then it has no predecessor. Let $S(v)$ denote the set of machines on which v can be processed: i.e., the set $\{i : \ p_{i,v} = p_v\}$. In the time-indexed LP formulation below, we consider the LP relaxation of the integer program in which the variables have the following interpretation. For each job v and time t, $r_v \leq t \leq l_v$, $x_{v,i,t}$ is the indicator for job v being *started* on machine i at time t, $z_{v,t}$ is the indicator for job v being started at time t, and C_v is the completion time of v. (Henceforth, any variable $x_{v,i,t}$ or $z_{v,t}$ where t is not in the range $[r_v, l_v]$, is taken to be zero.) The objective is $\min \sum_v w_v(C_v - r_v)$, subject to:

$$\forall v, \sum_{i \in S(v)} \sum_{t \in [r_v, \ldots, l_v]} x_{v,i,t} = 1 \tag{6.12}$$

$$\forall\ i \in [1, \ldots m]\ \forall t \in [1, \ldots T], \ \sum_{v} \sum_{\max\{r_v, t-p_v+1\} \leq t' \leq t} x_{v,i,t'} \leq 1 \tag{6.13}$$

$$\forall v\ \forall t \in [r_v, \ldots, l_v], \ z_{v,t} = \sum_{i \in [1, \ldots m]} x_{v,i,t} \tag{6.14}$$

$$\forall u \prec v\ \forall t \in [r_v, \ldots, p_u], \ z_{v,t} = 0 \tag{6.15}$$

$$\forall u \prec v\ \forall t \in [p_u + 1, \ldots T], \ \sum_{t' \in [1, \ldots, t]} z_{v,t'} \leq \sum_{t' \in [1, \ldots, t - p_u]} z_{u,t'} \tag{6.16}$$

$$\forall v, \ C_v = \sum_{t \in [1, \ldots T]} (t + p_v - 1) z_{v,t} \tag{6.17}$$

$$\forall v\ \forall i \in S(v)\ \forall t \in [r_v, \ldots, l_v], \ x_{v,i,t} \geq 0$$

The constraints (6.12) ensure that all jobs are processed completely, and (6.13) ensure that at most one job is (fractionally) assigned to any machine at any time. Constraints (6.15) and (6.16) are the precedence constraints and (6.17) defines the completion time C_v for job v.

Our algorithm proceeds as follows. We first solve the above LP optimally; if there is no feasible solution (e.g., if the deadlines l_v are too restrictive), we announce this and stop. Otherwise, let OPT be the optimal value of the LP and let x^*, z^* and C^* be the vectors denoting the optimal solution-values. We define a randomized rounding procedure for each chain such that the following three properties hold:

(A1) Let $Z_{v,t}$ be the indicator random variable which denotes if v is started at time t in the rounded solution. Let $X_{v,i,t}$ be the indicator random variable which denotes if v is started at time t on machine i in the rounded solution. Then $\mathbf{E}[Z_{v,t}] = z^*_{v,t}$ and $\mathbf{E}[X_{v,i,t}] = x^*_{v,i,t}$.

(A2) All precedence constraints are satisfied in the rounded solution with probability one.

(A3) Jobs in different chains are rounded independently.

Our rounding procedure to choose the $Z_{v,t}$ is as follows. For each chain Γ, we choose a value $R(\Gamma) \in [0, 1]$ uniformly and independently at random. For each job v belonging to chain Γ,

$$Z_{v,t'} = 1 \text{ iff } \sum_{t=1}^{t'-1} z^*_{v,t} < R(\Gamma) \leq \sum_{t=1}^{t'} z^*_{v,t}. \tag{6.18}$$

Bertsimas *et al.* [23] show other applications of such rounding techniques. After the $Z_{v,t}$ values have been determined, we do the machine assignment as follows: if $Z_{v,t} = 1$, then job v is started on exactly one machine at time t, with the probability for machine i being $(x^*_{v,i,t}/z^*_{v,t})$. A moment's reflection shows that:

- property **(A1)** holds because the condition on $R(\Gamma)$ in (6.18) happens with probability $z^*_{v,t'}$;
- **(A2)** holds due to the precedence constraints (6.15) and (6.16); and
- **(A3)** is true since the different chains choose the values $R(\Gamma)$ independently.

In general, this assignment strategy might result in jobs from *different* chains executing on the same machine at the same time, and hence in an infeasible schedule. (Jobs from the same chain cannot contend for the same machine at the same time, since property **(A2)** holds.) Let Y be the random variable which denotes the maximum contention of any machine at any time. We obtain a feasible solution by resource augmentation: deploying Y copies of each machine.

An application of the Chernoff-type bound from Fact 37 yields the following bound on Y, which we state in general terms without assuming that all the p_v are bounded by a polynomial of N:

Lemma 48 *Let $p_{\max} \doteq \max_v p_v$, and define $M = \max\{N, p_{\max}\}$. Let $\mathcal{E}$ denote the event that $Y \leq (\alpha \log M / \log\log M)$, where $\alpha > 0$ is a suitably large absolute constant. Event $\mathcal{E}$ occurs after the randomized machine assignment with high probability: this probability can be made at least $1 - 1/M^{\beta}$ for any desired constant $\beta > 0$, by letting the constant α be suitably large.*

Proof Let $L_{i,t}$ denote the contention on machine i at time-step t in the infeasible schedule. The number of such random variables $L_{i,t}$ is at most $m \cdot T \leq mn \cdot p_{\max}$, which is bounded by a fixed polynomial of M. So, it suffices to fix (i,t) arbitrarily, and to show that for any desired constant $\beta' > 0$, a large enough choice of the constant α ensures that

$$\Pr[L_{i,t} > \alpha \log M / \log\log M] \leq M^{-\beta'}; \tag{6.19}$$

a union bound over all (i,t) will then complete the proof.

Let us now prove (6.19). For each chain Γ, note that the total load imposed by Γ on machine i at time t in the infeasible schedule, is given by

$$U(\Gamma) \doteq \sum_{v \in \Gamma} \sum_{\max\{r_v, t-p_v+1\} \leq t' \leq t} X_{v,i,t'}.$$

(For notational convenience, we simply say "$U(\Gamma)$" instead of "$U_{i,t}(\Gamma)$".) So, $L_{i,t} = \sum_\Gamma U(\Gamma)$. We note some facts:

- By **(A1)** and by (6.13), $\mathbf{E}[L_{i,t}] \leq 1$;
- by **(A2)**, each $U(\Gamma)$ lies in $\{0, 1\}$; and
- by **(A3)**, the random variables $U(\Gamma)$ are *independent* of each other.

Since Fact 37 directly applies to a sum of independent binary random variables, we get, by setting $\sum_i q_i = 1$ in Fact 37, that for any $\delta > 0$,

$$\Pr[L_{i,t} \geq 1 + \delta] \leq (e/(1+\delta))^{1+\delta}.$$

A simple calculation now shows that (6.19) is satisfied by choosing $1+\delta = \alpha \log M / \log\log M$ for a large enough constant α. ■

Finally, we note that we construct a schedule only if the event $\mathcal{E}$ occurs. Otherwise, we can repeat the randomized machine assignment until event $\mathcal{E}$ occurs and resource-augment the resultant infeasible schedule. Since $p_{\max} \leq poly(N)$ by assumption, the event $\mathcal{E}$ implies that $Y = O(\log N / \log\log N)$. Note that for any job v, its completion time C_v equals $\sum_t (t + p_v - 1) \cdot Z_{v,t}$; so, $\mathbf{E}[C_v]$ equals the fractional completion time C_v^*, due to **(A1)** and the linearity of expectation. Thus, the expected value of the flow time F_v of v, also equals its fractional value $F_v^* = C_v^* - r_v$. Now, by Lemma 48, even conditional on the event $\mathcal{E}$,

$$\mathbf{E}[F_v \mid \mathcal{E}] \leq \frac{\mathbf{E}[F_v]}{\Pr[\mathcal{E}]} = \frac{F_v^*}{\Pr[\mathcal{E}]} = (1 + o(1)) \cdot F_v^*,$$

since $\Pr[\mathcal{E}] = 1 - o(1)$. So, even conditional on $\mathcal{E}$ (which happens with high probability), the expected cost of our solution is at most $(1 + o(1)) \cdot OPT$.

Integrality gap for the instance of Section 6.2.3. We now show that the relative of the above time-indexed formulation performs quite poorly for general DAGs, when applied to the GDSS problem (where we aim to minimize the makespan). Specifically, consider GDSS instances with all processing times being unity, as in Section 6.2.3. We first "guess" an upper bound T' for the makespan, as in Section 6.2.1. We write an LP such as the time-indexed one above, with the following modifications: (i) all the time variables t take values in $\{1, 2, \ldots, T'\}$; (ii) all the values r_v and l_v are trivial – i.e., $r_v \equiv 1$ and $l_v \equiv T'$, and (iii) the constraints (6.15) and (6.16) are included for *all* pairs of nodes (u, v) for which u is constrained to precede v. As mentioned in Section 6.2.3, the optimal (integral) solution for the instance therein has makespan $n = m^2$, but here is a fractional solution to this time-indexed formulation which makes $T' = 2m - 1$ feasible: if node v at level V_j has been pre-assigned to machine i, then $z^*_{v,t} = x^*_{v,i,t} = 1/m$ for $t = j, j + 1, \ldots, j + m - 1$ (and all other $z^*_{v,t}, x^*_{v,i,t}$ values are zero). Thus, even this time-indexed formulation has an integrality gap of $\Omega(\sqrt{n})$ for general DAGs.

Chapter 7

End-to-End Latency Minimization in Wireless Networks

7.1 Introduction

The recent past has witnessed an explosion of interest in multi-hop wireless networks. Nodes in these networks do not rely on any pre-existing routing infrastructure for communication, but instead communicate either directly or with the help of other intermediate nodes in the network. The distributed, wireless and self-configuring nature of multihop wireless networks make them suitable for a wide variety of applications such as sensing, monitoring, community networking, and disaster relief. At the same time, the unique characteristics of wireless networks also introduce major challenges in the design of wireless communication protocols.

A particular challenge is the broadcast nature of wireless medium which results in interference. When two or more edges in the network that are physically proximate to each other transmit simultaneously, then the signals from these transmissions destructively interfere with each other resulting in the loss of one or more of these transmissions. Our focus in this work is the design of wireless communication

protocols which effectively deal with interference and efficiently utilize the available network resources. We focus on a fundamental *latency minimization* problem in this Chapter. We are given a wireless network $G = (V, E)$ which is subject to interference (see Section 7.2 for details). Our first question is as follows: suppose we are given a collection of packets, with each packet also containing its source node, its destination node, and the path in the network it needs to traverse in order to reach from its source to its destination; each link can transmit at most one packet during a time step. How should we schedule the transmission of packets across their links in order to minimize the maximum time it takes for any packet to reach its destination (end-to-end latency or makespan minimization)?

End-to-end latency minimization is well understood in the case of *wireline* networks. An influential result here is the work of Leighton *et al.* [83, 82], who derive a centralized constant factor approximation algorithm for this problem. This work was a followed by a series of papers improving either the performance or the complexity of the algorithm (see [82, 114, 7]). Rabani and Tardos [106] give a distributed algorithm for this problem, that takes time $O(C+D(\log^* n)^{O(\log^* n)}+\log^6 n)$, which was improved by Ostrovsky and Rabani [97] to $O(C + D + \log^{1+\epsilon} n)$.

In this work, motivated by the geometric signal propagation properties in the wireless medium, we model the communication network as a geometric disk graph; our main contribution here is the design of near-optimal distributed and centralized approximation algorithms for end-to-end latency minimization under the disk graph network model. Specifically, we develop a centralized algorithm for arbitrary disk graphs, with an approximation ratio $O(\log n)$; we also develop a distributed implementation of this algorithm with a poly-logarithmic approximation guarantee. For the special case of unit disk graphs, we develop centralized and distributed algorithms with constant and log-factor approximation ratios respectively. For wireless networks with interference constraints, apart from our work, the only known results for latency

minimization is that of Heide *et al.* [93]. Heide *et al.* study the worst case trade-off between congestion, dilation and energy for routing algorithms for wireless networks, under network models similar to those considered here. They also provide online and off-line algorithms for the end-to-end scheduling problems considered in our work. However, the scheduling algorithms in [93] could have an approximation ratio $\Omega(n)$ in the worst case.

To the best of our knowledge ours is the first work which presents centralized and distributed poly-logarithmic approximation algorithms for end-to-end packet scheduling in wireless networks. A key technical innovation in our work is the notion of inductive ordering of the edges in the network. This notion plays a central role in our derivation of lower and upper bounds for end-to-end latency problem and is potentially of broader interest in the design of wireless communication protocols. In Chapters 8 and 9, we show two further applications of our geometric insights to throughput capacity estimation in wireless networks. We turn to the precise details of our network and interference models next.

7.2 Network model and problem statement

In this section, we provide a formal description of our network and interference models.

Communication Model: We consider multi-hop wireless networks, where all nodes operate on a fixed single channel. We model network connectivity using a directed disk graph $G = (V, E)$: all nodes in the graph are embedded in the plane $\mathbf{R}^2$, and node u has an associated range denoted by $r(u)$. A necessary (but not sufficient) condition for a node v to hear a transmission from node u is that v be within a distance $r(u)$ of u. Specifically, if transmission is not feasible from u to v either because v is outside the range of u or because of other reasons (such as the presence of an obstruction

between u and v), then the edge (u, v) is not present in the graph $G = (V, E)$. This is an important consideration for modeling realistic network scenarios such as indoor wireless networks or even outdoor networks in the presence of obstructions.

We assume that time is slotted and w.l.o.g., each time slot is one second in duration. We deal with both synchronous and asynchronous networks: when the network is synchronized, all nodes in the network know the index of the current time slot; in the asynchronous case, nodes may have different estimates for the index of the current time slot. A schedule $\mathcal{S}$ describes the specific times at which packets are moved over the links of the network. In other words, let $X_{e,t}$ be the indicator variable which is defined as follows:

$$X_{e,t} = \begin{cases} 1 & \text{if } e \text{ transmits successfully at time } t \\ 0 & \text{otherwise} \end{cases} \tag{7.1}$$

A schedule $\mathcal{S}$ is a $0-1$ assignment of the variables $X_{e,t}$, $e \in E$, $t \in 0, 1, \ldots$; we say a schedule $\mathcal{S}$ is periodic with period T, if $\forall e, t, i$: $X_{e,iT+t} = X_{e,(i+1)T+t}$.

Interference Model: Since the medium of transmission is wireless, simultaneous transmissions on proximate edges may *interfere* with each other resulting in collisions. Formally, we say that edges $e_1, e_2 \in E$ interfere with each other if edges e_1 and e_2 cannot both transmit successfully during the same time slot. Let $I(e_1)$ denote the set of edges which interfere with edge e_1. An *Interference Model* defines the set $I(e)$ for each edge e in the network. Several such models have been studied in the literature motivated by the variations in the underlying transmission technology and protocols. We consider two such interference models in this work.

Let $d_e(u, w)$ denote the Euclidean distance between the nodes u and w. In the **transmitter model (Tx-model)** of interference, a transmission from u is successful (i.e., received correctly by the intended recipient of the transmission) if and only

if any other transmitter w is such that, $d_e(u, w) \geq (1 + \Delta) \cdot (r(u) + r(w))$. Here, $\Delta > 0$ is a protocol-dependent constant which is specified as part of the input. This model was introduced by Yi *et al.* [128] to analyze the capacity of random ad hoc networks. Throughout this work, we focus on obtaining good performance guarantees for the Tx-model. We also show how to extend all our algorithms (while incurring at most a constant factor increase in the approximation ratios) to a more realistic interference model described next. In the **transmitter-receiver model (Tx-Rx model) [15, 74]** of interference: let $e_1 = (u, v) \in E$ be an edge along which there is a transmission. Let D denote the network distance (in terms of hop-count) between the edges and nodes in the network. Specifically, for any two edges e_1 and e_2, $D(e_1, e_2)$ is defined as the least hop-count distance between an incident node of e_1 and an incident node of e_2. The transmission along e_1 is successful if and only if any other transmission along an edge $e_2 \in E$ is such that $D(e_1, e_2) > 1$. In both the models above, a node can either receive a message or transmit a message (and not both) at the same time. Thus for any edge $e = (u, v)$, all other edges which are incident on u or v are also included in the set $I(e)$.

End-to-End Packet Scheduling Problem (EPSI): An instance of the end-to-end packet scheduling problem is specified as $EPSI\ (G(V, E), \{p_1, \ldots, p_k\})$. $G(V, E)$ is the underlying wireless network, and $p_1, \ldots, p_k$ are the packets to be transmitted, with packet p_i starting at s_i and destined for t_i, along the path P_i. The path P_i is encoded in packet p_i. We will assume that any packet takes one unit of time to cross a link and at any time at most one packet can cross a link. In addition, if packets are being sent simultaneously on edges e_1 and e_2, then for the transmission on e_1 to be successful, e_2 must *not* belong to the set $I(e_1)$ and vice-versa. If this condition is violated, the packet should be retransmitted on this edge during subsequent time slots until a successful transmission. Each node/edge has a buffer in which a packet

can wait till it successfully moves to the next node in its path. The objective is to construct a *valid schedule*, $\mathcal{S}$, that decides which packet should be sent out at a node at any time. A schedule is *valid* iff it sends all packets along their paths successfully subject to the above re-trial requirement. Let $C_{\mathcal{S}}(p_i)$ denote the time at which packet p_i is delivered in schedule $\mathcal{S}$. The *end-to-end latency* of $\mathcal{S}$, denoted by $C(\mathcal{S}) = \max_i C_{\mathcal{S}}(p_i)$ is the time taken by $\mathcal{S}$ to route all the packets, and our objective in the EPSI problem is to construct a schedule with low end-to-end latency.

7.3 Hardness of EPSI

In this section, we derive our complexity results for the EPSI problem. We first start with the hardness of approximating the EPSI problem in arbitrary (non-geometric) graphs under the Tx-Rx model of interference. In other words, we are given a directed graph $G' = (V', E')$, and a set of k packets along with the paths they need to traverse. Recall that in the Tx-Rx model of interference, the interference set $I(e) \subset E'$ for an edge e is the set of all edges e' such that at least one of the following conditions hold: (i) e and e' share an end-point; (ii) there is an edge from an end-point in e to an end-point in e'; (iii) there is an edge from an end-point in e' to an end-point in e. Our goal is to schedule the packet transmissions along the links in order to minimize the end-to-end latency without violating the interference constraint.

Lemma 49 *There exists a constant $\epsilon > 0$ such that, unless $P = NP$, there is no polynomial time algorithm to approximate the optimum makespan of every instance of $EPSI$ problem on arbitrary graphs, within a factor of n^{ϵ}.*

Proof We reduce the vertex coloring problem to $EPSI$. Given a graph $G(V, E)$, we construct a directed graph $G'(V', E')$ in the following manner. For each $v \in V$, we add vertices v and $m(v)$ in G'. Each edge $(u, v) \in E$ is part of E'; in addition, we have edges $(v, m(v))$, $\forall v \in G$ in E'. In the $EPSI$ instance, we have packet

p_v destined from v to $m(v)$, for each $v \in V$, and the path that p_v has to take is exactly the edge $(v, m(v))$. We will argue that the minimum makespan of the EPSI instance is exactly the chromatic number of G. Two edges $(u, m(u))$ and $(v, m(v))$ can be simultaneously scheduled if and only if the edges $(u, m(u))$ and $(v, m(v))$ do not interfere with each other. By construction, it is easy to see that this happens if and only if $(u, v) \notin E$. Therefore, in the EPSI instance, the set of packets that can be simultaneously transmitted during the same time slot corresponds to an independent set in G, and the number of slots required to transmit all the packets equals the chromatic number of G. Given the n^{ϵ}–hardness of chromatic number, the lemma follows. ■

Lemma 49 deals with the hardness of EPSI for arbitrary input graphs. However, as we show below, EPSI continues to remain NP-Hard even when the input graph is restricted to be a geometric (unit) disk graph. The following lemma and its proof holds for both the Tx model and the Tx-Rx model of interference.

Lemma 50 *Unless $P = NP$, there is no polynomial time algorithm for optimally solving every instance of $EPSI$ problem on unit disk graphs.*

Proof We exploit the fact that the coloring problem is NP-Hard even when restricted to unit disk graphs [32]. Given a unit disk graph $G = (V, E)$, we construct a new unit disk graph $G'(V', E')$ as follows. For each $v \in V$, we add vertices v and $m(v)$ in G'. Each edge (u, v) in E is part of E'; next, we add the edges $(v, m(v))$, $\forall v \in G$ in E'; finally, for every edge $(u, v) \in E$, we add edges $(u, m(v))$, and $(v, m(u))$ as well in E'. We claim that the new graph G' is also a unit disk graph: consider a two-dimensional realization of the unit disk graph G (in this realization, nodes in G are embedded on the plane, and a pair of nodes is within a distance of at most two from each other if and only their is an edge between them in G; since G is a UDG, such a realization is guaranteed to exist). For each node $v \in V$, we now create a new

node $m(v)$ and co-locate it with node v. The new unit-disk graph induced by the set of nodes V and nodes of the form $m(v)$, is exactly the graph G'. The rest of the construction for creating the $EPSI$ instance proceeds exactly along the lines of the proof of Lemma 49. Given that coloring is NP-Hard in the case of unit disk graphs, our lemma follows. ■

7.4 A Necessary condition for scheduling

In this section, we exploit the geometry of disk graphs to develop a necessary condition which must be satisfied by every valid schedule. Central to this necessary condition is the notion of *inductive ordering of edges*; this notion is the at the heart of the provably good approximation algorithms we develop in this chapter as well as in Chapter 8. We focus only on the Tx-model of interference, and defer the discussion for the Tx-Rx model. Recall that in the Tx-model of interference, a link (u, v) can transmit successfully during a time slot if and only if no other link (w, z) such that $d_e(u, w) \leq (1+\Delta)(r(u)+r(w))$ is active during the same slot. For ease of exposition, we assume that $\Delta = 0$ in the rest of this chapter for our analysis of the Tx-model; all claims and proofs easily extend when Δ is an arbitrary non-negative constant.

Definition 51 *Given a graph $G = (V, E)$, and an edge $e \in E$, the restricted interference set of $e = (u, v)$, $I_{\geq}(e)$ is defined as follows: $I_{\geq}(e) \doteq \{(p, q) \mid (p, q) \in I(e)$ and $r(p) \geq r(u)\}$.*

In other words, an edge e' is in the set $I_{\geq}(e)$, iff it interferes with e (i.e., if it belongs to $I(e)$), *and* the range of the transmitting end-point of e' is at least as much as the range of the transmitting end-point of e. We may view this as a decreasing ordering of the links in the network according their transmitter ranges; for a given link e, those links which precede e in the ordering and which interfere with it are in e's restricted

interference set. Recall that $X_{e,t}$ is the indicator variable which is 1 iff e transmits successfully during time t. The significance of the inductive ordering is due to the following claim.

Claim 52 *In any link schedule,*

$$\forall e \in E, \forall t \qquad X_{e,t} + \sum_{f \in I_{\geq}(e)} X_{f,t} \leq \beta \tag{7.2}$$

where β is a fixed constant that depends only on the interference model. In particular, for the Tx-model, the value of β is at most 5.

The intuition behind this claim is as follows. Partition the set of edges in $I_{\geq}(e) \cup \{e\}$ into at most β subsets such that within each subset, each edge interferes with all other edges. Thus, at most one edge can successfully transmit from each subset at any time slot, i.e., only β edges in $I_{\geq}(e) \cup \{e\}$ can simultaneously transmit successfully, as stated in the claim. We present a proof of this claim for the Tx-model below. Later, in Section 7.5 we prove the equivalent claim for the Tx-Rx model of interference.

Proof For any node u, define $I(u)$ and $I_{\geq}(u)$ analogous to the definition for edges as follows: $I(u) = \{w : d(u,w) < (1+\Delta) \cdot (r(u) + r(w))\}$. $I_{\geq}(u) = \{w \ : \ r(w) \geq r(u) \text{ and } w \in I(u)\}$. For any edge $e = (u, u')$, $I(e)$ is now defined as follows: $I(e) = \{e' = (w, v) \ : \ w \in I(u)\}$. Similarly, $I_{\geq}(e) = \{e' = (w, v) \ : \ w \in I_{\geq}(u)\}$. We now show that for any node u, at most five nodes in $I_{\geq}(u)$ can simultaneously transmit in any time slot without interfering with each other.

Consider any node u and a large disk C centered at u which contains all the nodes in the network. Consider any sector which subtends an angle of $\frac{\pi}{3}$ at u. Let $w, w' \in I_{\geq}(u)$ be two nodes in this sector. W.l.o.g., assume that $d(u,w') \geq d(u,w)$. It is easy to see that $d(w,w') \leq d(u,w')$. Further, we have $r(w') \geq r(u)$. Thus, w' has a bigger range than u and is closer to w than u. Since u and w interfere with

each other, clearly, w and w' also each interfere with each other and hence can not transmit simultaneously. Thus the angle subtended at u by any two simultaneous transmitters in the set $I_{\geq}(u)$ is strictly greater than $\frac{\pi}{3}$. Hence, there can be at most five successful transmitters from this set which proves the claim. ■

7.5 End-to-end distributed scheduling

We present our centralized and distributed algorithms for the EPSI problem in this section. Our algorithms for the EPSI problem involve choosing random delays at the first step, as in [83] and then scheduling packets at each time step by solving a distributed scheduling problem.

7.5.1 Disk graphs

We present a sequential scheduling algorithm for disk graphs which yields an $O(\log n)$ approximation to the makespan, while a distributed scheduling yields an $O(\log^2 n(1+\log \frac{r_{\max}}{r_{\min}}))$ approximation, where $r_{\max}$ and $r_{\min}$ are the maximum and the minimum radii of the nodes respectively. We need some additional notation. For edge $e = (u, v)$, define $r(e) = r(u)$. We extend the notion of the restricted interference set for a vertex u: define $I_{\geq}(u)$ to be the set of all edges $e' = (w, z)$, such that $r(w) \geq r(u)$, and e' interferes with some link of the form (u, v). For any fixed constant α, define $I_{\geq}(\alpha, u) = \{e' \mid e' \in I_{\geq}(u) \text{ and } r(e') \geq \alpha \cdot r(e)\}$. Let $H(e, i)$ denote the number of times packet i visits edge e. For any fixed value α, and an edge $e = (u, v)$, define $C(\alpha, e)$ to be the total number of times the edges in the set $I_{\geq}(\alpha, u)$ is used by the packets: i.e., $C(\alpha, e) = \sum_{e' \in I_{\geq}(\alpha,u)} \sum_i H(e', i)$. We say a set of edges is interference free, if no pair of edges which belong to this set interfere with each other.

Lemma 53 *For any vertex v and a fixed constant $\alpha > 0$, the size of the largest*

interference-free set in the subgraph induced by $I_{\geq}(\alpha, u)$ is at most a constant whose value depends only on α.

Proof The proof is via a packing argument. Suppose $S = \{e_1, \ldots, e_k\}$ is an interference-free set in $I_{\geq}(\alpha, u)$. Let $S_1 \subseteq S$ be the set of edges such that their transmitter range is strictly less than the range of u, and let $S_2 = S \setminus S_1$. By Claim 52, we have $|S_2| = O(1)$; hence, it suffices to show that $|S_1| = O(1)$. Observe that for any link $e' = (w, z) \in S_1$, we have $r(w) \geq \alpha r(u)$. Since $r(w) \leq r(u)$, by definition, w is at a distance at most $2r(u)$ from u. For each such link $e' = (w, z) \in S_1$, consider a disk of size $\frac{\alpha r(u)}{2}$ drawn around w; any pair of these disks must be non-overlapping, since the corresponding edges will interfere with each other otherwise. We can *pack* at most $O(1)$ such non-overlapping disks of radius $\frac{\alpha r(u)}{2}$, such that all their centers lie within a distance of $2r(u)$ from u. Hence, $|S_1| = O(1)$, which completes the proof of the lemma.

■

Let $C \doteq \max_e C(0.5, e)$. Define D to be the dilation of the paths, or the maximum number of links in a path traversed by a packet.

Lemma 54 $OPT = \Omega(C + D)$

Proof Every schedule needs at least D time slots to schedule the packet with the longest path; hence, we just need to verify that $OPT = \Omega(C)$. This follows from Lemma 53: at any time instant, the set of edges on which packets can be transmitted simultaneously forms an interference-free set. From Lemma 54, for any edge $e = (u, v)$, at most $O(1)$ edges can be simultaneously used within $I_{\geq}(0.5, u)$. Therefore, $\Omega(C(0.5, e))$ timesteps are needed to transmit all the packets which use the edges in the set $I_{\geq}(0.5, e)$. ■

We now extend the notion of inductive edge-ordering and provide the equivalent of the crucial Lemmas 53 and 54 for the Tx-Rx model of interference. The rest

of the proofs (for all the other lemmas) as well as the centralized and distributed scheduling algorithms remain the same under these two models.

Tx-Rx Interference model: Recall the Tx-Rx interference model: let $e_1 = (u, v) \in E$ be an edge along which there is a transmission. Let D denotes the network distance (in terms of hop-count) between the edges and nodes in the network. Specifically, for any two edges e_1 and e_2, $D(e_1, e_2)$ is defined as the least hop-count distance between an incident node of e_1 and an incident node of e_2. The transmission along e_1 is successful if and only if any other transmission along an edge $e_2 \in E$ is such that $D(e_1, e_2) > 1$. We need to redefine our notation for this model. For edge $e = (u, v)$, define $r(e) = \max\{r(u), r(v)\}$. For a vertex v, define $I_{\geq}(v) = \{e' \mid D(v, e') \text{ or } D(e', v) \leq 1,\ r(e') \geq r(v)\}$ and $I_{\geq}(e) = \{e' \mid D(e, e') \text{ or } D(e', e) \leq 1,\ r(e') \geq r(e)\}$. For any fixed constant α, define $I_{\geq}(\alpha, v) = \{e' \mid D(v, e') \text{ or } D(e', v) \leq 1,\ r(e') \geq \alpha \cdot r(v)\}$ and $I_{\geq}(\alpha, e) = \{e' \mid D(e, e') \text{ or } D(e', e) \leq 1,\ r(e') \geq \alpha \cdot r(e)\}$.

As before, let $H(e, i)$ denote the number of times packet i visits edge e. For any fixed value α, define $C(\alpha, e)$ to be the total number of times the edges in the set $I_{\geq}(\alpha, e)$ is used by the packets: i.e., $C(\alpha, e) = \sum_{e' \in I_{\geq}(\alpha, e)} \sum_i H(e', i)$. Let $C \doteq \max_e C(0.5, e)$. D is still defined to be the dilation.

Lemma 55 *For any vertex v and a fixed constant $\alpha > 0$, the size of the largest interference-free set of edges in the subgraph induced by $I_{\geq}(\alpha, v)$ is at most a constant whose value depends only on α.*

Proof The proof is via a packing argument. Suppose $S = \{e_1, \ldots, e_k\}$ is an interference-free set of edges in $I_{\geq}(\alpha, v)$. Let $S_1 \subseteq S$ be the set of edges such that $D(v, e') \leq 1$ and let $S_2 = S \setminus S_1$ (i.e., if $e \in S_2$, then, $D(e', v) \leq 1$). We will show that both $|S_1|$, $|S_2|$ (and hence, $|S|$) $= O(1)$.

We first show that $|S_1| = O(1)$. Recall that $r(v)$ is the transmission range of v. Consider the disk $\mathcal{W}$ of radius $(1 + \frac{3\alpha}{2})r(v)$ centered at vertex v. We will show that

each edge $e \in S_1$ "occupies" a disjoint space of area of at least $\Omega((\alpha r(v))^2)$ in $\mathcal{W}$. Hence, $|S_1| \leq O((\frac{1}{\alpha} + \frac{3}{2})^2) = O(1)$ as claimed. Consider an edge $(p, q) \in S_1$. There are two possible cases.

Case 1: $r(p), r(q) \geq \alpha \cdot r(v)$. In this case, assume w.l.o.g. that $D(v, q) \leq 1$. Hence q is within a distance of at most r from v.

Case 2: $r(p) < \alpha \cdot r(v)$ and $r(q) \geq \alpha \cdot r(v)$. Since $\min(D(v, p), D(v, q)) \leq 1$, node q is at most a distance of $(1 + \alpha) \cdot r(v)$ away from v. (The third case where $r(q) < \alpha \cdot r$ and $r(p) \geq \alpha \cdot r$ is identical to the second, and is ignored).

In both the cases, consider a disk of radius $\frac{\alpha \cdot r(v)}{2}$ around q. We say that edge (p, q) "occupies" this region of area $\frac{\pi(\alpha \cdot r(v))^2}{4}$. Crucially, no other edge $(f, g) \in S_1$ can "occupy" any of this region (otherwise q is within the range of either f or g and hence $\min\{D(q, (f, g)), D((f, g), q)\} \leq 1$, which violates the interference constraint). Hence the claim $|S_1| = O(1)$ follows.

We now show that $|S_2| = O(1)$. We first note that for any edge $(p, q) \in S_2$, $D((p, q), v) \leq 1$. W.l.o.g., we assume that $D(q, v) \leq 1$. Consider an arbitrarily large disk centered centered at v which contains all the vertices in the network. Consider a sector which subtends an angle of 60 degrees at v. Let $d_e(x, y)$ denote the Euclidean distance between points x and y. For any two points x, y which lies in this sector, it is easy to see using simple geometry that, $d_e(x, y) \leq \max\{d_e(x, v), d_e(y, v)\}$. Hence, if two nodes q and f exist in this sector such that edges (q, v) and (f, v) are both present, then there is an edge between these two nodes. In particular, this implies that at most one edge from $|S_2|$ can have an end point in the sector (without violating the interference constraint). Since the disk can be partitioned into at most six disjoint sectors of angle 60 each, $|S_2| \leq 6$. This completes the proof of the lemma. ■

Lemma 56 $OPT = \Omega(C + D)$

Proof We just need to verify that $OPT = \Omega(C)$. This follows from Lemma 53: at any time instant, the set of edges on which packets can be transmitted simultaneously forms an interference-free set. From Lemma 55, for any edge $e = (u, v)$, at most $O(1)$ edges can be simultaneously used within $I_{\geq}(0.5, u)$ and within $I_{\geq}(0.5, v)$. Therefore, $\Omega(C(0.5, e))$ timesteps are needed to transmit all the packets which use the edges in the set $I_{\geq}(0.5, e)$. ■

We are now ready to present our distributed algorithm for solving EPSI on disk graphs. Figure 7.1 contains a description of `DiskEPS`, our distributed algorithm for solving the $EPSI$ problem on disk graphs. The intuition behind this algorithm as well as the sequential version of this algorithm is as follows. As in the scheduling algorithm of Leighton *et al.* [83], we construct an *invalid* schedule $\mathcal{S}'$ by giving a random delay at the origin of each packet, and then letting it zip through its path, one hop at a time. These hops are in general invalid, since the schedule does not respect the interference constraints. However, due to the initial random delays, we show that only a logarithmic number of other transmissions contend with a particular transmission at any fixed time: this allows us to *expand* each invalid hop into polylogarithmic time slots and create a valid schedule. We note that while *inductive scheduling* allows us to expand each invalid hop into at most $O(\log n)$ time slots, the distributed scheduling algorithm based on Luby's distributed graph coloring algorithm [89] expands each invalid hop into at most $O(\log^2 n(1 + \log\lceil\frac{r_{\max}}{r_{\min}}\rceil))$ slots, yielding these two values for their respective approximation ratios.

Below, we first state the Chernoff-Hoeffding tail bounds and a variant of it for negatively correlated random variables, which will be useful in our analysis.

Fact 57 ***([30, 58, 98])*** *Given independent or negatively correlated r.v.s $X_1, \ldots, X_t \in [0, 1]$, let $X = \sum_{i=1}^{t} X_i$ and $\mu = \mathbf{E}[X]$, then for any $\delta > 0$, $\Pr[X \geq \mu(1+\delta)] \leq G(\mu, \delta)$, where $G(\mu, \delta) = \left(e^{\delta}/(1+\delta)^{1+\delta}\right)^{\mu}$.*

Algorithm DISKEPS

1. Each packet p_i chooses a delay Y_i uniformly at random from $\{1, \ldots, cX_0\}$ ($c > 0$ is a specific constant and $X_0 = C + D$), and waits for $Y_i c \log^2 n(1 + \log\lceil \frac{r_{\max}}{r_{\max}} \rceil)$ slots at s_i.

2. From the end of time Y_i, packet p_i moves along path P_i.

3. p_i reaches the jth node v on P_i by time $Y_i + j \cdot c \log^2 n(1+\log\lceil \frac{r_{\max}}{r_{\max}} \rceil)$ (with the source node being considered as the 0^{th} node). If it reaches before this time, it remains *inactive* and becomes *active* only after this time slot.

4. Let T be a multiple of $c \log^2 n(1 + \log\lceil \frac{r_{\max}}{r_{\max}} \rceil)$. All packets are moved to their next hop from their current location at time T, during the time interval $[T+1, \ldots, T + c \log^2 n(1 + \log\lceil \frac{r_{\max}}{r_{\max}} \rceil)]$ by the following steps.

   (a) Let E_T be the set of (active) edges on which active packets await transmission, after time slot T. Run subroutine `LubySched` below in $c \log^2 n(1 + \log\lceil \frac{r_{\max}}{r_{\max}} \rceil)$ time, which moves all packets to their next hop (Lemma 59 guarantees that this happens w.h.p.).

   (b) Remove packet p_i once it reaches t_i.

Subroutine LUBYSCHED

- Do the following for stages $j = 1 + \lceil \log \frac{r_{\max}}{r_{\min}} \rceil, \ldots, 1$. Only active edges in the set $E_j = \{e \mid r(e) \in (2^j, 2^{j-1}]\}$ participate during stage j.

  – Do the following for phases $i = 1, \ldots, c_1 \log n$. Each of these phases if of length $c_2 \log n$ time slots.

    * At the beginning of each phase, every currently active edge e (which participates in this stage) picks a random slot $slot(e) \in \{1, \ldots, c_2 \log n\}$.
    * Attempt the transmission of the packet along edge e during $slot(e)$ of this phase. If the transmission is succeeds, the edge is marked *inactive*. Else, it remains active and attempts to retransmit at a later phase. Packets which remain active till the end of their stages are dropped from the network. Lemma 59 guarantees that all edges successfully transmit their packets and become inactive at the end of their stage w.h.p.

Figure 7.1: **Distributed algorithm for solving $EPSI$ problem on disk graphs.**

We now show the performance of our algorithm using the following claims.

Lemma 58 *At any T (which is a multiple of $c\log^2 n(1+\log\lceil\frac{r_{\max}}{r_{\min}}\rceil)$) and for any fixed index $j \in [1,\ldots,1+\lceil\log\frac{r_{\max}}{r_{\min}}\rceil]$, define the (undirected) interference graph $I(T,j)$ as follows: the vertex set of this graph consists of the active edges at the end of slot T whose radii fall in the range $[2^j\cdot r_{\min}, 2^{j-1}\cdot r_{\min}]$ and two vertices have an edge between them if their corresponding edges cannot both simultaneously transmit without violating the interference constraint. We note that $I(T,j)$ is a random graph whose structure depends on the (random) set of transmissions which needs to be scheduled at the end of slot T. The maximum degree of any node in $I(T,j)$ is $O(\log n)$ w.h.p.*

Proof Consider an edge e of radius r. Let F be the set of all transmissions which need to be scheduled on all the edges in the set $I_{\geq}(0.5,e)$. Let $X_f(e,T)$ be the indicator random variable which denotes if edge e is active at the end of time T due to transmission f. By definition $C \geq |F|$. Since each packet chooses a random value X in the set $\{0,\ldots,c\cdot(C+D)\}$ and becomes active at its j^th hop exactly after time $(X+j)\cdot c\log^2 n(1+\log\lceil\frac{r_{\max}}{r_{\min}}\rceil)$, for any transmission f, we have $\mathbf{E}[X_f(e,T)] = \Pr[X_f(e,T)=1] =\leq \frac{1}{c\cdot(C+D)}$. By choosing a suitably large constant c, we have, $\mathbf{E}[\sum_{f\in F} X_f(e,T)] \leq 1$. Since, two transmissions f_1 and f_2 from the same packet on a fixed edge cannot both be active simultaneously, $X_{f_1}(e,T)$ and $X_{f_2}(e,T)$ are negatively correlated with each other. Otherwise, if f_1 and f_2 belong to different packets, their values are independent. Hence, using Fact 57, we have $\Pr[\sum_{f\in F} X_f(e,T) > O(\log n)] \leq \frac{1}{n^\delta}$, where δ is a constant which can be made arbitrarily large by choosing the other constants appropriately. ■

Lemma 59 *Subroutine* ***LubySched*** *runs in time $O(\log^2 n(1+\log\lceil\frac{r_{\max}}{r_{\min}}\rceil))$ and moves all the packets on the active edges E_T to their next hop successfully, w.h.p.*

Proof Consider a specific transmission f along an edge e during stage j of subroutine `LubySched`. We first observe that, by lemma 58, the total number of transmis-

sions which interfere with f during its stage j is at most $O(\log n)$ w.h.p. During each phase of stage j, edge e and all other edges participating in stage j choose random slots in the range $[1, \ldots, c_1 \log n]$. Hence during a specific phase, the probability of transmission f not being successful is at most $(1 - \frac{1}{c_1 \log n})^{O(\log n)} \leq \frac{1}{e}$, if constant c_1 is suitably large. Hence, after $c_2 \log n$ phases, all transmissions can be guaranteed to be successful w.h.p. (specifically, with probability at least $1 - \frac{1}{2^\delta}$, where δ can be made an arbitrarily large constant by choosing an appropriate value for the constants c_1 and c_2). ■

Theorem 60 *The distributed algorithm* `DiskEPS` *runs in time* $O((C + D) \log^2 n(1 + \log\lceil \frac{r_{\max}}{r_{\min}} \rceil))$ *and delivers all packets to their corresponding destinations successfully, w.h.p.*

Proof This theorem follows from the fact that each packet waits at most $O((C + D) \log^2 n(1 + \log\lceil \frac{r_{\max}}{r_{\min}} \rceil))$ at its source and Lemma 59 which guarantees that each packet traverses each of its hop in time $O(\log^2 n(1 + \log\lceil \frac{r_{\max}}{r_{\min}} \rceil))$ w.h.p. ■

Theorem 61 *The sequential version of the scheduling algorithm runs in time* $O((C + D) \log n)$ *and delivers all packets to their corresponding destinations successfully, w.h.p.*

Proof As noted earlier, the sequential algorithm first creates an invalid scheduled $\mathcal{S}'$ as follows: each packet i chooses a random delay in the range $\{1, \ldots, c(C + D)\}$. After its initial delay, each packet zips through its path, one hop at every single step. Consider any step T and the set of transmissions F_T scheduled at time T in the invalid schedule. We now show how to *expand* this invalid step into $O(\log n)$ time slots such that all transmissions in F_T can be scheduled without interference w.h.p.

Consider a transmission $f \in F_T$ along an edge e. We first observe that the total number of transmissions in F_T which use edges in the set $I_{\geq}(1.0, e)$ is at most

$O(\log n)$. The proof of this claim is identical that of Lemma 58. This leads to the following inductive coloring scheme. Consider a time frame $\Gamma = \{1, \ldots, k \log n\}$, where k is suitably large. Sort the transmissions in F_T according to the ranges of the transmitting end-points of their edges. We process the transmissions in F_T in this sorted order. For the current transmission f along edge e, we assign it the first feasible time slot in Γ such that f does not interfere with any of the other transmissions which have currently been scheduled. Observe that, since at most $O(\log n)$ transmissions in the set $I_{\geq}(1.0, e)$ contend with f w.h.p., this assignment schedules all transmissions in F_T in $O(\log n)$ time w.h.p. This completes the proof of the theorem. ■

7.5.2 Unit disk graphs

When all nodes have the same range (and hence all disks have the same radius), we obtain significant improvements in the approximation guarantee. By a repeated planar decomposition, we can derive an $O(1)$ approximation. This decomposition only requires a sparsity condition rather than geometry, and can be applied to bounded genus graphs also. We then obtain an $O(\log n)$ distributed algorithm by refining the analysis of the algorithm DiskEPS in Section 7.5.1. Finally, we give a distributed algorithm in the asynchronous model with a $O(\log^2 n)$ approximation guarantee.

An $O(1)$ approximation algorithm

Let the common range for all the nodes be one unit. Let B be a bounding box in the plane for the points in V. If we assume that G is a connected graph, B must have sides of length $O(n)$. Let B_k^0 be a partition of B into smaller grid cells, each cell having dimensions $k \times k$. Let B_k^1, B_k^2 and B_k^3 be the partitions obtained by translating the grid B_k^0 by $k/2$ along the x axis, the y axis, and both x and y axes respectively. A cell in these partitions will refer to one of the $k \times k$ sized pieces in it, and a point in

these partitions is any lattice point with integer coordinates. We will denote lattice points within by small letters and the $k \times k$ cells within by capital letters.

For a disk S of radius 1 in the plane, let $C(S)$ be the number of paths P_i that visit some vertex $v \in V$, located within S. Define $C = \max_S\{C(S)\}$. As before, D is the length of the longest path; $\max\{C, D\}$ is still a lower bound on the optimal size, and as the following observation shows, even if C is defined as the maximum of $C(D(x))$ for points $x \in B$, $C + D$ is still at least a constant factor of the optimal.

The main intuition for the partitioning algorithm is the following. After the initial step of giving random delays, both C and D become $O(\log n)$ within each time frame. This means that the smaller scheduling subproblem in any frame is localized to a $O(\log n) \times O(\log n)$ region of the plane. Thus, in addition to the temporal decomposition, we are able to do a *spatial decomposition* as well. If we carry this process once more, we end up with scheduling problems on regions of size $O(\log \log n) \times O(\log \log n)$, and at this point, we can solve the *subsubproblems* by brute force in $poly(n)$ time. The algorithm is described in Figure 7.2 and is called **Algorithm** UnitDiskEPS. Subroutine Partition(m) in Figure 7.2 creates a partition of the problem, originally on an $m \times m$ grid, into smaller subproblems, each on a grid of dimensions $2 \log m \times 2 \log m$.

Lemma 62 *There exists a choice of random delays for all the packets in step 1 of subroutine* Partition(m) *which satisfies the following property: for any time frame T of length $\log m$, and for any lattice point p in the input to the subroutine, the number of paths visiting some vertex $u \in V$ located in $S(p)$ is $O(\log m)$.*

Proof The input points lie in a $m \times m$ grid. By our assumption that each path visits only a constant number of vertices within $S(v)$ for any v, it follows that the largest path, D, is $O(m)$. Let $X_0 = C + D$. Consider any grid point v, and any time t.

Pr[packet p_i passes through D(v) at t] $\leq \frac{1}{cX_0}$ and E[# packets through D(v) at t] $\leq$

Algorithm UNITDISKEPS

1. Run subroutine PARTITION(n) to create smaller problems on $2\log n \times 2\log n$ sized grids.
2. For each of the subproblems on a $2\log n \times 2\log n$ sized grid, run subroutine PARTITION($2\log n$) to create smaller subproblems on $O(\log\log n) \times O(\log\log n)$ sized grids.
3. Solve the scheduling problem within a $O(\log\log n) \times O(\log\log n)$ sized grid by exhaustive search (details in Lemma 64).
4. Combine the schedules for all the subproblems together to form the whole schedule.

Subroutine PARTITION(m)
INPUT A scheduling instance on a $m \times m$ region.
OUTPUT Partition this instance into smaller scheduling problems, defined on grid cells of size $2\log m \times 2\log m$.

1. Construct an invalid schedule $\mathcal{S}_1(\Pi)$ in the following manner:
   (a) For each packet, choose a random delay from $\{1, \ldots, c(C + D)\}$, where $c > 0$ is a specific constant, such that Lemma 62 is satisfied (the property in Lemma 62 can be checked in polynomial time; so this step involves choosing the random delays, checking the property and repeating if necessary).
   (b) Allow each packet to zip through along its path, after waiting for the random delay at the source.
2. Partition B into grids $B^0_{2\log m}$, $B^1_{2\log m}$, $B^2_{2\log m}$, and $B^3_{2\log m}$.
3. Consider successive time frames of length $\log m$.
4. For each time frame T of size $\log m$, assign each packet p_i to a unique cell Z in $B^0_{2\log m}, B^1_{2\log m}, B^2_{2\log m}$, or $B^3_{2\log m}$ such that the path traversed by p_i during T lies completely within Z; break ties arbitrarily.
5. For each time frame T, for each cell Z in $B^0_{2\log m}, B^1_{2\log m}, B^2_{2\log m}$, and $B^3_{2\log m}$, the problem restricted to Z involves scheduling the packets assigned to it during T, along the segments of the paths within Z.

Figure 7.2: **Algorithm for solving $EPSI$ problem on unit disk graphs.**

1. By the Chernoff bound, the number of paths visiting vertices in $D(v)$ during a time frame T of length is $O(\log n)$ with probability at least $1 - \frac{1}{m^c}$, for some constant $c > 0$. Since there are $O(m^2)$ grid points, and $O(m)$ time frames to consider, the lemma follows by the union bound. ■

Lemma 63 *In step 4 of subroutine* PARTITION(m)*, the path traversed by any packet p_i during a time frame T (of length $\log m$) can be uniquely assigned to some cell in $B^0_{2\log m}$, $B^1_{2\log m}$, $B^2_{2\log m}$, or $B^3_{2\log m}$.*

Proof Let $B_{\log m}$ be a partition of B. Any packet p_i during a time from T traverses a sub-path of length at most $\log m$. Any such path straddles at most four cells in $B_{\log m}$ and all such cells are adjacent to each other. Clearly, these four cells of side $\log m$ each are together contained in some cell in $B^0_{2\log m}$, $B^1_{2\log m}$, $B^2_{2\log m}$, or $B^3_{2\log m}$ which proves the lemma. ■

Lemma 64 *A schedule of length $O(\log\log n)$ can be constructed for the scheduling problem on a grid of size $O(\log\log n) \times O(\log\log n)$.*

Proof By our assumption, each packet can only traverse $O(\log\log n)$ steps within such a grid cell; so $D = O(\log\log n)$. Also, by the guarantees of the subroutine PARTITION, $C = O(\log\log n)$ within such a grid. Therefore, the total number of packets is $O((\log\log n)^3)$. The maximum number of possible schedules is this number raised to the power of C, which is at most a polynomial in n. Therefore, we can try out all schedules in polynomial time. ■

The above arguments lead to the following theorem.

Theorem 65 *A schedule for the EPSI problem on unit disk graphs of length $O(1)$ times the optimal can be found in polynomial time.*

Distributed algorithms

Synchronous model Algorithm DiskEPS detailed in Figure 7.1 for disk graphs can be modified to yield a better bound of $O(\log n)$ for the case of unit disk graphs. Since all disks have the same radius, the notation and ordering of Section 7.5.1 is not needed. We will use the lower bounds C, D defined in the previous subsection.

The first three steps of the algorithm DiskEPS are unchanged, except for the delays and time frames being multiples of Ψ (whose value is to be specified shortly). In step 4 of the algorithm, instead of running algorithm LubySched, we run Luby's vertex coloring algorithm [89]. The vertex coloring algorithm runs on a graph H constructed as follows: for each edge $e \in E_T$, add a *vertex* v_e in H; if two edges $e_1, e_2 \in E_T$ interfere with each other, then add the edge (v_{e_1}, v_{e_1}) in H. This takes $O(\log n)$ steps and uses $O(\Psi)$ colors. Here, Ψ is the maximum degree of any node in H. Recall the definitions of $I_{\geq}(u)$, C, and D from Sections 7.4 and 7.5.

Lemma 66 $\Psi = O(\log n)$.

Proof Let $e = (p, q)$ and let v_e be a vertex with maximum degree Δ in H. For any u, let $I_u = I_{\geq}(u) \bigcap E_T$. For a specific node v, how large can $|I_v|$ be? I_v comprises of precisely the set of edges which carry packets during a particular time slot in the invalid schedule (obtained after the initial random delays, and letting the packets zip through the network). Since the maximum initial random delay is $\Omega(C)$, Chernoff bounds imply that $|I_v| = O(\log C) = O(\log n)$ w.h.p. Hence, $O(\max_u |I_u|) = O(\log n)$ w.h.p. Finally, we observe that, by definition, $\Psi \leq |I_p| = O(\max_u |I_u|)$. Hence, $\Psi = O(\log n)$. This completes the proof of the lemma. ■

Lemma 67 *The above distributed algorithm constructs a valid schedule for the packet-scheduling problem on unit disk graphs such that all packets reach their destination in* $O((C + D) \log n)$ *time w.h.p.*

Proof Luby's coloring algorithm yields a valid coloring in $O(\log n)$ time w.h.p. Hence, every packet advances one hop towards its destination, without interference, in every time frame of length $O(\log n)$ w.h.p. Since the maximum initial random delay is $O(C + D) \log n$, the lemma follows. ■

Asynchronous model The algorithm described in the previous section needs centralized, synchronous control, which is difficult in practice. We now describe a completely distributed, asynchronous, randomized algorithm that gives a schedule of length at most $O(\log^2 n)$ times the optimal, with high probability.

The basic idea is to combine contention resolution methods along with the random delays plus coloring techniques that have been used so far. Note that if there are C packets in the vicinity of some packet p, that are contending for a transmission slot at a time, all of these can be scheduled in $O(C \log n)$ steps with high probability. The random delays step allows us to reduce the effective congestion at every step, and after that one can perform coloring via the contention resolution. Note that we need to simulate some sort of synchronization, to ensure that the right set of packets is contending at any time, and this can easily be achieved by suitable waiting for polylogarithmic steps for each packet at each edge. The algorithm is described in Figure 7.3 and is referred to as **Algorithm** AsynchronousUnitDiskEPS.

Lemma 68 *Each packet p_i moves on its ℓth edge during the interval $T_{i,\ell}$ w.h.p.*

Proof Consider packet p_i during interval $T_{i,\ell}$. The initial random delays ensure that there are at most $O(\log n)$ other packets which contend with this packet during this time interval (see Lemma 66). Since packets attempt to transmit at any time slot with probability $\frac{1}{\alpha_3 \log n}$, and there are $O(\log n)$ contending packets, packet p_i will be successful at time t with probability $\frac{1}{\alpha_3 \log n} \times (1 - \frac{1}{\alpha_3 \log n})^{O(\log n)}$ which is $\Omega(\log n)$. Since $T_{i,\ell}$ has $\alpha_2 \log^2 n$ time slots, packet p_i transmits successfully across its ℓ^{th} edge with high probability. The lemma now follows by a union bound. ■

Algorithm AsynchronousUnitDiskEPS

1. Each packet p_i chooses a delay Y_i uniformly at random from $\{1, \ldots, \alpha_1 X_0\}$, where $\alpha_1 > 0$ is a constant and X_0 is as defined before.

2. Each packet waits at its source for $(\alpha_2 Y_i \log^2 n)$ steps, where α_2 is a constant.

3. Packet p_i traverses its ℓ^{th} edge during the time interval $T_{i,\ell} = [\alpha_2 Y_i \log^2 n + (\ell - 1)\alpha_2 \log^2 n + 1, \ldots, \alpha_2 Y_i \log^2 n + \ell \alpha_2 \log^2 n]$ as follows:

   (a) Let $t \in T_{i,\ell}$ denotes the current time.

   (b) If packet p_i has already traversed its ℓ^{th} edge, then it keeps waiting till the end of the interval $T_{i,\ell}$.

   (c) If p_i has not yet traversed its ℓ^{th} edge, it chooses to traverse this edge at time t with probability $\frac{1}{\alpha_3 \log n}$, where α_3 is a constant.

   (d) If there is a collision during time t, p_i retries this at the next time step.

Figure 7.3: **Asynchronous distributed algorithm for solving $EPSI$ problem on unit disk graphs.**

Corollary 69 *All packets are delivered within time* $O(OPT \log^2 n)$ *w.h.p where* OPT *is the length of the optimal schedule.*

Chapter 8

Algorithmic Aspects of Capacity in Wireless Networks

8.1 Introduction

Two central questions in communication networks are: what is the throughput capacity of the network, and how can one utilize the network close to the capacity? In other words, given a collection of source-destination pairs $\{(s_i, t_i)\}$, what is the maximum rate (throughput) at which the network can transfer data from the sources to their corresponding destinations? There are many factors effecting this question such as interference, fairness and energy constraints. For a wired network, some of these constraints can be formulated easily as a simple linear program (LP), but this problem is non-trivial to solve in the case of wireless networks due to interference. An influential result on the capacity of wireless networks is that of Gupta and Kumar [53]. They show that, given n identical randomly distributed nodes on a unit square, with each node having an independent randomly chosen destination, the uniform per node throughput capacity in bit-meters/second, is $\Theta(\frac{1}{\sqrt{n \log n}})$: this is sub-linear in the number of nodes, in contrast with the wired setting. Several extensions of the basic

result have recently been considered, see Section 8.8 for additional discussion.

Here, building on the earlier results in [72], [61], and Chapter 7 of this thesis, we study the algorithmic aspects of both the inter-related questions posed earlier, namely: (i) What is the maximum throughput capacity of the network? (ii) How to design network protocols that jointly route the packets and schedule transmissions at rates close to the maximum throughput capacity. *In contrast to the results in [53], we focus on (a) arbitrary instances rather than just random node distributions, (b) allow nodes to have varying transmission ranges instead of uniform ranges, (c) consider not only the uniform node-throughput metric but other natural linear functionals of node throughputs and (d) consider linear constraints such as total energy consumed and path length.*

Key technical contributions of our work include a novel definition for congestion which captures the central properties of wireless interference, linear models for a wide class of wireless throughput maximization problems, and the notion of rate competitiveness of scheduling algorithms. Our techniques can accommodate a variety of path selection constraints such as low energy, low hop-count, etc. The algorithmic and analytical techniques introduced here are applicable to a variety of interference models and could be of independent interest. Our main contributions in this chapter are as follows:

1. Given an arbitrary wireless network $G = (V, E)$, where each node can have a different transmission range, and a set of k arbitrary source-destination pairs, we describe a polynomial time approximation algorithm that computes the maximum achievable throughput in G to within a constant factor. Thus, this is an algorithmic version of the Gupta and Kumar [53] result, and gives a way of quantifying the capacity for an *arbitrary* wireless network. In contrast the work of [53] focuses on a random wireless network. As noted and empirically shown in [61], the capacity of an arbitrary wireless network could differ substantially from a random network.

Our results are based on a linear programming formulation of the problem and crucially uses the properties of wireless interference models. The results hold for various interference models and for variable power levels at individual transceivers. A new stability measure is introduced that provably bounds the optimum capacity to within constant factors.

2. Our approach allows us to incorporate the per flow end-to-end fairness constraints in the throughput maximization problem. The resulting LP formulation can enforce any given (long term) fairness objective; our scheduling algorithm guarantees that the total throughput is within a constant factor of the optimal, for such a fairness constraint. As radio devices become cheaper and smaller, sensor and ad hoc networks are becoming more and more prevalent. This is leading to a new challenge: how to design protocols such that radio devices do not drain battery power very fast. There has been much research on all aspects of routing and scheduling with low energy requirements; see, e.g., [121, 66] and the references therein. Our approach makes it very simple to add energy constraints into the formulation: given such a bound on the energy, we aim to maximize throughput. In fact, we can add any set of requirements that can be modeled by linear constraints. Our LP-formulations differ from the formulations presented in [61, 72] as follows: although [72] presents constant factor approximate LP formulations for a class of scheduling problems, they do not handle wireless interference constraints in their formulations, thus limiting the utility of their approach to most realistic wireless network scenarios. The approach of [61] can model arbitrarily complex interference models; however, they do not discuss how close their computed throughput is to the optimal throughput. Our modeling techniques overcome both these limitations.

3. We also study the empirical performance of a natural class of congestion-aware path selection strategies which arise from our LP formulations. Since these heuristics essentially involve computing shortest paths using congestion-aware link metrics,

standard routing protocols such as AODV can be easily modified to incorporate such link metrics. This yields a *unified protocol for MAC, routing and (aspects of) transport layers in a wireless network* which gives good throughput and does not require too many changes from existing routing protocols. The algorithm of [72] involves multiple phases (as does that of [46]), and cannot be easily used this way. As shown in [13, 20, 113, 124], there is a significant interaction between individual layers of a OSI protocol stack and plugging in optimal protocols for each layer does not lead to optimal overall performance [20]. As a result, recent work has focused on designing unified protocols for wireless networks [112]. The unified routing+scheduling protocols developed here overcome this performance loss.

4. We perform extensive simulations to study the performance of our algorithm and the shortest path heuristics. We obtain explicit tradeoff between fairness and total throughput, which shows the increase in throughput with decreasing fairness requirement; while this behavior is completely expected, our results also *quantify* this tradeoff, i.e., by what fraction does the throughput increase for a given loss in fairness. Similarly, we also study such a relationship between the energy consumed and throughput. Thus, our results provide a formal way of quantifying the tradeoffs between different constraints for wireless networks. Our simulations also indicate that routes obtained using our congestion-aware shortest path heuristics have much better throughput in every instance than the routes obtained using the hop-count based shortest path algorithm.

8.2 Preliminaries

This section contains basic definitions and concepts used in the rest of the chapter. For the sake of completeness, we recollect much of the definitions and assumptions made in Chapter 7. We consider multi-hop wireless networks. The network is modeled

as a directed graph $G = (V, E)$. The nodes of the graph correspond to individual transceivers and a directed edge (u, v) denotes that u can transmit to v directly. Each edge in $G = (V, E)$ has a capacity $c(e)$ bits/sec and denotes the maximum data that can be carried on e in a second. We assume that the system operates synchronously in a time slotted mode. Each time slot is τ seconds long. Thus, at most $\tau c(e)$ bits of information can be transmitted over link e during any time slot.

A schedule $\mathcal{S}$ describes the specific times at which data is moved over the links of the network. In other words, let $X_{e,t}$ be the indicator variable which is defined as follows:

$$X_{e,t} = \begin{cases} 1 & \text{if } e \text{ transmits successfully at time } t \\ 0 & \text{otherwise} \end{cases} \tag{8.1}$$

A schedule $\mathcal{S}$ is a $0-1$ assignment to the variables $X_{e,t}$, $e \in E$, $0 \leq t$. We will focus on periodic schedules in this work. A schedule $\mathcal{S}$ is periodic with period T, if $\forall e, t, i$: $X_{e,iT+t} = X_{e,(i+1)T+t}$. In wireless networks, links can be scheduled to transmit in the same time slot only if they do not interfere. The precise notion of interference will be made clear next. *For ease of exposition, we will assume that $c(e)$ is 1 and τ is also 1.* All the results generalize directly, when we relax these constraints.

Network and Interference Models We assume that vertices V are embedded in the plane $\mathbf{R}^2$. Each vertex (transceiver) u has an associated range denoted by $range(u)$. A necessary (but *not* sufficient) condition for a transceiver v to hear u is that v be within a distance $range(u)$ of u. Specifically, if transmission is not feasible from u to v either because v is outside the range of u or because of other reasons (such as the presence of an obstruction between u and v), then the edge (u, v) is not present in the graph $G = (V, E)$. This is an important consideration for modeling realistic network scenarios such as indoor wireless networks or even outdoor networks in the presence of obstructions. We assume that all antennas are omnidirectional although

generalization to directional antennas is possible, and is omitted here.

Since the medium of transmission is wireless, simultaneous transmissions on proximate edges may *interfere* with each other resulting in collisions. Formally, we say that edges $e_1, e_2 \in E$ interfere with each other if edges e_1 and e_2 cannot both transmit successfully during the same time slot. Let $I(e_1)$ denote the set of edges which interfere with edge e_1, i.e., e_1 cannot transmit successfully whenever an edge $e_2 \in I(e_1)$ is transmitting. An *Interference Model* defines the set $I(e_1)$ for each edge e_1 in the network. Several such models have been studied, because of variations in the underlying technology, protocol, etc. We consider two such models, namely, the Tx-model and Tx-Rx models of interference (see Chapter 7 for their definitions).

Network Flows Given a set of flows, with flow i starting at a source node s_i and ending at a destination node t_i, we will be concerned with the *rates* at which data can be sent along these flows. If the rate for flow i is r_i bits per second, then, on an average, in one time slot, r_i bits sent by s_i are received by t_i. In our LP formulations, rates for each flow i will translate to a per edge rate, $x(e, i)$ for edge e: this is the rate at which flow i is routed through edge e. As in [72], we assume an infinitesimally divisible flow model for data transmission - this leads to flow conservation constraints for the data. Let $\vec{x}$ denote the link flow rate vector; this vector associates a link rate $x(e)$ (the total rate of all flows on link e) with each link e. Recall that $X_{e,t}$ is the indicator variable which denotes if there was a successful transmission on link e during time t. By definition, as time $t' \to \infty$, we have $x(e) = \sum_{t \leq t'} X_{e,t}/t'$.

For the link scheduling and the end-to-end scheduling problems studied here, the central question we are concerned with is that of *stability*: a schedule is said to be stable if every packet incurs a bounded delay, and consequently, all buffers have bounded sizes. A stable rate vector is one for which there exists a stable schedule. In Section 8.3, given a link-rate vector $\vec{x}$, we will either show that $\vec{x}$ is not stable, or

show how to approximate an optimal schedule for vector $\vec{x}$ by a near optimal schedule with a slightly smaller throughput. This will serve as a useful building block for our end-to-end scheduling and throughput maximization techniques in Sections 8.4 and 8.5.

Two of the fundamental end-to-end throughput maximization problems we will consider are the maximum multicommodity flow problem (MFP) and maximum concurrent flow problem (MCFP) [1]. In MFP (as defined in the context of wired networks), given a directed graph $G(V, E)$ and a collection of source-destination pairs $\{(s_i, t_i)\}$, the goal is to find a stable end-to-end rate vector for the (s_i, t_i) pairs such that data can be injected into the network by the sources at these rates without violating individual edge capacities; the objective is to maximize the *total* rate of injection for these pairs; packets injected at such a rate can be scheduled in a wired network, since the only constraints are the edge capacities. Note that this formulation does not consider any notion of fairness among the different flow values; MCFP incorporates fairness by requiring that the total rate of injection be maximized subject to the constraint that all the (s_i, t_i) pairs have the same rate. We note that standard LP formulations exist for optimally solving both MFP and MCFP for wired networks. The problems we consider here for wireless networks are variations of these classical multi-commodity flow problems wherein, flow on the links that interfere with each other cannot be scheduled simultaneously. Thus the task of finding optimal multi-commodity flows in wireless networks becomes considerably more complicated.

8.3 Link-Flow Scheduling

In this section, we develop a link-flow scheduling algorithm to schedule a set of flows specified on the links of the network. We also develop necessary and sufficient conditions for link flow stability.

Link-Flow Stability: Necessary Conditions Recall that for an edge $e = (u, v) \in E$, $I(e)$ denotes the set of edges which interfere with e. We restate the definition for $I_{\geq(e)}$ from Chapter 7.

Definition 70 $I_{\geq}(e) = \{(p, q) : (p, q) \in I(e)$ *and* $d(p, q) \geq d(u, v)\}$.

$I_{\geq}(e)$ is the subset of edges in $I(e)$ which are greater than or equal to e in length. Recall that $X_{e,t}$ is the indicator variable which is 1 iff e transmits successfully during time t. We restate the following lemma from Chapter 7.

Claim 71 *(restatement of Claim 52) In any link schedule,*

$$\forall e \in E, \forall t \qquad X_{e,t} + \sum_{f \in I_{\geq}(e)} X_{f,t} \leq c \tag{8.2}$$

where c is a fixed constant that depends only on the interference model. In particular, for the Tx-model, the value of c is at most 5.

Let $\vec{x}$ be a link-flow vector. We define the *congestion* on a link e to be $c(e) = x(e) + \sum_{f \in I_{\geq}(e)} x(f)$. The following lemma imposes a simple necessary condition for link-flow stability.

Lemma 72 *Let c be the constant in Claim 52. $\vec{x}$ is a stable link-flow only if the following holds:*

$$\forall e \in E, \qquad x(e) + \sum_{f \in I_{\geq}(e)} x(f) \leq c$$

Proof Assume that the flow vector $\vec{x}$ is stable, i.e., there exists a stable schedule $\mathcal{S}$ which achieves the link-rates specified by $\vec{x}$. Let $X_{e,t}$ be the transmission indicator variable for this schedule for edge e and time t. As time $t' \to \infty$, we have $\sum_{t \leq t'} X_{e,t}/t' \to x(e)$, since $x(e)$ is the link-rate associated with edge e. The lemma

now follows by summing up equation (8.2) over time slots $[1, \ldots, t']$ and taking the average. ■

Link-Flow Scheduling Algorithm In this section we present both centralized and distributed algorithms for scheduling a link-flow vector $\vec{x}$ and analyze conditions under which this algorithm yields a stable schedule (and hence sufficient conditions for link-flow stability). The algorithm works as follows: time is divided into uniform and contiguous windows or *frames* of length w, where w is a sufficiently large positive integer. (We assume w.l.o.g. that w is such that for all e, $w \cdot x(e)$ is integral.) The algorithm employs a subroutine called *frame-scheduling* which specifies a schedule for each edge e within each frame. This schedule is repeated periodically for every frame to obtain the final schedule. We now present the details of the frame-scheduling algorithm whose pseudo-code is presented in **Algorithm 1**.

Consider a single frame W whose time slots are numbered $\{1, \ldots, w\}$. For each edge e, the subroutine assigns a subset of slots $s(e) \subseteq W$ such that the following hold:

1. $|s(e)| = w \cdot x(e)$, i.e., each edge receives a fraction $x(e)$ of time slots.
2. $\forall f \in I(e)$, $s(f) \cap s(e) = \Phi$, i.e., two edges which interfere with each other are not assigned the same time slot.

For all edges $e \in E$, the set $s(e)$ (set of time slots in W which are currently assigned to e) is initialized to Φ. Edges in E are processed sequentially in the non-increasing order of their lengths. Let the current edge being processed be e. Let $s'(e)$ denote the set of time slots in W which have already been assigned to edges in $I(e)$ (and hence cannot be assigned to e): $s'(e) = \bigcup_{f \in I_{\geq}(e)} s(f)$. In the remaining slots $W \setminus s'(e)$, we choose any subset of $w \cdot x(e)$ time slots and assign them to $s(e)$.

Lemma 73 *Algorithm 1 produces a conflict-free schedule, i.e., for any two interfering edges $e_1, e_2 \in E$, $s(e_1) \bigcap s(e_2) = \Phi$.*

Algorithm 4 SCHEDULE($\vec{x}, w$)

1: **for all** $e \in E$ **do**
2: $s(e) = \Phi$
3: **end for**
4: Sort E in non-increasing order of edge-lengths.
5: **for** $i = 1$ to $|E|$ **do**
6: $e = E[i]$
7: $s'(e) = \bigcup_{f \in I(e)} s(f)$
8: $s(e) =$ any subset of $(W \setminus s'(e))$ of size $w \cdot x(e)$
9: **end for**

Proof Assume w.l.o.g. that e_1 is processed before e_2 by the algorithm. Since e_1 and e_2 interfere, it follows that $e_1 \in I_{\geq}(e_2)$, and hence $s(e_1) \subseteq \bigcup_{f \in I_{\geq}(e_2)} s(f) = s'(e_2)$. Since, $s(e_2) \subseteq W \setminus s'(e_2)$, the lemma follows. ■ The following lemma proposes a sufficient condition for which the link-flow scheduling algorithm yields a *valid* schedule, i.e., sufficient number of slots are chosen for each edge within a frame.

Lemma 74 *The link-flow scheduling algorithm produces a valid schedule for $\vec{x}$ if the following holds:*

$$\forall e \in E, \qquad x(e) + \sum_{f \in I_{\geq}(e)} x(f) \leq 1.$$

Proof The schedule produced by the link-flow scheduling algorithm is stable if step 8 in Algorithm 8.3 is well defined, i.e., there are always $w \cdot x(e)$ slots available in the set $W \setminus s'(e)$. We now show that this is the case for all edges. Assume otherwise, i.e.,

there exists an edge e such that $|W \setminus s'(e)| < w \cdot x(e)$. Hence,

$$\begin{aligned} |W| &< |s'(e)| + w \cdot x(e) \\ &\leq |\bigcup_{f \in I_{\geq}(e)} s(f)| + w \cdot x(e) \\ &\leq \sum_{f \in I_{\geq}(e)} |s(f)| + w \cdot x(e) \\ &\leq \sum_{f \in I_{\geq}(e)} w \cdot x(f) + w \cdot x(e) \end{aligned}$$

Dividing both sides above by w and rearranging the terms, we have

$$x(e) + \sum_{f \in I_{\geq}(e)} x(f) > 1$$

which contradicts our assumption. This completes the proof of the lemma. ■

Suppose we have a set of end-to-end flows $\mathbf{f}_i$ between each $\{s_i, t_i\}$ pair. For each $e \in E$, let $x(e) = \sum_i \mathbf{f}_i(e)$ denote the total flow on link e where $\mathbf{f}_i(e)$ is the amount of flow i on edge e and let $\vec{x}$ denote the link-rate vector. If $\vec{x}$ satisfies the conditions of lemma 74, the following result shows that we get a stable schedule, i.e., each packet is delivered in a bounded amount of time.

Observation 75 *If the vector $\vec{x}$ above satisfies the conditions of lemma 74, each packet is delivered in at most Wn steps.*

Proof Assume that W is such that $W\mathbf{f}_i(e)$ is integral for each i and e. Consider any flow i. The number of packets injected for this flow during the window of size W is exactly $r_i W$. For each edge e, partition the $Wx(e)$ slots into $\mathbf{f}_i(e)W$ slots for each i. Then, it is clear that for each flow i, each packet will move along one edge in W steps. ■

8.4 Scheduling End-to-End Flows

In this section, we discuss efficient algorithms for scheduling end-to-end flows. Specifically, given a collection of paths and an associated rate vector R^*, our goal is to find a stable schedule whose rate is αR^*, where α is the scaling factor whose value we seek to maximize. The basic idea behind the end-to-end scheduling algorithm is as follows. Let the vector R^* induce a set of link flows x^*. Let κ denote the maximum congestion on any edge: i.e., $\kappa = \max_e(x^*(e) + \sum_{e' \in N_{\geq}(e')} x^*(e'))$. If $\kappa \leq 1$, then Lemma 74 implies that the induced link rate x^* (and hence the end-to-end rate R^*) can be stably scheduled by the link scheduling algorithm. Hence, we can achieve a stable end-to-end scheduling by simply repeating the link schedule periodically, with the period being the length of the frame. However, if $\kappa > 1$, then we scale the end-to-end rate vector (and hence the link rates too) by a factor κ^{-1}. Crucially, the new rates allow us to stably schedule the scaled end-to-end flows by repeating the link scheduling algorithm periodically. How good is the scaling factor of $\alpha = \kappa^{-1}$? We now turn to the notion of *competitiveness of scheduling algorithms* to answer this question.

Competitiveness of Scheduling Algorithms: We now introduce the notion of competitiveness for scheduling algorithms; as will be explained next, this metric plays a key role in understanding the *end-to-end* efficiency of such algorithms. Let $\mathcal{P}$ be a collection of paths, and for each $p \in P$, let $r(p)$ denote the rate associated with the path p. These end-to-end flows induce a link-flow vector $\vec{x}$ which specifies the a rate $x(e)$ for each edge $e \in E$: $x(e) = \sum_{p:e \in p} r(p)$. In general, the end-to-end flow vector may not be stable, i.e., there might not exist any scheduling algorithm which achieves the rates specified by the flow vector. Given a end-to-end flow scheduling algorithm $\mathcal{A}$, define its *throughput fraction* to be the maximum scalar value $q = q(\mathcal{A})$ such that a rate of $q \cdot r(p)$ can be scheduled by $\mathcal{A}$ for each $p \in \mathcal{P}$. Let q^* be the optimal throughput fraction, i.e., q^* is the maximum throughput fraction achievable

by any scheduling algorithm. The competitiveness of the scheduling algorithm $\mathcal{A}$ is defined as $q(\mathcal{A})/q^*$. The following lemma states that our scheduling algorithm is α-competitive where α is a constant (which depends only on the interference model). To our knowledge, this is the first such guarantee known; such a worst-case guarantee rigorously proves the utility of our algorithms.

Lemma 76 *The end-to-end flow scheduling algorithm is α-competitive where $\alpha > 0$ is a constant. For the Tx-model, the value of α is at least* 0.2*.*

Proof Given a link rate vector $\vec{x}$, let q^* be the optimal throughput fraction. Thus, the vector $q^*\vec{x}$ can be scheduled by an optimal scheduling algorithm. By Lemma 72, q^* is such that for all edges e, $q^*(x_e + \sum_{f \in I_{\geq}(e)} x(f)) \leq c$, where c is a constant. We now *scale down* this link rate vector by the scalar c to obtain the vector $\vec{y} = \frac{q^*}{c}\vec{x}$. Clearly, we now have for all edges e, $y(e) + \sum_{f \in I_{\geq}(e)} x(f) \leq 1$. Therefore by Lemma 74, the end-to-end scheduling algorithm can schedule vector $\vec{y}$. Hence the throughput fraction of algorithm $\mathcal{A}$ for the link vector $\vec{x}$ is at least $\frac{q^*}{cq^*} = \frac{1}{c}$. Since this is true for all vectors $\vec{x}$, the algorithm is α-competitive where $\alpha = \frac{1}{c} > 0$ is a constant. Since Lemma 72 implies that $c \leq 5$ for the Tx-model, $\alpha \geq 0.2$ for the Tx-model. This concludes the proof of the lemma. ■

8.5 Linear Programming Formulations

We now present the LP formulations for the maximum flow (MFP) and the maximum concurrent flow (MCFP) problems in wireless networks. See Section 8.2 for the relevant definitions. Let $C = \{1, 2, \ldots, k\}$ denote a set of *commodities*. For each commodity i, let s_i and t_i represent the source and destination for the commodity. Let $\mathcal{P}_i$ denote the set of all paths between source s_i and destination t_i of commodity i. For any $p \in \mathcal{P}_i$, let $r(p)$ denote the *data rate* associated with the path p: this is the

rate at which data is transferred from s_i to t_i along p. Let r_i denote the total rate at which data is source s_i injects packets for destination d_i: i.e., thus $r_i = \sum_{p:\ p \in \mathcal{P}_i} r(p)$. For any edge $e \in E$, $x(e)$ denotes the total rate at which data is transferred across edge e: i.e., $x(e) = \sum_{p:\ e \in p} r(p)$. As noted in Section 8.2, in MFP, we would like to maximize the sum of all rates r_i subject to the wireless interference constraints. In MCFP, we would like to maximize the sum of the rates r_i subject to the additional constraint that all the r_i's are equal.

We now present a generalized LP-formulation, called the MAXFLOWLP which captures both these problems by incorporating end-to-end fairness constraints. The central notion in this formulation is that of the *fairness index* λ. The fairness index $\lambda \in [0, 1]$ denotes the ratio between the minimum and maximum rates: $\lambda = \frac{\min_i r_i}{\max_i r_i}$. Note that λ equal to 0 and 1 correspond to the special cases of MFP and MCFP respectively. Our formulation is as follows:

$$
\begin{aligned}
\max \quad & \sum_{i \in C} r_i \quad \text{subject to} \\
\forall i \in C, \quad & r_i = \sum_{p \in \mathcal{P}_i} r(p) \\
\forall i \in C, \forall j \in C \setminus \{i\}, \quad & r_i \geq \lambda r_j \\
\forall e \in E, \quad & x(e) = \sum_{p:\ e \in p} r(p) \\
\forall e \in E, \quad & x(e) + \sum_{f \in I_{\geq}(e)} x(f) \leq 1 \\
\forall i \in C, \forall p \in \mathcal{P}_i, \quad & r(p) \geq 0
\end{aligned}
$$

We make the following observations about this LP formulation. First, we note that the stability conditions derived in Lemmas 72 and 74 are crucial for modeling the effect of interference in the LP and still guarantee a constant-factor performance ratio. This is a significant distinction between our techniques and those of [61] and

[72]: the former does not guarantee good performance bounds and the latter does not model wireless interference; Next, we observe that the size of this program may not be polynomial in the size of the network G as there could be exponentially many paths $\mathcal{P}_i$. However, using standard techniques, the same program could be equivalently stated as a polynomial-size network flow formulation [1]; we choose to present this standard formulation here for ease of exposition.

The first set of constraints define the total rate r_i for each commodity. The second set of constraints are the fairness constraint which ensure that the ratio between the minimum and maximum end-to-end rates is at least λ. The third set of constraints define the link rates $x(e)$ for each link e. The fourth set of constraints capture wireless interference. These constraints along with the end-to-end scheduling algorithm discussed in Section 8.4 ensure that the flows computed by the LP can be feasibly scheduled. Finally, the objective value of this LP is at most a constant factor away from an optimal solution: the optimal schedule induces a rate $x^*(e)$ on each link e; further the rates x^* also satisfy the conditions of Lemma 74. Hence, scaling down the optimal end-to-end rates and hence the link rates by a factor c (the constant which appears in Lemma 72) results in a feasible solution. The following lemma formalizes the last two observations.

Theorem 77 *The MAXFLOWLP formulation always results in a solution which can be stably scheduled. Further, the value of the objective function computed by the MAXFLOWLP is within a constant factor from the optimal solution to the corresponding flow problem. This factor is has a value of at most* 5 *for the Tx-model.*

Additional Constraints Any set of linear constraints can be added to the LP formulation. Recall that $x(e,i) = \sum_{p \in \mathcal{P}_i : e \in p} r(p)$ denotes the rate for flow i on edge e. Let $d(e)$ denote the length of edge e. To bound the total amount of energy used

for the ith flow by some quantity q, we can add a constraint of the form for each i:

$$\sum_e d(e)^\beta x(e,i) \leq q r_i \tag{8.3}$$

The constant β is the exponent that relates the energy needed to transmit over a given distance. Similarly, for bounding the total hops used in a flow by some number h, the following constraint can be added for each i:

$$\sum_e x(e,i) \leq h r_i \tag{8.4}$$

8.6 Heuristics for Path Selection

Several ad hoc routing protocols such as AODV, DSR, and DSDV use hop-count as the path metric for selecting routes: whenever a route needs to be established between a source and destination, amongst all the available routes, the protocol selects the one with the shortest number of hops. In general, hop-count based shortest paths do not optimize network throughput since several shortest paths could potentially pass through a small region in the network, resulting in "hot-spots" or regions of heavy congestion in the network. Several recent approaches have been proposed for devising path metrics to avoid hot-spots (see [38] for instance).

Motivated by the theoretical techniques developed in this work, we propose some congestion aware path selection strategies to alleviate hot-spots. The basic idea behind our path selection strategies is as follows: each link e in the network is associated with a length function $l(e)$ which is an increasing function of the link-congestion $c(e)$. Recall that the congestion $c(e)$ as defined in Section 8.3 takes into account, the load on e as well as the load on the edges which interfere with e. In practical settings, this can be estimated at the MAC layer by passively hearing the transmissions by neighboring nodes. A simple alternative would be to use the number

of end-to-end flows through a link as a substitute for the load on the link. The length of the path is the sum of all the length of its edges. Whenever we need to choose a route between two nodes s and t, we choose the route with the least length according to this metric.

Two functions suggest themselves naturally for the length metric: the linear length metric and the exponential length metric. In the linear length metric, the length of an edge $l(e) = \alpha c(e) + \beta$, where $\alpha > 0$ and $\beta \geq 0$ are protocol parameters. In the exponential length metric, the length of an edge $l(e) = e^{\epsilon c(e)}$, where $\epsilon > 0$ is a protocol parameter. In Section 8.7, we experiment with these link metrics as well as the hop-count based link metric. Our simulations indicate that both the linear and exponential link metric which takes into account congestion from interfering links, significantly outperform the hop-count based link metric in terms of network throughput.

8.7 Simulations

This section deals with the experimental performance evaluation of our algorithms and LP formulations through simulations. There are two main goals of our simulations: (i) understand the unconstrained network throughput of a random geometric network, as determined by the LP solution, and the impact of various constraints such as fairness, energy, and dilation on it, and (ii) a comparison of different path selection heuristics, which convert the LP based multi-path routing solution into a single path solution- this also helps in quantifying the single path vs multi-path tradeoff. Our simulation setup is described in Figure 8.1.

Impact of Network Constraints: We now present our simulations which deal with the impact of the network conditions on the throughput. All the experiments here were performed on the uniformly random distribution in the unit square. We

1. **Network type:** We consider the networks resulting from two types of point distributions. The first is a random distribution of 245 points in a 7×7 square. The second corresponds to a distribution of cars in a region of downtown Portland, OR, obtained by running the TRANSIMS simulation [125]. Figure 8.6(a) presents a map of the node distribution in this network. This consists of 500 points in a $3km$ by $3km$ region . We will denote it by `real-network`.
2. **Number of flows:** We experiment with varying number of end-to-end flows, each of which has a randomly chosen source and destination node. Also denoted by k.
3. **Edge Capacities:** All edges have a transmission rate of one unit of data per time unit.
4. All nodes have a unit transmission radius.
5. All data points are averaged over ten runs of the experiment.

Figure 8.1: **Summary of Simulation Setup.**

study the unconstrained network throughput as well as throughput subject to fairness, energy and dilation constraints. The objective of each experiment, the results and an analysis of these results are presented.

Experiment #1: Study the variation of the maximum throughput (sum over all rates) as a function of the number of end-to-end flows subject to wireless interference constraints, without any other additional constraints.

Results and Explanation: Figure 8.2 plots the results of our experiments averaged over ten runs and for a single run of the experiment. The maximum aggregate throughput, on an average increases steadily with the number of end-to-end pairs. However, observe that the for a single run of the experiment, the throughput exhibits a step-like behavior w.r.t. the number of end-to-end flows. This is due to the fact that the maximum network throughput is achieved at the cost of assigning a rate of zero to certain end-to-end flows. In other words, certain flows could be completely starved so that the maximum possible aggregate throughput can be achieved by the

remaining flows. Also, the total throughput flattens out, as the number of flows is increased- signifying that the absolute bound on the total capacity for the instances is reached at that point.

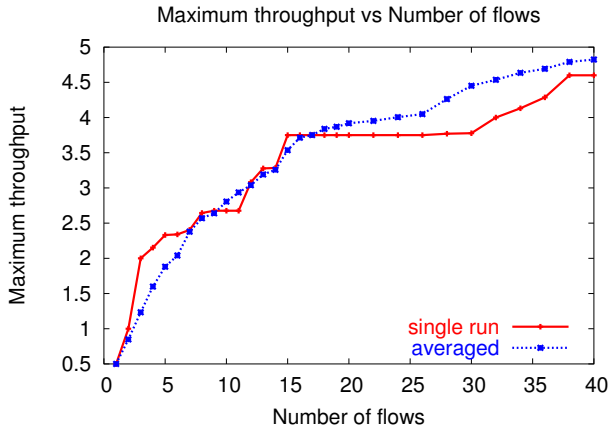


Figure 8.2: Experiment # 1: Variation of throughput w.r.t the number of flows. There is no fairness constraint.

Experiment #2: Study the variation of aggregate throughput as a function of end-to-end fairness. Recall the definition of the fairness index λ from Section 8.5: λ is the ratio between the minimum rate and the maximum rate across all flows; $\lambda = 0$ implies complete starvation of flows would be allowed while $\lambda = 1$ implies that all flows have identical throughput.

Results and Explanation: Figure 8.3 plots the maximum aggregate throughput as a function of the fairness index λ. As expected, the aggregate throughput decreases monotonically as a function of λ; the "fair" throughput is almost half of the maximum total throughput for the current choice of parameters. The results of this experiment should be contrasted with those of Experiment # 1; they provide a trade-off between system and user optimum.

Experiment #3: Study the variation of throughput w.r.t the energy consumption and dilation (number of hops traversed) per unit flow respectively.

Results and Explanation: Figures 8.4(a) and (b) summarize the results. In both

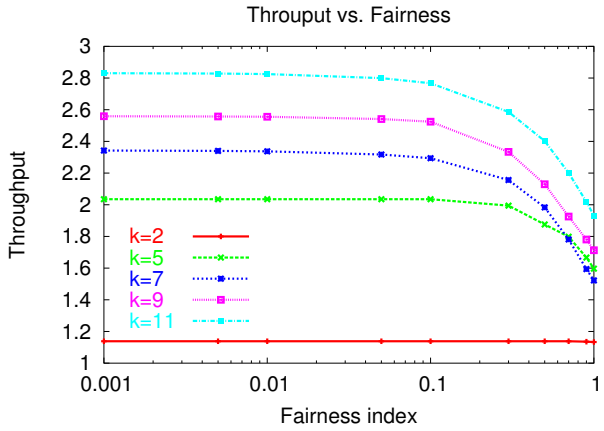
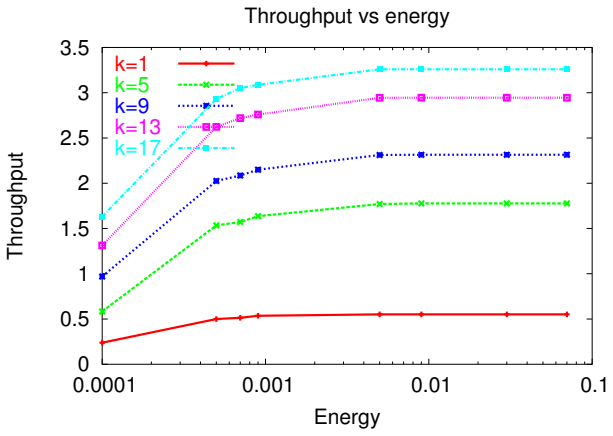
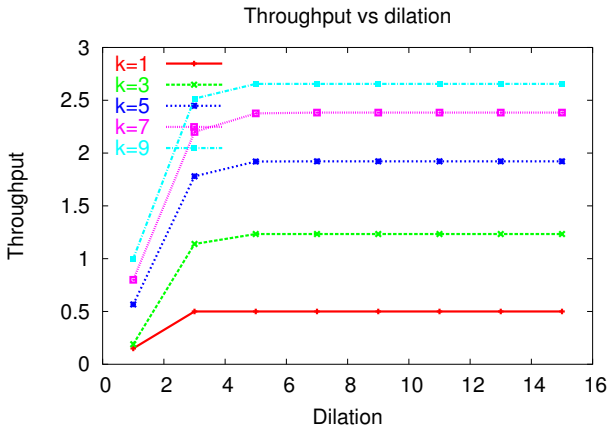


Figure 8.3: Experiment #2: Variation of throughput w.r.t fairness.

these plots, note how the throughput increases as a concave function of energy and dilation and reaches an upper limit. The similarity in the trends observed in these two plots can be explained by the fact that the constraints which model energy consumption as well as dilation are both packing constraints and are similar in structure (see Section 8.5). Also, in both of these plots, the throughput flattens out at some point after which allowing longer paths or paths with more energy does not make any difference to the throughput.



(a) Variation of throughput w.r.t energy; each curve represents the throughput for a given number of flows (denoted by k)



(b) Variation of throughput w.r.t dilation

Figure 8.4: Throughput with energy and dilation constraints

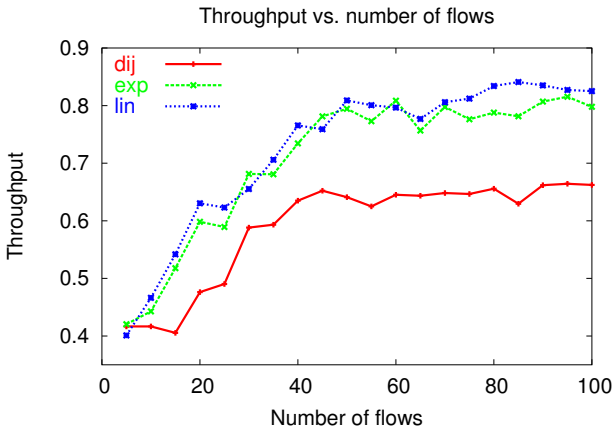
Impact of Path Selection Strategies: We now discuss the impact of various path selection strategies on the throughput. In all the measurements in this subsection, the fairness index $\lambda = 1$, i.e., we maximize aggregate throughput subject to the constraint that all end-to-end flows have equal rates.

Experiment #4: Study the impact of three different path selection strategies on the aggregate throughput. The three strategies considered here are the simple hop-count based shortest path strategy (marked dij; stands for Dijkstra), the shortest path strategy based on the linear link-metric (marked lin) and shortest path based on the exponential link-metric (marked exp).[1]

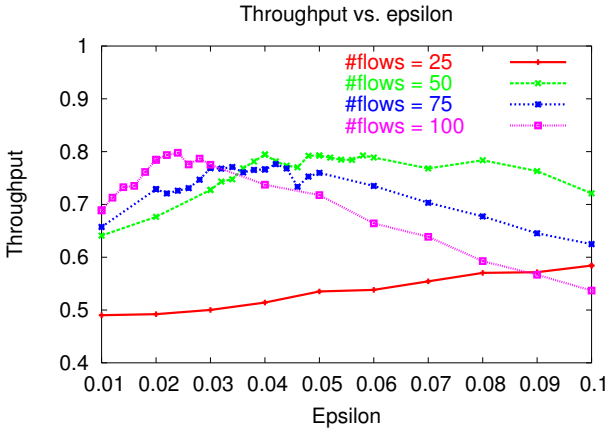
Results and Explanation: Figures 8.5(a) and (b) summarize the results. Figure 8.5(a) plots the aggregate throughput w.r.t. the number of flows for the three path selection strategies. Clearly, both the linear and exponential congestion based strategies outperform the hop-count based shortest path algorithm significantly. Further, between the linear and the exponential link metrics, the linear metric seems to be slightly better than the exponential link metric as the number of flows increase. It is interesting to compare this plot with the LP solution in Figure 8.3. Both these plots are for the same scenario, but Figure 8.3 allows multi-path routing while Figure 8.5(a) only considers single path routing. This comparison shows that the capacity drops by more than a factor of five by restricting the routes to be single paths; however, multi-path routing protocols clearly incur more overhead in terms of route maintenance, size of the routing table, etc. and these factors need to be taken into consideration for any serious comparison between single-path and multi-path routing.

Figure 8.5(b) measures the sensitivity of the throughput to the value of the exponent (denoted epsilon) in the exponential link metric. In general, note how the throughput initially increases as a function of epsilon, reaches a peak, and starts decreases as a function of epsilon. Further, when the number of flows is higher,

[1]See Section 8.6 for details of the last two strategies; for the exponential link-metric, we choose the exponent for each set of flows which maximizes the throughput



(a) Variation of throughput vs. number of flows for three path selection strategies



(b) Variation of throughput w.r.t the exponent epsilon

Figure 8.5: Throughput delivered by various routing strategies

observe that the peak value is attained for a smaller value of epsilon. Intuitively, this implies that a lower value of epsilon is more effective during times of heavy traffic than a higher value of epsilon.

Impact of Network Structure: In this subsection, we describe our simulations on realistic ad-hoc network topologies. The network we consider is described earlier and is henceforth referred to as `real-network`.

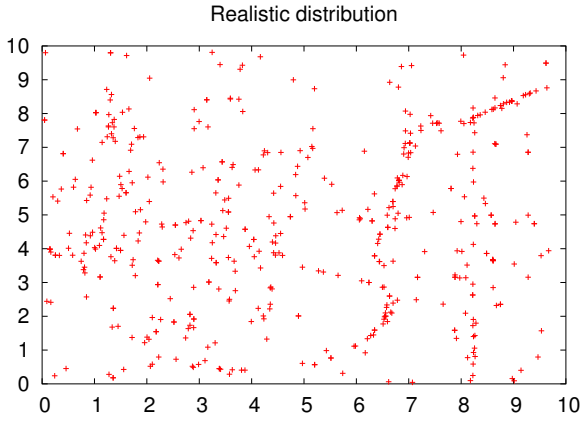
Experiment #5: Study the impact of path selection strategy on the total throughput in `real-network`. Figure 8.6(a) presents a map of the node distribution in this network. Contrast the node densities with those obtained by random distribution of nodes.

Results and Explanation: Figures 8.6(b) and (c) summarize our results. Once again, both the linear and exponential link-metric based strategies significantly outperform the shortest-path algorithm. In general, the results for this realistic network seem to be qualitatively the same as those for the random network.

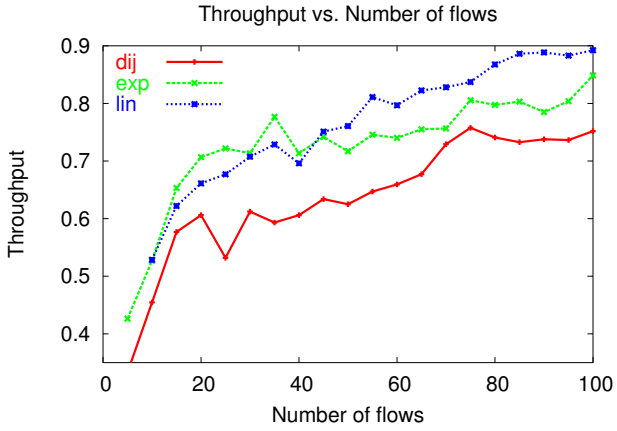
8.8 Related Work

There has been much research on determining the optimal rates to maximize throughput via LP formulations. The first attempts can be traced back to Hajek and Sasaki [54], and to Baker *et al.* [12]. Jain *et al.* [61] propose LP-formulations for max-flow and related problems in a wireless network; in fact, they formulate their constraints in terms of arbitrary conflict graphs which can incorporate any interference model. Their formulations do not fully exploit the properties of wireless interference constraints; further, they do not discuss how close their LP-formulations are with respect to the optimal solution, or how actual scheduling protocols can be derived from the LP solution.

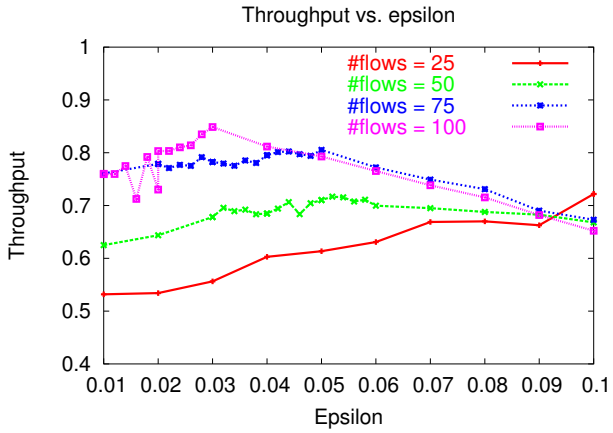
Over the last few years, the capacity of random wireless ad-hoc networks has



(a) Node Distribution in a realistic network `real-network`



(b) Throughput variation for various the three path selection strategies for `real-network`



(c) Throughput variation for the exponential metric w.r.t epsilon for `real-network`

Figure 8.6: Node distribution in a realistic network and its achievable throughput for various path-selection strategies

been a subject of active research; see [17, 75, 52, 53, 88, 109, 100, 107] and the references therein. Researchers have considered random ad-hoc networks, hybrid networks wherein one has infrastructure support, energy constraints, maximum power range constraints and mobility effects.

Our work builds upon two different results: [72] and the geometric insights of Chapter 7, and can be viewed as a synthesis of these two results. The approach of formulating the interaction between the MAC, routing and transport layers using linear programming was first considered by Kodialam and Nandagopal [72], who propose similar LP-formulations and a scheduling algorithm for determining the maximum transmission rates for a given network, and for three specific interference models (PCA, RCA, TRCA). They also show how to use the approach in [46] for solving the LPs using a sequence of shortest-path computations. They do not show how to use the LP solution to get an actual schedule with provable performance guarantees. Solving LPs is very time consuming, and an LP based method is generally impractical, especially for mobile computing applications. To remedy this, Kodialam and Nandagopal show how to use the framework of Garg and Konemann [46] to device a combinatorial method of solving such an LP, within any desired level of accuracy. In Chapter 7 of this thesis, we formulate the problem of minimizing latency for the MAC and routing layers together and develops efficient distributed algorithms for this problem in geometric graphs, which are a commonly used abstraction of radio transmission. However, this work only deals with a static setting, i.e., when packets are not injected continuously into the network, and obtaining provable results for the unified protocol design problem with continuous packet injections has been a very interesting open problem. De Couto *et al.* [37] and Draves *et al.* [38] present link-metrics for shortest-path based routing algorithms which optimizes the total expected number of transmissions of a packet and the expected time to transmit a packet respectively. Our link-metrics differ from the one presented in [37] by accounting for interference

between links which may belong to different routes and by adapting to load changes on a link and the set of links which interfere with it.

In practice, the problem of transmitting packets between each source-destination pair in a OSI protocol stack based model is broken down into sub-problems, the most important of which are: (i) choosing routes for each such pair - a protocol like AODV chooses some sort of (single) shortest path for each pair, (ii) MAC scheduling of the packets along these paths - this resolves contention, and determines which nodes transmit at a given time slot, (iii) actual transmission of the packets on the physical channel, and (iv) choosing rates of transmission for each source-destination pair - this is achieved dynamically by a TCP like protocol, which uses feedback from the network to regulate the flow. While this modularity is useful in designing the network, it is almost impossible to determine the quality of the performance of such protocols, and how to improve the performance. In fact, there is a significant interaction between protocols at different layers [20], and plugging in optimal protocols for each layer does not lead to optimal overall performance [13, 20, 113, 124]. This has motivated the study of unified protocols, and unified measures that capture the overall performance. The work of Anil Kumar et al. [76] presents unified algorithms for routing+scheduling with provable guarantees for wireless networks under static packet injections.

Chapter 9

On the Capacity of Random Access Wireless Networks

9.1 Introduction

Chapter 8 initiated the study of capacity estimation and throughput maximization in arbitrary wireless networks subject to interference constraints. A key ingredient of our results in Chapter 8 was the *deterministic* inductive scheduling protocol which schedules links without interference, and yields provably good end-to-end throughputs. Wireless devices in practice, however, almost always use random access contention resolution protocols for arbitrating channel access. Nodes in such networks do not transmit on precomputed times in order to avoid interference related conflicts. Instead, nodes access the wireless medium probabilistically, with the twin goals of utilizing the network bandwidth effectively, and minimizing the chances of interference related losses. Random-access networks however continue to remain a "black box" from an analytical perspective. In particular, the fundamental question of end-to-end capacity estimation and throughput maximization in such networks remains to be well understood. This is in stark contrast to *wired* networks or deterministically sched-

uled wireless networks, where the use of linear / convex programming techniques has yielded a rich analytical framework for routing, throughput maximization, congestion control, fair bandwidth allocation, and related network optimization problems. As a result, the vast majority of practical wireless protocols are heuristic in nature, without rigorous analytical justification for *why* they do or do not work, and how then can be improved.

In this work, motivated by the large scale deployment of 802.11 and other random-access protocols, we initiate an analytical framework for end-to-end throughput estimation and maximization in random-access wireless networks. We begin by developing a non-linear programming (NLP) formulation which characterizes the achievable end-to-end throughputs in these networks. While this NLP precisely captures the impact of interference and the specific constraints imposed by a given scheduling protocol, solving this NLP is computationally infeasible. Our first contribution is to demonstrate how *the NLP can be approximated by a provably good linear program (LP) for a wide-class of random access protocols.* This paves the way for LP based optimization techniques in random-access wireless networks. Specifically, we show how a network planner can use our LP as a capacity planning tool for optimizing end-to-end throughputs, throughput based utilities, end-to-end fairness, network-wide energy consumption, and route selection in random-access wireless networks.

As the second broad contribution in this work, we show how our analytical insights can be applied in the design of routing metrics in distributed wireless routing protocols. Designing good routing metrics which accurately captures link quality is critical to the performance of routing protocols in wireless networks. In this work, we use our LP and NLP formulations to analytically interpret and quantify the differences between existing routing metrics such as hop-count, ETX [37], and ETT [38]. Further, we develop the Available Capacity Metric (ACM), which builds upon ETX and ETT by carefully incorporating the effect of interfering traffic on link quality, without

incurring any additional measurement overhead. We demonstrate using NS-2 simulations that ACM is better than ETT in capturing link-level packet transmission times, and correlates better with end-to-end throughput than ETX, ETT, and hop-count.

The main difficulty in developing linear programming models for random-access wireless networks is that *the interaction between the link flows across interfering links is governed by a complex stochastic process*, which makes their characterization very difficult in a linear program. To make the above statement more precise, let us define the link-flow vector $\mathbf{f}$ with $|E|$ components, one for each link e in the network, where component $\mathbf{f}(e)$ represent the total data flow across link e. $\mathbf{f}$ is said to be a *feasible* link-flow vector if and only if the MAC protocol can simultaneously support a flow of $\mathbf{f}(e)$ for each link e in steady state. *The crux of the capacity estimation and throughput optimization problem is to efficiently express the set of all feasible link-flow vectors of a wireless network with a given random-access scheduling protocol.* In this work, we first formulate a non-linear program (NLP) which captures the feasible link-flow region in a random-access wireless network, and then show how to approximate this NLP using a provably good near-optimal linear program (LP).

9.2 Notation and Assumptions

In this section, we formalize our assumptions and notation, which we summarize in Table 9.1. The models introduced here have much in common with those used in Chapters 7 and 8.

Communication Network: We consider a multi-radio multi-channel multi-hop wireless network with non-uniform link rates, and model it as a directed graph $G = (V, E)$. A vertex in the graph represents an individual radio or wireless interface within a wireless node. A link $(u, v) \in E$ denotes that vertex u can transmit

directly to vertex v. For a transmission to be successful, both the data frame and the acknowledgment must be successfully sent (in opposite directions). Hence, we include only bidirectional links in G (*i.e.*, $(u, v) \in E$ implies that $(v, u) \in E$ as well), though the two directions may be asymmetric and have very different properties.

Each directed edge $e = (u, v) \in E$ has capacity $c(e)$, the maximum rate at which node u can transmit data to node v. Each vertex operates on a fixed channel from a set of *orthogonal* channels. Henceforth, we assume that all nodes send on the same channel, but as we discuss later, this can be easily generalized. For a wireless link (u, v) to be present, two conditions must hold: (i) both u and v operate on the same channel, and (ii) the distance between u and v is at most the maximum possible transmission range in the network. These conditions are necessary but not sufficient; for example, if u cannot transmit to v because of the presence of an obstruction between them, then the link $(u, v) \notin E$. Allowing for such arbitrary edges (as opposed to, say, assuming a purely distance-based model) is important when modeling realistic scenarios such as indoor wireless networks or even outdoor networks in the presence of obstructions and multi-path effects.

Random Access MAC Protocols: We start with a general model for exponential backoff-based random access contention resolution protocols. A node which needs to transmit data, initializes its *backoff timer* to a value chosen uniformly at random in the range $[0, W]$. The backoff timer is decremented by one for every contiguous time period of length T_{id}, during which the channel is sensed idle. The timer is paused when the medium is sensed busy, and reactivated when the medium is again sensed idle. When the backoff timer expires, the node performs one last check that the channel is idle and transmits if it is. Upon receiving the data frame successfully, the receiver sends an acknowledgment (ACK) without using the above back off procedure. If the ACK is not received by the sending node after a fixed period of time (either because

the data packet is lost or the ACK itself is lost), or if the channel was not idle during the sender's final check, then the sending node schedules a retransmission. The value of W is set to W_0 in the first attempt, and is set to W_i for the i^{th} retransmission: in 802.11, $W_i = 2^{i+5}$. A maximum of m retransmission attempts are allowed before the packet is dropped.

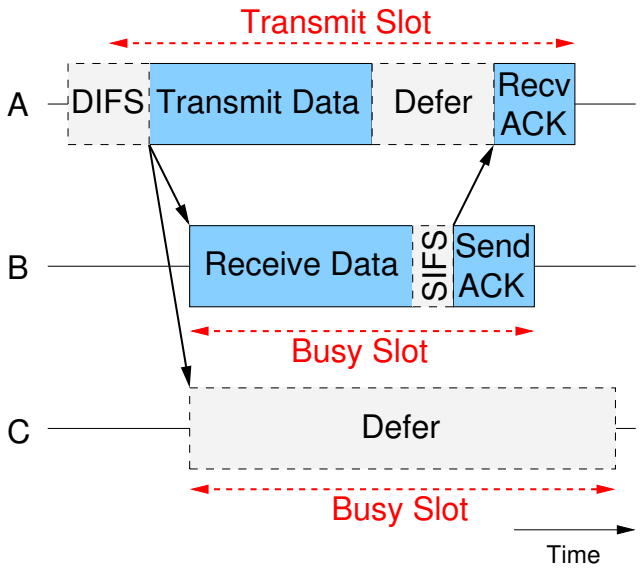


Figure 9.1: Timing diagram for the basic access method of the 802.11 MAC protocol.

The access mechanism is depicted in Figure 9.1. Each node u perceives time as slotted, where each time slot is one of the following three types. An *idle slot* occurs when neither u nor the nodes within its carrier-sense range are currently transmitting. A *transmit slot* occurs when u transmits data to one of its neighbors. A *busy slot* occurs when u is not currently transmitting but another node v within the *carrier-sense range* of node u is transmitting. In this case, u senses a busy medium and considers the entire block of time for which the medium is busy to be a busy slot. An important distinction between our use of time slots and that of systems such as IdleSense [57] is that we do **not** assume the network to be a clique, hence different nodes can have *very different pictures* of which slots are idle, transmissions, or busy.

The length of each idle slot, T_{id}, is a constant defined by the MAC protocol as

the maximum delay incurred by a node in detecting a busy medium after some node in its carrier-sense neighborhood has initiated transmission. When node u sends to its neighbor v, the length of the transmit slot, $T_{xmit}(u, v)$, is the total time taken by u to transmit its data frame and receive a link-level acknowledgment from v (upon successful transmission). For ease of exposition, we assume that all data packets have a fixed size of M bits, and the time needed for receiving the ACK is negligible compared to the time needed to transmit the data frame. In other words, we have $T_{xmit}(u, v) \approx \frac{M}{c(u,v)}$. The length of a busy slot as measured at u depends on which nodes in the carrier-sense neighborhood of u are occupying the channel. A precise formulation of this does not admit a simple expression. For ease of exposition, we assume that the lengths of transmit and busy slots are integral multiples of T_{id}, and the first slot begins at the same time for all nodes in the network. We stress that the integrality and synchronization assumptions simplify the analysis and are not necessary for our results to hold. The slot types and lengths are illustrated in Figure 9.1.

Packet Loss Model: Transmission failures may occur either due to channel errors or due to interference from neighboring links. We make certain to distinguish between these two causes so that we can account for losses that are independent of other nodes' traffic (channel errors) as well as losses that are a direct result of others' traffic (interference). We assume that packet losses due to channel errors occur with a fixed probability, $(1 - \rho(e))$, for each link e (*i.e.*, $\rho(e)$ represents the channel success rate). A transmission may also fail when a nearby link attempts to transmit at the same time, resulting in a collision. Let $I(e)$ denote the set of all links which interfere with a given link $e = (u, v)$, that is, if $(w, x) \in I(e)$ then at least one of w or x interferes with at least one of u or v. If the transmit slots of a node in link e overlap in time with the transmit slots of a node in link $e' \in I(e)$, then e's transmission experiences

Notation	Meaning
$c(e)$	The capacity of edge e.
$f(e)$	Total flow rate sent on edge e.
$I(u)$	The set of nodes whose transmission interferes with node u.
$I(e)$	$I(u) \cup I(v)$: the interference set of edge $e = (u, v)$
$r(i)$	The data rate at (either generated by or forward by) node u.
$\rho(e)$	Channel success rate for edge e.
T_{id}	The length of an idle slot (constant defined by 802.11).
$T_{xmit}(u)$	The mean length of transmission slot as measured by node u.
$T_{coll}(u)$	The mean length of a collision slot as measured by node u.
$\overline{T(u)}$	The mean slot length as measured by node u.
$\tau(e)$, $\tau(u)$	$\tau(e)$ = channel-access probability for edge e, $\tau(u) = \sum_{(u,v)\in E} \tau((u, v))$.
$\eta(e)$	Probability that a given channel access on edge e is successful.
$p(u)$	Probability of transmission failure for node u.
$\mathcal{P}_i$	The set of paths used by source-destination pair (s_i, t_i).
M	Packet size.
W_i	Window size for i^{th} retransmission.
m	The maximum number of retransmissions after which the MAC protocol drops the packet.

Table 9.1: Notation used in this chapter.

a collision and fails.[1]

9.3 A Non-Linear Model

Working under the above assumptions, we now present a non-linear programming model which captures the available capacity of an arbitrary, multi-hop 802.11 network.

The Decoupling Approximation: It is possible to model the joint backoff processes of all the nodes in the network as a Markov chain, for the very special case of a single-hop network (all nodes in the network are within each other's communication range and interference range) in which nodes are saturated (every node always has a packet to transmit). Even in this special case, the Markov chain model is analytically

[1] In the case of multiple channels, since channels are orthogonal, link e' can interfere with e only if e' operates on the same channel as e (this is a necessary but not sufficient condition).

intractable from the perspective of algorithm design and optimization. Instead, a standard approach is to invoke the *decoupling approximation* which was introduced by Bianchi [24] and works as follows. Consider a link $e = (u, v)$ and its interfering set of links $I(e)$. The decoupling approximation states that the aggregate attempt process of all the links in the set $I(e)$ is independent of the backoff process at node u. More concretely, let $\tau(u)$ and $\tau(e)$ denote the *steady state* channel-access probability for node u and link e; *i.e.*, $\tau(u)$ denotes the probability of node u attempting to transmit during a slot in steady state; $\tau(e)$ denotes the probability of node u attempting to transmit across link e during a slot in steady state. The decoupling approximation states that the steady-state probabilities $\tau(u)$ and $\tau(e)$ exist (i.e., a stationary distribution exists) and are independent of the steady-state access probability $\tau(e')$ for any other link $e' \in I(e)$. It is reasonable to expect, as is verified in our simulations in Section 9.4, that the decoupling approximation works well in *low-rate regimes* where each link's access probability is small in comparison with the total access probability of its interfering links.

An NLP for Maximizing Throughput in Multi-hop 802.11 Networks: Here we provide an NLP that captures the feasible link-flow region of an 802.11 network. We start with modeling flows in a wired network (without interference or channel-error related losses). This is the well-known *multi-commodity network flow problem*, and can be formulated easily as in Figure 9.2.

Let us review the constraints in Figure 9.2. The objective function (9.1) states that the total end-to-end rate of all connections, $\sum_i r(i)$ should be maximized. Let $\mathcal{P}_i$ denote the set of all paths in the network from s_i to t_i; for any path $p \in \mathcal{P}_i$, let $r(p)$ denote the flow through p (*i.e.*, the steady-state rate at which data is routed through p). The total end-to-end flow for connection i, $r(i)$, is the sum of the flows across all paths $p \in \mathcal{P}_i$, which is represented by constraint (9.2). Similarly, the total flow

$$\text{Maximize} \sum_i r(i) \qquad \text{subject to} \tag{9.1}$$

$$\forall i \in \{1, \ldots, k\}, r(i) = \sum_{p \in \mathcal{P}_i} r(p) \tag{9.2}$$

$$\forall e, f(e) = \sum_{e \in p} r(p) \tag{9.3}$$

$$\forall e, f(e) \leq c(e) \tag{9.4}$$

Figure 9.2: The multi-commodity flow problem in wired networks can be formulated as a linear program and solved in polynomial time.

across a link e, $f(e)$, is the sum of end-to-end rates that use that link, as captured by constraint (9.3). Lastly, constraint (9.4) guarantees that any link e cannot send more than its capacity. These straightforward constraints constitute a simple linear program which can be solved *optimally* in polynomial time.[2].

Our approach now is to add constraints to the LP in Figure 9.2 to address (i) the loss incurred by interfering links, and (ii) the inherent limitations of the underlying MAC-level protocol.

Modeling Interference: Let the expected length of a time slot for node u in steady-state[3] be $\overline{T(u)}$. If the link-flow vector is $\mathbf{f}$, it follows that the expected number of bits generated *per time slot* for link e is equal to $f(e)\overline{T(u)}$. Recall that $(1 - \rho(e))$ is the probability of a channel error occurring during a transmission across link e. Let $(1-\eta(e))$ denote the probability of a collision occurring for link e (due to interference) during a transmission. The expected number of bits successfully transmitted over link e per time slot is M (the nominal data packet size) times the probability of a successful

[2]The formulation in Figure 9.2 is exponential in size since there is a rate variable $r(p)$ for each $s_i - t_i$ path p, and there could be exponentially many paths. We use this formulation for ease of exposition; the equivalent standard LP formulation using flow-conservation constraints has only polynomial size and time complexity[1].

[3]All probabilities and expectations in this section represent steady-state values.

transmission on e during that time slot; i.e.,

$$\begin{aligned} & \Pr[\text{Transmission attempted}] \cdot \Pr[\text{Success}] \cdot M \\ = \;& \tau(e) \cdot \rho(e)\eta(e) \cdot M \end{aligned}$$

This product incorporates $\tau(u)$ (the probability of transmission occurring across link e during a slot), $\rho(e)\eta(e)$ (the probability of the transmission succeeding), and M, the nominal data packet size.

We are now ready to add a constraint to our NLP. Clearly, the expected number of bits generated at link e cannot exceed the expected number of bits which can be successfully transmitted on e. Otherwise, e would not be able to handle its demand, $f(e)$, and hence the link-flow vector $\mathbf{f}$ would be infeasible. Formally, we capture this *stability condition* as:

$$\forall e = (u, v) \in E : \quad f(e) \cdot \overline{T(u)} \;=\; \tau(e) \cdot \rho(e)\eta(e) \cdot M \tag{9.5}$$

Modeling a Specific MAC-Level Protocol: Equation (9.5) guarantees that for a suitable choice of channel-access probabilities, $\tau(e)$ for each link e, the link-flow vector $\mathbf{f}$ can be scheduled by the MAC protocol. However, the MAC protocol itself imposes certain limits on the steady-state channel access probabilities, thereby further constraining the set of feasible link-flows. Clearly, not all MAC-level protocols are created equal, so a given NLP must model one in particular. This means we must add constraints for $\tau(e)$ that represent the capabilities of that protocol. In this work, we consider 802.11.

Let $out(u)$ denote the set of out-going links from u. Clearly, we have $\tau(u) = \sum_{e \in out(u)} \tau(e)$. Let $p(u)$ denote the steady-state probability of transmission failure for node u. By analyzing the backoff process at a node, Bianchi [24] showed that $\tau(u)$ is upper-bounded by $\frac{1+p(u)+\cdots+p(u)^m}{W_0+p(u)W_1+\cdots+p(u)^m W_m}$. When node u is saturated, then $\tau(u)$

equals its upper bound; otherwise, it is strictly less than its upper bound. Hence, the saturation constraints imposed by the 802.11 MAC protocol can be captured through equations (9.6) and (9.7):

$$\forall u \in V, \tau(u) = \sum_{out(u)} \tau(e) \tag{9.6}$$

$$\forall u \in V, \tau(u) \leq \frac{1 + p(u) + \cdots + p(u)^m}{W_0 + p(u)W_1 + \cdots + p(u)^m W_m} \tag{9.7}$$

Equations (9.5), (9.6), and (9.7), when added to the LP in Figure 9.2, results in our NLP for end-to-end throughput maximization in wireless networks.

Discussion: An exact solution to this NLP would reveal the individual connection throughputs $r(i)$ as well as the distribution of per-connection flows across the flow paths, which maximizes the aggregate throughput $\sum_i r(i)$. However, this NLP is only a *partial* specification. Observe that the variables $\overline{T(u)}$, $\eta(e)$, and $p(u)$ are all functions of the $\rho(e), \tau(e)$, and $f(e)$ values. However, no closed form expressions are known for the former variables in terms of the latter ones. A complete specification of the NLP requires us to derive such closed-form expressions, and add them as constraints to the NLP. Even if we could derive constraints, the resulting NLP is unlikely to be a convex program and hence, unlikely to admit polynomial-time algorithms for computing the optimal solution. This is in contrast to wired networks, where an LP completely characterizes the feasible link-flow region, and can be solved in polynomial-time to optimize the aggregate connection throughput.

9.4 An Approximate Linear Model

The model in Section 9.3 accurately represents a multi-hop 802.11 wireless network, but since it is non-linear, we are unable to solve problems such as the multi-commodity

flow problem in polynomial time. A linear programming model would allow us to employ the many techniques that have benefited wired networks, such as the computation of routes that maximize system-wide throughput. In the absence of exact algorithms which optimize the end-to-end throughput, we are forced to investigate approximate techniques. In this section, we shift our focus towards such an approximate, but *provably good*, linear programming-based algorithm. We show that, when the network is unsaturated, our LP is a very close approximation in practice. First, we introduce the concept of low-congestion regime which is fundamental to our modeling framework.

The Low-Congestion Regime: Consider a link-flow vector $\mathbf{f}$. Given a link $e = (u, v)$, w.r.t. the link-flow vector $\mathbf{f}$, we define the *channel occupancy* of e as

$$x(e) \stackrel{\text{def}}{=} \frac{f(e)}{c(e)}$$

In other words, $x(e)$ is the fraction of the time e is utilizing the channel successfully to support the vector $\mathbf{f}$. Define the *congestion* of a link as

$$\ell(e) \stackrel{\text{def}}{=} x(e) + \sum_{e' \in I(e)} x(e')$$

Note that congestion for e includes its occupancy as well as that of all other links which interfere with e. We define $\ell(u)$, the congestion for node u similarly as $\sum_{e \in I(u)} x(u)$.

We say that the network operates under the *low-congestion regime* if the following condition holds:

$$\forall e \in E: \qquad \ell(e) = x(e) + \sum_{e' \in I(e)} x(e') \ \leq \ \epsilon \tag{9.8}$$

where, ϵ is a small positive constant (say $\frac{1}{3}$). The low-congestion regime requires that for every link e in the network, the total fraction of the time during which e or any of

its interfering links in $I(e)$ are involved in a successful packet transmission, is upper bounded by a small constant ϵ. We note that condition (9.8) automatically implies that for each node u, $\ell(u)$ is upper-bounded by ϵ as well.

Analysis: As discussed above, solving the NLP exactly seems intractable. We now state and formally prove our main result in Theorem 78, that under certain assumption about the rates, we can get linear approximations for $\tau(e)$ and $\eta(e)$. We will verify these approximations empirically later in this section, and then use them to derive a linear approximation to the NLP. For ease of analysis and exposition, we assume absence of hidden terminals for each link in the network in Theorem 78 and discuss how this assumption can be removed in later. We also assume that all packets are M bytes in size.

Theorem 78 *Let $\beta \doteq \min_e \frac{T_{xmit}(e)}{T_{id}}$ denote the ratio between the minimum length of a transmission slot for any link and the length of an idle slot. Suppose $\forall e \in E, \ell(e) = x(e) + \sum_{e' \in I(e)} x(e') \leq \epsilon$, where ϵ is a suitably chosen constant dependent only upon β (in other words, the network operates in the low-congestion regime). Then, there exists a value $\tau(e)$ for each link e such that*

(P1) $\tau(e) \leq \frac{2f(e)}{\beta(1-\epsilon)c(e)}$*;*

(P2) $\eta(e) \geq 1 - \frac{2\epsilon}{\beta(1-\epsilon)}$*; and*

(P3) *the non-linear stability condition (9.5) holds.*

Proof We first show that **(P2)** holds *whenever* **(P1)** holds. Fix a link e and link e' which interferes with e. Since there are no hidden terminals, it follows from the

decoupling assumption that:

$$\begin{aligned} \forall e,\ \eta(e) &= \Pi_{e' \in I(e)}(1 - \tau(e')) && (9.9) \\ &\geq 1 - \sum_{e' \in I(e)} \tau(e') && (9.10) \\ &\geq 1 - \sum_{e' \in I(e)} \frac{2f(e)}{\beta(1-\epsilon)c(e)} && (9.11) \\ &\geq 1 - \frac{2\epsilon}{\beta(1-\epsilon)} && (9.12) \end{aligned}$$

Eqn. (9.11) holds since we assumed **(P1)** holds; (9.12) follows from the low-congestion condition. So, **(P2)** holds.

We now show that if we set $\tau(e) = \frac{2f(e)}{\beta(1-\epsilon)c(e)}$ for all e, then the L.H.S. of (9.5) is upper-bounded by the R.H.S.: i.e.,

$$\forall e = (u, v) \in E: \quad f(e) \cdot \overline{T(u)} \ \leq \ \tau(e) \cdot \rho(e)\eta(e) \cdot M \tag{9.13}$$

It thus follows that a suitable choice of τ's exist which satisfies all of **(P1)**, **(P2)**, and **(P3)**.

Fix a link $e = (u, v)$. Let $\gamma(u)$ be the expected fraction of the time during which the channel is perceived as idle by node u. Consider any time window of unit length. The expected number of idle slots for node u within this window is $\frac{1-\gamma(u)}{T_{id}}$. Since $\overline{T(u)}$ is the expected length of a time for node u, we have

$$\overline{T(u)} \leq \frac{T_{id}}{1 - \gamma(u)} \tag{9.14}$$

Consider the interference set $I(e)$. If u does *not* perceive the channel as busy, then either e or some link(s) $e' \in I(e)$ is (are) involved in transmission. The expected fraction of the time during which link e transmits is equal to $\frac{x(e)}{\eta(e)}$: this follows from the fact that $x(e)$ is the expected duration for which e occupies the channel successfully,

and $\frac{1}{\eta(e)}$ is the expected number of times e needs to transmit a packet before its successful reception. Define $\delta \stackrel{\text{def}}{=} \min_e \eta(e)$. We now have

$$\gamma(u) \leq \frac{x(e)}{\eta(e)} + \sum_{e' \in I(e)} \frac{x(e')}{\eta(e')} \leq \frac{1}{\delta} \cdot (x(e) + \sum_{e' \in I(e)} x(e')) \leq \frac{\epsilon}{\delta}.$$

Thus, to show that (9.13) holds, it is enough to show that

$$\begin{aligned}
f(e) \cdot \overline{T(u)} &\leq \tau(e) \cdot \eta(e) \cdot M \\
\text{i.e., } \frac{f(e)T_{id}}{1-\gamma(u)} &\leq \tau(e) \cdot \eta(e) \cdot M \\
\text{i.e., } \frac{f(e)T_{id}}{1-\frac{\epsilon}{\delta}} &\leq \frac{2f(e)}{\beta(1-\epsilon)c(e)} \cdot \delta \cdot c(e)T_{xmit}(e) \\
\text{i.e., } \frac{f(e)T_{id}}{1-\frac{\epsilon}{\delta}} &\leq \frac{2f(e)}{(1-\epsilon)c(e)} \cdot \delta \cdot c(e)T_{id} \\
\text{i.e., } \frac{1}{1-\frac{\epsilon}{\delta}} &\leq \frac{2\delta}{(1-\epsilon)} \\
\text{i.e., } 1 &\leq \frac{2 \cdot (\delta - \epsilon)}{(1-\epsilon)} \qquad (9.15)
\end{aligned}$$

Finally, it follows from **(P2)** that $\delta \geq 1 - \frac{2\epsilon}{\beta(1-\epsilon)}$; it is easy to verify that (9.15) holds for any ϵ such that $\frac{4\epsilon}{(1-\epsilon)^2} \leq \beta$. Hence, Theorem 78 holds. ■

Experimental Investigation: We now investigate the behavior of the network under the low-congestion regime, using NS-2 simulations of an 802.11 network. We will empirically verify the consequences (P1) and (P2) of Theorem (78). The network consists of 196 nodes arranged in a 14×14 grid topology as shown in Figure 9.3. The nodes are located in the lattice points of the grid, and all the nodes operate on a single channel with a peak data-rate of $54Mbps$. The transmission range and carrier-sense range are such that nodes belonging to adjacent lattice points are neighbors of each other and nodes that are two hops away are within the carrier-sense range of each other. There are four connections: $\{s_1 \rightarrow t_1, s_2 \rightarrow t_2, s_3 \rightarrow t_3, s_4 \rightarrow t_4\}$, and packets from each source is routed to the corresponding destination by the shortest

straight-line path between them. All sources inject UDP traffic into the network at the same rate, with the total injected load steadily increasing with time as shown in Figure 9.4. All packets are of fixed size $M = 512$ Bytes. In this simulation, the channel is error-free (i.e., all ρ values are 1) and all packet losses are due to interference. Our main objective is to characterize the steady-state network behavior, in particular, the average slot duration for a node ($\overline{T(u)}$), the channel access probability of a link ($\tau(e)$), and the probability of successful transmission for a link ($\eta(e)$) in the low-congestion regime.

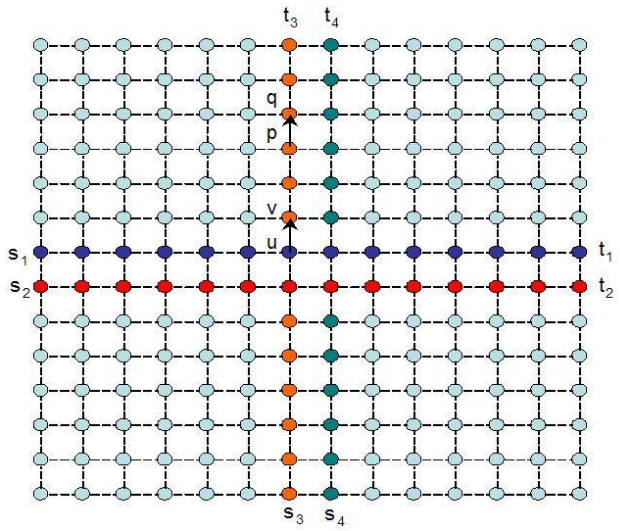


Figure 9.3: The 14×14 grid used in the simulation. Nodes in the adjacent lattice points are one-hop neighbors, and nodes within two hops are in each other's carrier sense range. Source s_i sends data to destination t_i through the straight line path. The measurements are for node u and link (u, v). Observe that node p is in the carrier-sense range of v but not u and hence is a hidden terminal for link (u, v)

Main Insights: We now present the main results from the simulation.

$\overline{T(u)}$ is close to T_{id} Figure 9.5 plots the variation of the average slot length for node u w.r.t. time. In the low-congestion regime (time $\leq$ 1000 sec), the traffic carried by each link is not very high, and hence node u perceives the channel to be idle for a large fraction of the time. In any large window of time, the total expected number of busy

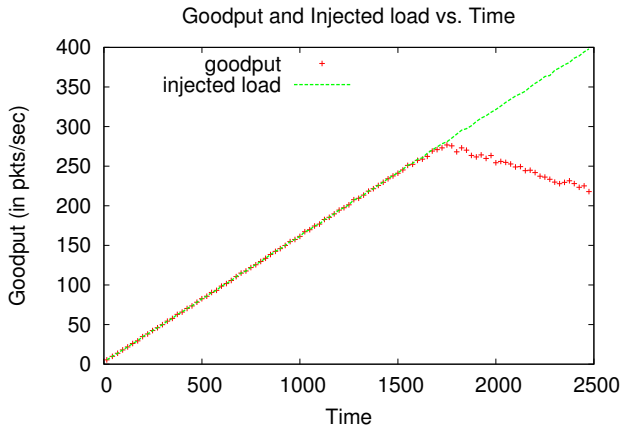


Figure 9.4: Total injected load and network-wide throughput over time. The network saturates around 1750 sec

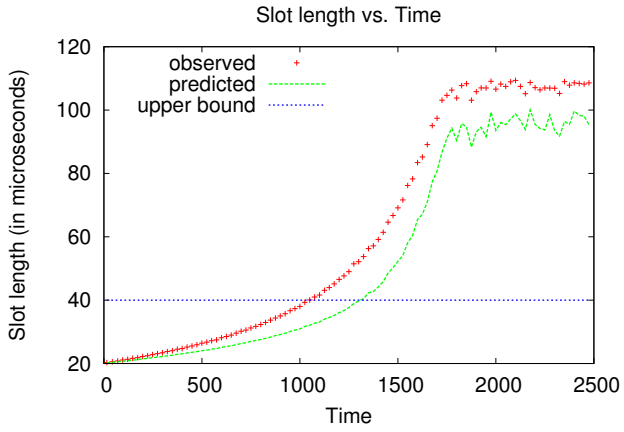


Figure 9.5: The variation of average slot length $\overline{T(u)}$ over time. Note how the observed and predicted values never exceed twice the length of an idle slot within the low-congestion regime.

and transmit slots for u is negligible compared to the expected number of idle slots. Hence, $\overline{T(u)}$, u's mean slot length, can be approximated as $\frac{T_{id}}{1-\ell(u)}$. Further, since $\ell(u)$ is upper bounded by a small positive constant ϵ, it follows that $\overline{T(u)} \approx \frac{T_{id}}{1-\ell(u)} \leq \frac{T_{id}}{1-\epsilon}$. Notice how well the predicted value $\frac{T_{id}}{1-\ell(u)}$ matches the observed value of $\overline{T(u)}$ in Figure 9.5; the gap between the observed and predicted values can be reduced further by incorporating 802.11 model-specific parameters (such as DIFS and SIFS) into our prediction. Crucially, in the low-congestion regime, we observe that $\overline{T(u)}$ never exceeds $2T_{id}$.

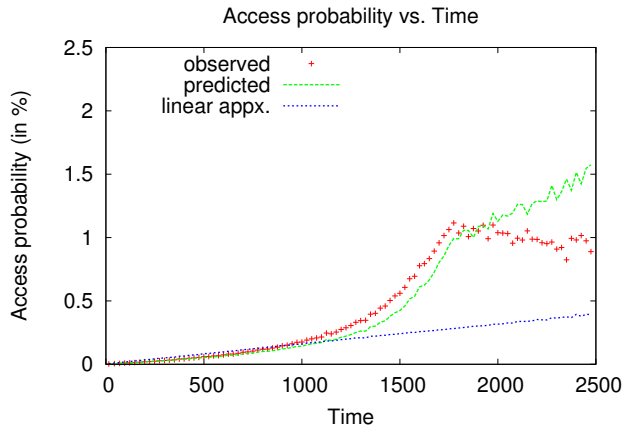


Figure 9.6: The variation of the access probability $\tau(e)$ over time. Both the predicted value and the linear approximation match the observed value very well in the low-congestion regime.

$\tau(e)$ can be approximated by a linear function of $f(e)$ Figure 9.6 plots the channel access probability τ for the link $e = (u, v)$. In the low-congestion regime, the total channel occupancy of the links that interfere with e is at most ϵ; hence, $\eta(e)$, the success probability is very high and can be approximated as 1.0. From Eqn. (9.5)

it follows that

$$\begin{aligned} \tau(e) &= \frac{f(e)\overline{T(u)}}{\rho(e)\eta(e)M} \approx \frac{f(e)\overline{T(u)}}{M} \\ &\approx \frac{T_{id}f(e)}{(1-\ell(u))M} \\ &\leq \frac{2T_{id}f(e)}{M} \end{aligned} \tag{9.16}$$

Notice that $\frac{T_{id}f(e)}{(1-\ell(u))M}$ yields a non-linear approximation for $\tau(e)$ as a function of the network load, while $\frac{2T_{id}f(e)}{M}$ yields a linear approximation for $\tau(e)$ which is directly proportional to the link flow $f(e)$. As Figure 9.6 indicates, both the non-linear and the simplified approximations for $\tau(e)$ work very well in the low-congestion regime.

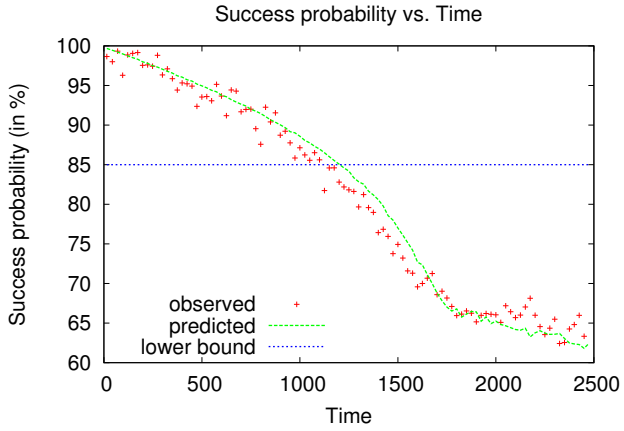


Figure 9.7: The variation of $\eta(e)$ over time. The predicted value is in perfect agreement with observed value, and $\eta(e)$ is greater than 85% in the low-congestion regime.

$\eta(e)$ is close to 1 Figure 9.7 plots the success probability η for links $e = (u, v)$ and $g = (w, s_4)$ w.r.t. time. Notice that node p in Figure 9.3 is within the carrier-sense range of v but not within the range of u; hence p is a hidden terminal for link e. However, link g has no hidden terminals, since every node within the carrier-sense range of s_4 is also within that of w. In the absence of hidden terminals, we can use the expression for τ derived earlier in eqn. (9.16) to approximate η. Since link g has no hidden terminals, a link $g' \in I(g)$ can collide with g only if g' and

g initiate transmissions during the same time slot. By the decoupling assumption, the probability of link g' initiating transmission during the same slot at which g has initiated a transmission is $\tau(g')$. Hence,

$$\eta(g) \geq 1 - \sum_{g' \in I(g)} \tau(g') \geq 1 - \sum_{g' \in I(g)} \frac{2T_{id} f(e')}{M}.$$

This does not hold in the presence of hidden terminals: e.g., the probability of a transmission on link $e' = (p, q)$ colliding with a transmission on $e = (u, v)$ is not $\tau(e')$ but can be approximated as $x(e')$ - the channel occupancy of link e'. Let $I_1(e) \subseteq I(e)$ consist of links whose source nodes are within the carrier-sense range of the source of e, and let $I_2(e) = I(e) \setminus I_1(e)$. Hence,

$$\begin{aligned} \eta(e) &\geq 1 - \sum_{e' \in I_1(e)} \tau(e') - \sum_{e' \in I_2(e)} x(e') \\ &\geq 1 - \sum_{e' \in I_1(e)} \frac{2T_{id} f(e')}{M} - \sum_{e' \in I_2(e)} \frac{f(e')}{c(e')} \end{aligned}$$

As shown in Figure 9.7, we can approximate η very well both when hidden terminals are present (e) and absent (g). Crucially, in the low-congestion regime, observe is η is close to one ($\geq 85\%$) for both the links e and g. All the basic insights above continue to hold even in the presence of hidden terminals. However, incorporating hidden terminals has an impact on the performance ratio of our LP, which would become proportional to the ratio of maximum and minimum link capacities in any neighborhood, i.e., $\max_e \frac{\max_{e' \in I(e)} c(e')}{\min_{e' \in I(e)} c(e')}$.

A Provably good Linear Program: Recall from Section 9.3 that the only non-linear constraints in our NLP are (9.5) and (9.7). We now use the consequences of Theorem 78 in order to "linearize" these. The stability equation (9.5) always holds under the low-congestion constraints. Hence, (9.5) can be replaced by (9.8).

The constraints (9.7) are more involved, but they can be linearized in the following cases. First, for the idle sense protocol [57], the number of backoff stages is one, and therefore the constraints (9.7) are actually linear. Next, in the case of 802.11, in the low congestion regime (9.8), and if the channel error probabilities $1 - \rho(e)$ are small, we can again show that (9.7) can be linearized. Thus, in these two cases, the NLP can be replaced by the LP of Fig. 9.8, with these additional linear constraints.

$$\text{Maximize} \sum_i r(i) \qquad \text{subject to} \qquad (9.17)$$

$$\forall i \in \{1, \ldots, k\}, r(i) = \sum_{p \in \mathcal{P}_i} r(p) \qquad (9.18)$$

$$\forall e, f(e) = \sum_{e \in p} r(p) \qquad (9.19)$$

$$\forall e, \frac{f(e)}{c(e)} + \sum_{e' \in I(e)} \frac{f(e')}{c(e')} \leq \epsilon \qquad (9.20)$$

Figure 9.8: Throughput maximization linear program for random-access wireless networks

How does the throughput yielded by our LP in Figure 9.8 compare with the maximum achievable throughput? The crucial constraint in Figure 9.8 is the low-congestion constraint (9.20); while, the set $I(e)$ could be very large, constraint (9.20) restricts the total utilization of all the links in $I(e)$ to be at most ϵ. This is not a severe restriction; all the links in $I(e)$ interfere with e and hence are close to each other in space. Hence, at most of a few links in $I(e)$ can transmit simultaneously without conflicting with each other. Specifically, for the $Tx - Rx$ model of interference, we showed in Chapter 7 that at most k links in $I(e)$ can be simultaneously active without leading to collisions. Hence, under the conditions discussed earlier, the LP throughput is at most a factor of k/ϵ away from the optimal. We summarize these arguments in the theorem below.

Theorem 79 *Under the Tx-Rx model of interference, the total throughput achievable using the LP in Figure 9.8 is at most a factor of k/ϵ away from the maximum achievable throughput using any protocol, where k and $\epsilon > 0$ are fixed constants.*

The performance guarantee $\frac{k}{\epsilon}$ of Theorem 79 is a loose upper-bound for several reasons. First, it can be tightened by a more careful geometric analysis of the Tx-Rx interference model. Second, if we compare the LP solution with the maximum throughput achievable by any known conflict-free link scheduling protocol (such as [77]), then our solution is only a factor of $\frac{2}{\epsilon}$ away. Third, our guarantee can be improved even further, if we compare it with the optimal throughput achievable by a random-access protocol (i.e., that yielded by our NLP). Most significantly, we note that Theorem 79 implies that by a careful choice of end-to-end connection rates, routes, and channel-access probabilities, we can use a random-access scheduling protocol to achieve throughputs that are within a constant factor of those achievable via centralized, deterministic, conflict-free scheduling protocols.

The constant-factor guarantee shown by Theorem 79 for the Tx-Rx model also holds for other interference models based on wireless geometry such as the Tx-model ([128]), and the Distance-2 Interference model ([14]). If the interference model is arbitrary and non-geometric (i.e., $I(e)$ is an arbitrary subset of E for each link e), the performance ratio of the LP may no longer be within a constant-factor of the optimal solution. However, even if the interference model is non-geometric, Theorem 78 guarantees that the LP solution can be stably scheduled by choosing the access-probabilities carefully.

9.5 Routing Metrics

We now use our analysis from the preceding section for the study of routing metrics for multi-hop 802.11 networks. We quantify what two existing path metrics (ETX [37]

and ETT [38]) represent, and also develop the Available Capacity Metric (ACM) which incorporates the impact of interfering traffic on link quality.

Hop-Count, ETX and ETT: Link-level routing metrics attempt to capture the quality of a given path through a multi-hop wireless network. Used in conjunction with a distributed shortest-path based routing algorithm, path metrics help reveal the end-to-end path which can yield the maximum throughput. Hop-count based shortest path routing is the simplest path metric (unit weight for each link), and simply attempts to minimize the number of hops between the source and destination. Since it does not account for packet loss rates and link capacities, the end-to-end throughput yielded by hop-count based shortest path algorithms have shown to be very far from optimal in wireless networks.

The ETX metric partially rectifies this by accounting for packet loss probabilities across links. Specifically, nodes periodically broadcast probe packets to their neighbors, and each node tracks the fraction of probe packets it has received from each of its neighbors. The ETX of a link (u, v) is computed as follows. Let $\phi(u, v)$ and $\phi(v, u)$ be the fraction of u's probes which have been received by v, and the fraction of v's probes which have been received by u respectively. Then, $ETX(u, v) \stackrel{\text{def}}{=} \frac{1}{\phi(u,v)\cdot\phi(v,u)}$. ETX measures the expected number of times a unicast packet needs to be transmitted across a link before its successful reception, and captures the fact that both data and acknowledgment needs to be sent (in forward and reverse directions) for a successful packet transmission.

The ETT metric builds upon ETX by incorporating link capacity information. If $c(u, v)$ is the capacity of link (u, v) and The ETT metric builds upon ETX by incorporating link capacity information. If $c(u, v)$ is the capacity of link (u, v) and M is the nominal packet size, then $ETT(u, v) \stackrel{\text{def}}{=} \frac{ETX(u,v)\cdot M}{c(u,v)}$. Notice that $\frac{M}{c(u,v)}$ is the amount of time a packet occupies a channel during transmission; hence ETT cap-

tures the expected amount of time *a packet occupies the channel* before its successful reception. The ETT and ETX for a route is simply the sum of the link-level ETT and ETX values respectively.

We make two observations based on these three routing metrics: (i) Since each of them is isotonic (*i.e.*, the metric for $u \rightarrow w$ is the metric for $u \rightarrow v$ plus the metric for $v \rightarrow w$, if v is in the shortest-path from u to w), each of them allows for their straightforward implementation using distributed shortest-path based protocols (such as DSDV [101]). (ii) Greater efficiency comes with incorporating a more precise picture of the traffic near the path in question. To see this second point, observe the trend in path-metric design; hop-count does not account for network traffic was superseded by ETX, which uses packet loss rates in its estimation; this was in turn superseded by ETT, which includes link capacities into its estimation. This motivates the use of our NLP — which takes into account all of these link-level characteristics — as a tool for understanding existing metrics and developing new routing metrics.

NLP and Routing Metrics: We begin by turning to our NLP to answer: what is the available capacity on a link, e, and how do ETX and ETT compare to this? Using our notation from Section 9.3, the expected number of transmissions on an edge $e = (u, v)$ is simply $1/\eta(e)\rho(e)$, the inverse of the probability that a given transmission is successful. This is precisely the value that ETX attempts to estimate. It immediately follows that ETT estimates the time to transmit transmit a packet of size M as $M/\eta(e)\rho(e)c(e)$.

Recall that, in the random backoff model, before a node u is attempts transmission, it checks if the channel is idle; u initiates a back-off timer if the channel is idle, and transmits when the timer expires; if the channel is busy, u freezes its back-off timer and restarts the timer only when the channel is sensed idle again. Thus, the actual time to transmit a packet is composed of three quantities: (i) the total

time it spends occupying the channel, (ii) the total time it spends during back-off (by decrementing the back-off timer), *and (iii) the total time it spends when the sender has frozen the back-off timer, due to a busy channel.* We note that the ETT metric captures (i) alone, while ETT with the back-off time accounted (ETT-with-backoff [38]) captures (i) and (ii). However, when the channel is busy most of the time due to traffic from interfering links, (iii) can dominate both (i) and (ii) and both ETT and ETT-with-backoff can underestimate the actual time to transmit a packet drastically. To capture this phenomenon, we define the quantity $\gamma(u)$, which is the fraction of the time that node u perceives the channel to be busy. In the extreme case where $\gamma(u)$ is 1.0, node u can never attempt a transmission, and hence the available capacity of for any link incident on node u is zero; in general, we can use $\gamma(u)$ to compute the available capacity of link $u \rightarrow v$ as follows:

$$\text{available capacity for } (u,v) = \eta(u)\rho(u) \cdot c(u,v) \cdot \gamma(u) \tag{9.21}$$

To motivate the available capacity defined above, consider the example network in Figure 9.9, in which s needs to choose either the upper or lower path to route packets to t; the lower-path has a capacity of 54Mbps compared to the higher-path whose capacity is 24Mbps; however, the 1Mbps carries an interfering flow which competes only with the lower-path for channel access. As a consequence, the lower path perceives the channel to be busy for a much larger fraction of the time, than the upper-path, and hence has a much lower available capacity than the upper-path. ETT will not capture this fact and will return the lower-path as the better one.

Available Capacity Metric: Observe that each value in (9.21) can be easily measured or estimated by node u. The fraction of time for which the channel is idle, $\gamma(u)$, is completely local information for node u. The capacity, $c(u,v)$ is known, and the rest of (9.21) can be estimated by the same way as in ETX. Using this, we introduce

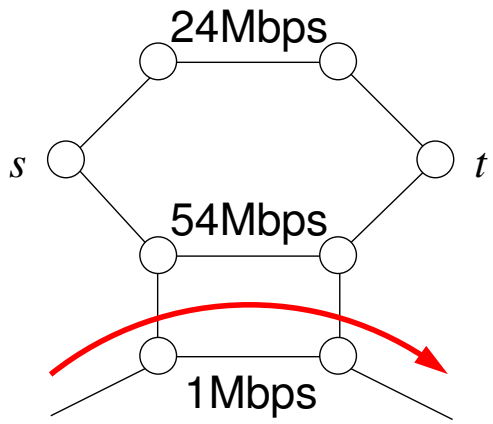


Figure 9.9: A motivating example for a new path metric. ETT will fail to take into account the fact that the 1Mbps link is consuming much of the available capacity of the lower $s-t$ path. All links are 54 Mbps unless otherwise marked.

a new path metric, the *Available Capacity Metric (ACM)*, for link $u \rightarrow v$:

$$\text{ACM}(u,v) \stackrel{\text{def}}{=} \frac{ETT(u,v)}{\gamma(u)} \tag{9.22}$$

Observe that ACM for a link e is defined as the inverse of its available capacity, and hence more closely captures the actual time to transmit a packet across the link. We emphasize that although the name ACM suggests available capacity of a link, the metric value increases as the available capacity decreases (and the link quality degrades), as in ETX and ETT. ACM thus naturally builds upon ETT by also incorporating the effect of interfering traffic on the available capacity, through the parameter $\gamma(u)$.

Experimental Comparison: We now compare the link metrics — shortest path, ETT, ETX, and ACM — experimentally with ns-2 simulations. Our objective is to study how well each of the routing metrics capture the quality of a link and correlate with end-to-end throughputs. We demonstrate that, at a per-link level, ACM more closely captures the actual time to transmit a packet than does ETT and ETT-with-backoff. We also show that, aggregated across network paths, end-to-end throughputs

correlate better with ACM than any other routing metrics.

Experimental Setup. The topology consists of 200 nodes placed uniformly at random in the plane. Motivated by observations from real-world network studies such as [111], we set link bandwidths uniformly at random from the set allowable by 802.11a. We use a set of 15 randomly chosen source-destination pairs, and use a set of manually chosen routes to establish connections between the source-destination pairs. All sources inject UDP traffic into the network at the same rate, with the total injected load steadily increasing with time. All packets are of fixed size $M = 512$ Bytes. In this simulation, the channel-error probability for each link is chosen uniformly at random from $[0, 5]\%$. We record the actual time to transmit, and the success probability for each link. We also track the average fraction of the time each node perceives the channel as busy over time. Each of these is measured directly by monitoring events at the MAC layer.

Accurate Representation of Transmission Time: We begin our experimental evaluation of the path metrics by asking: how well does it model the quality of a given link? To answer this, we compare ACM, ETT, and ETT-with-backoff with the mean time to transmit a packet successfully on a given link. First, we present a representative run on a single link in the network in Figure 9.10. It is clear that ACM is significantly more accurate than ETT and ETT-with-backoff in approximating the actual time to transmit across a link. Both ACM and ETT-with-back off closely capture the short-term peaks and valleys in packet transmission times; however, the mean packet transmission times are much closer to the mean ACM values than the mean of the other two metrics.

Next, we choose a random point in time during the simulation and calculate the per-link errors relative to the actual time to transmit, presented in Figure 9.11. The relative error for a particular metric (say ACM) for a link e is measured as follows:

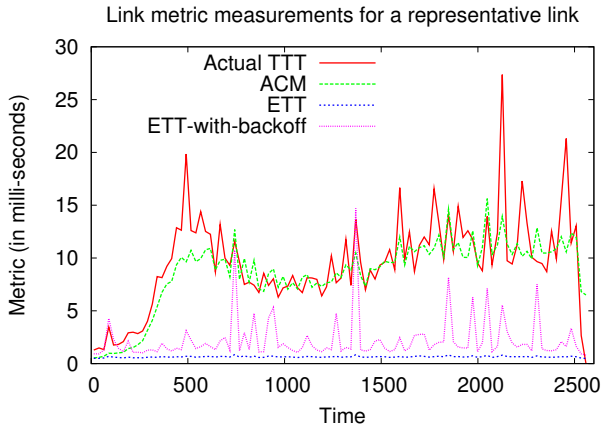


Figure 9.10: A representative run, showing that ACM captures the actual time to transmit more closely than ETT and ETT-with-backoff.

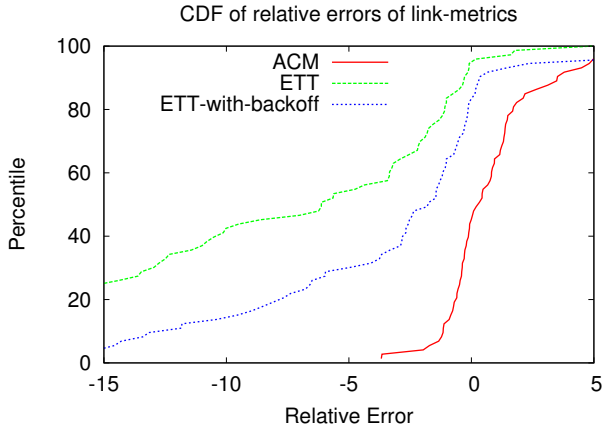


Figure 9.11: CDF of relative error for each metric, computed at a fixed random time in the simulation.

rel-error-acm(e) $= \frac{acm(e)-att(e)}{\min\{acm(e),att(e)\}}$. This definition ensures that an over-estimation and under-estimation by the same factor w have the same relative error $w-1$ (in absolute value). ETT underestimates the actual transmit time, by more than factor of 5 for more than 50% of the links; ETT-with-backoff underestimates the actual time to transmit, by more than a factor 5 for more than 25% of the links; by total contrast, the relative error of ACM for more than 60% of the links is less than a factor of one.

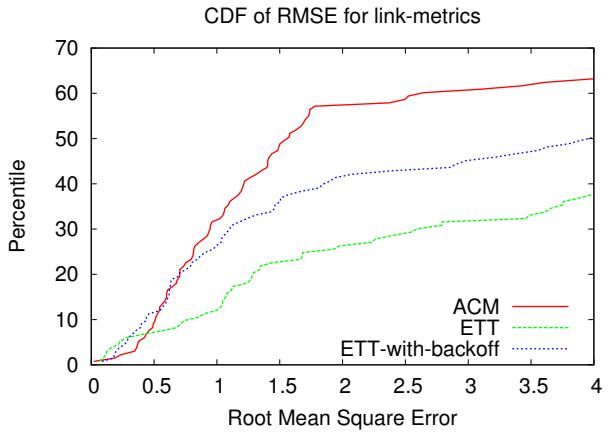


Figure 9.12: CDF of Root Mean Squared Error (RMSE) for each metric. The root mean-squared error tracks the relative error of a metric throughout the simulation.

To present a single error value for each link through out the simulation, we calculate the Root Mean Squared Error of a link as follows. For a specific metric (say ACM), the RMSE of ACM for link $e = \sqrt{\frac{\sum_i relative-error-acm_i(e)^2}{k}}$, where $relative-error-acm_i(e)$ is the relative error of ACM for link e during second i of the simulation, and k is the total number of seconds. As Figure 9.12 demonstrates, ACM has a much better error profile across links throughout the simulation than ETT or ETT-with-backoff.

Routing Metrics and End-to-End throughput: We have shown that ACM yields a very accurate measure of a link's available capacity and actual time to transmit a packet. However, providing a more accurate representation of links' quality does not immediately imply a better path metric. Instead, one must consider the relative

Metric	Correlation Coefficient
ACM	**0.5064**
ETX	0.4825
ETT-with-backoff	0.4491
Bottleneck-ACM	0.4005
Hop-count	0.3549
ETT	0.2914

Table 9.2: ACM correlates much better with end-to-end throughputs indicating that a smaller ACM is much more indicative of higher end-to-end throughputs, than for any other metric

ranking of paths that the metric would return. In this experiment, we measure how closely the various metrics correlate the throughputs yielded by the paths. We record the metrics for each source-destination path in the experiment (for each of hop count, ETX, ETT, ETT-with-backoff, and ACM) and also the end-to-end throughput yielded by each of the paths. In Table 9.2, we present the correlation coefficient between the throughput achieved by a path and the *inverse* of its path-metric value. A correlation coefficient of 1.0 implies that there is a perfect correspondence between how the metric ranks the paths and how end-to-end throughput ranks the paths. A value of 0.0 implies no correlation between end-to-end throughput and routing metric. Of all the metrics, the 1/ACM correlates much more strongly (correlation coefficient ≈ 0.5) with end-to-end throughputs than all the other metrics, while 1/ETX follows as a close next.

9.6 Related Work

1. Path Metrics: The initial routing protocols proposed for multi-hop wireless networks, such as AODV, DSR, DSDV, etc., essentially used hop-count as the metric for path selection. Hop-count based path selection has several disadvantages. For instance, reducing hop-count results in an increase in the average distance between the transmitter and receiver in each link. This results in reduced signal-strength at

the receiver which results in high packet-loss rates per link. ETX [37] was proposed to remedy this. ETX associates a link-weight with each metric which is the expected number of times a packet needs to be transmitted across the link before successful reception. The value is obtained through periodic link-probes. The routing algorithm now chooses paths with least weights as defined by ETX. ETX still has two problems: it does not take into account the bandwidth of the individual links (a low-bandwidth link with low-loss rate is not necessary desirable compared to high-bandwidth link with high-loss rates). Next, ETX does not take into link capacities. ETT [38] was introduced to remedy this. However, while ETT aims to combine loss rates with link capacities, it does not account for interference related congestion. An excellent evaluation of these as well as related metrics can be found in [127]. Our work analyzes all these metrics in a rigorous setting and also builds upon them to develop ACM.

2. Single-hop Networks: There is a very large body of work which deals with analysis and optimization of throughput and fairness in single-cell random-access wireless networks (i.e., networks that are isomorphic to cliques which can support at most one transmission at any time; e.g., a wireless LAN environment). Several authors have analyzed the performance of 802.11 MAC protocol and proposed enhancements to it with the objective of maximizing throughout and/or fairness [24, 26, 25, 69]. The Idle-sense protocol [57] proposed recently overcomes many performance-related problems of 802.11 such as high-contention overhead, short-term unfairness, low throughput, etc. (see the related works section in [57] for an excellent survey of results which deal with the analysis and optimization of random-access protocols). None of them deal with routing or throughput optimization in multi-hop ad hoc wireless networks.

3. End-to-End Cross-layer Optimization: Cross-layer optimization deals with the design of transport, routing, and scheduling protocols whose joint performance is guaranteed to be close to network capacity [72, 73, 61, 77]. Unlike most existing literature in cross-layer optimization which assume centralized, conflict-free schedul-

ing protocols, our work aims to model and optimize the throughput performance in practical settings such as wireless ad hoc/sensor/mesh networks which employ random-access scheduling protocols. Our goal is to design efficient throughput optimization and path-selection (routing) algorithms subject to the scheduling constraints imposed by a given MAC protocol such as 802.11.

Chapter 10

Conclusions

10.1 Summary of Contributions

In this thesis, we explored resource allocation challenges that arise in a diverse collection of networked and distributed environments. We dealt with assignment problems in the first part of this thesis. The common goal across all the problems studied here was to obtain a minimum cost assignment of a set of objects to a set of locations, without violating the capacity constraints of the locations. Our central contribution here was a single assignment scheme founded on a combination of linear algebraic principles and randomization, which handles a broad range of assignment problems. We showed how to relate this with other ideas, to derive multi-criteria approximation algorithms for unrelated parallel machine scheduling, social network modeling, and broadcast scheduling. In the second part of the thesis, we dealt with two precedence-constrained scheduling problems, namely sweep scheduling and tree scheduling, which arise in the context of high performance scientific computing applications. For the sweep scheduling problem, our solution featured an application of the random-delays technique along with random processor assignment; for the tree scheduling problem, we combined random-delays with a new tree-decomposition technique. In the final

part of the thesis, we dealt with latency minimization and capacity estimation in wireless networks. The central contribution here was our novel geometric insights for characterizing the properties of wireless interference, and the fusion of geometry with the randomization & linear programming. Thus, the creative use of linear programming and probabilistic techniques for resource allocation along with application-specific ideas is the broad theme and overarching contribution of this thesis.

10.2 Future Directions

We now survey some specific open problems as well as some broad directions for future research that are inspired by this thesis.

Assignment and scheduling:

- At the heart of our assignment algorithms in Chapters 3 and 4 was the dependent randomized rounding approach, which shows how to round a fractional assignment probabilistically while preserving the marginal distribution property, and not violating the cardinality / capacity constraints for objects or locations. A broad direction for future research is to develop an equivalent rounding approach for online optimization problems. In particular, if the objects are revealed only one at a time, and we need to assign an object to a location as soon as it (and its fractional assignment) is revealed, how well can we trade-off the marginal distribution and capacity preservation properties?

- In Chapter 3, we presented a bi-criteria algorithm for simultaneously optimizing makespan and weighted completion time to within a factor of $(2, \frac{3}{2})$ in the setting of unrelated parallel machine scheduling. Improving either of these two components remains a tantalizing open problem. Semidefinite programming seems a promising avenue for improving the weighted completion time guarantee

beyond $\frac{3}{2}$.

- In the context of the broadcast scheduling problem, dependent rounding yields essentially the best possible result under the resource augmentation framework. Bansal *et al.* [16] have made exciting progress for this problem, by presenting a 1-speed polylogarithmic approximation algorithm. A challenging open question here is: does there exist a constant factor approximation algorithm (or a hardness of approximation result on the contrary) for the broadcast scheduling problem?

- The general setting of scheduling under unrelated parallel machines under precedence constraints remains poorly understood. The random-delays technique of Leighton *et al.*[82, 83] addressed job-shop scheduling with chain-like precedences; our results in Chapter 6 addressed tree-like precedences. Handling any other type of constraints remains an interesting open problem. In particular, is it possible to device good approximation algorithms when the precedence constraints have a fixed tree-width?

Wireless communication:

- Leighton *et al.*[82, 83] derive a constant factor approximation for the job-shop scheduling problem through a repeated application of random-delays and the local lemma. Is it possible to derive such a constant-factor approximation algorithm for end-to-end packet scheduling under disk graphs with wireless interference constraints? In Chapter 8, we presented centralized LP-based routing and scheduling algorithms which are guaranteed to operate a wireless network close to its capacity. Do there exist polynomial time *distributed* algorithms which achieve the same goal?

- Almost all known results for wireless capacity estimation, including our results presented in Chapter 7, assume that interference can be modeled using conflict-

graphs: i.e., interference prohibits specific pairs of links in the network from being active simultaneously. A more realistic way of modeling interference is to use signal-to-noise-ratio (SNR) based constraints: interference occurs at a receiver, when the ratio between the signal (from its intended sender) to the total noise (due to ambient conditions and other simultaneous transmitters) drops below a certain threshold. In such a setting, conflict-hypergraphs which prohibit sets of links from transmitting simultaneously (rather than pairs) is a more general tool for dealing with interference. A major open problem is the estimation of network capacity under this general SNR-based interference assumptions.

- Current work in wireless capacity focuses mainly on the design of wireless communication protocols which are guaranteed to achieve the capacity of the network. A broad direction for future research is the joint design of wireless networks and communication protocols. How can we build / configure a wireless network (i.e., how can we determine the positions of the nodes within the network) such that the network is guaranteed to have a high capacity to start with?

Bibliography

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[2] N. Alon, Y. Azar, G. J. Woeginger, and T. Yadid. Approximation schemes for scheduling. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pages 493–500, 1997.

[3] N. Amato and P. An. Task scheduling and parallel mesh-sweeps in transport computations, Jan 2000. Technical Report, TR00-009, Department of Computer Science, Texas A&M University.

[4] Konstantin Andreev, Bruce M. Maggs, Adam Meyerson, and Ramesh K. Sitaraman. Designing overlay multicast networks for streaming. In *SPAA '03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 149–158, New York, NY, USA, 2003. ACM Press.

[5] Matthew Andrews and Lisa Zhang. Scheduling over a time-varying user-dependent channel with applications to high-speed wireless data. *J. ACM*, 52(5):809–834, 2005.

[6] J. Aslam, A. Rasala, C. Stein, and N. Young. Improved bicriteria existence theorems for scheduling. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pages 846–847, 1999.

[7] F. Meyer auf der Heide and B. Vöcking. A packet routing protocols for arbitrary networks. *Lecture Notes in Computer Science*, 439:291–302, 1995.

[8] B. Awerbuch, Y. Azar, E. F. Grove, M.-Y. Kao, P. Krishnan, and J. S. Vitter. Load balancing in the L_p norm. In *IEEE Symposium on Foundations of Computer Science*, pages 383–391, 1995.

[9] Y. Azar and A. Epstein. Convex programming for scheduling unrelated parallel machines. In *Proc. of the ACM Symposium on Theory of Computing*, pages 331–337, 2005.

[10] Y. Azar, L. Epstein, Y. Richter, and G. J. Woeginger. All-norm approximation algorithms. *J. Algorithms*, 52(2):120–133, 2004.

[11] Y. Azar and S. Taub. All-norm approximation for scheduling on identical machines. In *SWAT*, pages 298–310, 2004.

[12] D. J. Baker, J. E. Wieselthier, and A. Ephremides. A distributed algorithm for scheduling the activation of links in self-organizing mobile radio networks. In *IEEE Int. Conference Communications*, pages 2F6.1–2F6.5, 1982.

[13] H. Balakrishnan. *Challenges in Reliable Data Transport Over Heterogeneous Wireless Networks*. Ph.D. Thesis, Department of Computer Science, University of California at Berkeley, 1998.

[14] H. Balakrishnan, C. Barrett, V. S. Anil Kumar, M. Marathe, and S. Thite. Induced matchings and its relationship to maximum instantaneous capacity of media access layer. manuscript.

[15] H. Balakrishnan, C. Barrett, V. S. Anil Kumar, M. Marathe, and S. Thite. The distance-2 matching problem and its relationship to the mac layer capacity of

ad-hoc wireless networks. *IEEE J. Selected Areas in Communications*, 22(6), August 2004.

[16] Nikhil Bansal, Don Coppersmith, and Maxim Sviridenko. Improved approximation algorithms for broadcast scheduling. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 344–353, New York, NY, USA, 2006. ACM Press.

[17] Nikhil Bansal and Zhen Liu. Capacity, Delay and Mobility in Wireless Ad-Hoc Networks. In *IEEE INFOCOM 2003*, San Francisco, CA, April 1–3 2003.

[18] Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph Naor, and Baruch Schieber. A unified approach to approximating resource allocation and scheduling. *J. ACM*, 48(5):1069–1090, 2001.

[19] Amotz Bar-Noy, Sudipto Guha, Yoav Katz, Joseph Naor, Baruch Schieber, and Hadas Shachnai. Throughput maximization of real-time scheduling with batching. In *Proc. Thirteenth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 742–751, 2002.

[20] C. L. Barrett, M. Drozda, A. Marathe, and M. V. Marathe. Characterizing the interaction between routing and MAC protocols in ad-hoc networks. In *Proceedings of the ACM Symposium on Mobile Ad Hoc Networking and Computing*, pages 92–103, 2002.

[21] Yair Bartal and S. Muthukrishnan. Minimizing maximum response time in scheduling broadcasts. In *Proc. Eleventh annual ACM-SIAM Symposium on Discrete Algorithms*, pages 558–559, 2000.

[22] Yigal Bejerano, Seung-Jae Han, and Li (Erran) Li. Fairness and load balancing in wireless lans using association control. In *MobiCom '04: Proceedings of*

the 10th annual international conference on Mobile computing and networking, pages 315–329, New York, NY, USA, 2004. ACM Press.

[23] D. Bertsimas, C.-P. Teo, and R. Vohra. On dependent randomized rounding algorithms. *Operations Research Letters*, 24(3):105–114, 1999.

[24] G. Bianchi. Performance analysis of the ieee 802.11 distributed coordination function. *IEEE Journal on Selected Areas in Communications*, 18(3):535–547, 2000.

[25] Luciano Bononi, Marco Conti, and Enrico Gregori. Runtime optimization of ieee 802.11 wireless lans performance. *IEEE Transactions on Parallel and Distributed Systems*, 15(1):66–80, 2004.

[26] Frederico Cali, Marco Conti, and Enrico Gregori. Dynamic tuning of the ieee 802.11 protocol to achieve a theoretical throughput limit. *IEEE/ACM Trans. Netw.*, 8(6):785–799, 2000.

[27] A. K. Chandra and C. K. Wong. Worst-case analysis of a placement algorithm related to storage allocation. *SIAM J. on Computing*, 4(3):249–263, 1975.

[28] C. Chekuri and M. Bender. An efficient approximation algorithm for minimizing makespan on uniformly related machines. *Journal of Algorithms*, pages 212–224, 2001.

[29] C. Chekuri and S. Khanna. *Approximation algorithms for minimizing weighted completion time*. CRC Press, 2004.

[30] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23:493–509, 1952.

[31] F. A. Chudak and D. B. Shmoys. Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds. *J. Algorithms*, 30(2):323–343, 1999.

[32] B. Clark, C. Colbourn, and D. Johnson. Unit disk graphs. *Discrete Mathematics*, 86:165–177, 1990.

[33] C. Cooper and A. Frieze. On a general model of web graphs. In *Proc. European Symposium on Algorithms*, pages 500–511, 2001.

[34] Colin Cooper and Alan Frieze. Crawling on web graphs. In *Proc. Thiry-fourth annual ACM Symposium on Theory of computing*, pages 419–427. ACM Press, 2002.

[35] Colin Cooper and Alan Frieze. Crawling on simple models of web graphs. *Internet Mathematics*, 1:57–90, 2003.

[36] Colin Cooper and Alan Frieze. A general model of web graphs. *Random Struct. Algorithms*, 22(3):311–335, 2003.

[37] Douglas S. J. De Couto, Daniel Aguayo, John Bicket, and Robert Morris. A High-Throughput Path Metric for Multi-Hop Wireless Routing. In *Proceedings of Mobicom*, pages 134–146. ACM Press, 2003.

[38] Richard Draves, Jitendra Padhye, and Brian Zill. Routing in multi-radio, multi-hop wireless mesh networks. In *MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 114–128, New York, NY, USA, 2004. ACM Press.

[39] Thomas Erlebach and Alexander Hall. Np-hardness of broadcast scheduling and inapproximability of single-source unsplittable min-cost flow. *Journal of Scheduling*, 7:223 – 241, may 2004.

[40] S. Eubank, H. Guclu, V. S. A. Kumar, M. V. Marathe, Srinivasan A., Z. Toroczkai, and N. Wang. Modeling disease outbreaks in realistic urban social networks. *Nature*, 429(6988):180–184, May 2004.

[41] S. Even. *Graph Algorithms*. Computer Science Press, 1979.

[42] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM '99: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, pages 251–262, New York, NY, USA, 1999. ACM Press.

[43] Uriel Feige and Christian Scheideler. Improved bounds for acyclic job shop scheduling. *Combinatorica*, 22(3):361–399, 2002.

[44] R. Gandhi, S. Khuller, Y. Kim, and Y. C. Wan. Algorithms for minimizing response time in broadcast scheduling. *Algorithmica*, 38(4):597–608, January 2004.

[45] Rajiv Gandhi, Samir Khuller, Srinivasan Parthasarathy, and Aravind Srinivasan. Dependent rounding and its applications to approximation algorithms. In *Journal of the ACM (JACM), to appear; preliminary version appears in IEEE Symposium on Foundations of Computer Systems (FOCS), 2005.*

[46] Naveen Garg and Jochen Koenemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, page 300. IEEE Computer Society, 1998.

[47] Michael Gastpar and Martin Vetterli. On The Capacity of Wireless Networks: The Relay Case. In *IEEE INFOCOM 2002*, New York, NY, June 23–27 2002.

[48] A. Goel and A. Meyerson. Simultaneous optimization via approximate majorization for concave profits or convex costs. Tech. Report CMU-CS-02-203, December 2002, Carnegie-Mellon University.

[49] Leslie Ann Goldberg, Mike Paterson, Aravind Srinivasan, and Elizabeth Sweedyk. Better approximation guarantees for job-shop scheduling. *SIAM J. Discrete Math.*, 14(1):67–92, 2001.

[50] F. Chung Graham and L. Lu. Connected components in random graphs with given degree sequences. *Annals of Combinatorics*, 6:125–145, 2002.

[51] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[52] Matthias Grossglauser and David N. C. Tse. Mobility increases the capacity of ad hoc wireless networks. *IEEE/ACM Trans. Netw.*, 10(4):477–486, 2002.

[53] Piyush Gupta and P. R. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, 46(2):388–404, March 2000.

[54] B. Hajek and G. Sasaki. Link scheduling in polynomial time. *IEEE Transactions on Information Theory*, 34:910–917, 1988.

[55] L. Hall. Approximation algorithms for scheduling. in *Approximation Algorithms for NP-Hard Problems*, Edited by D. S. Hochbaum. PWS Press, 1997.

[56] Leslie A. Hall, Andreas S. Schulz, David B. Shmoys, and Joel Wein. Scheduling to minimize average completion time: off-line and on-line approximation algorithms. *Mathematics of Operations Research*, 22(3):513–544, 1997.

[57] Martin Heusse, Franck Rousseau, Romaric Guillier, and Andrzej Duda. Idle

sense: an optimal access method for high throughput and fairness in rate diverse wireless lans. *SIGCOMM Comput. Commun. Rev.*, 35(4):121–132, 2005.

[58] W. Hoeffding. Probability inequalities for sums of bounded random variables. *American Statistical Association Journal*, 58:13–30, 1963.

[59] J.A. Hoogeveen, J.K. Lenstra, and B. Veltman. Three, four, five, six, or the complexity of scheduling with communication delays. *Operations Research Letters*, 16:129–137, 1994.

[60] Jing-Jang Hwang, Yuan-Chieh Chow, Frank D. Anger, and Chung-Yee Lee. Scheduling precedence graphs in systems with interprocessor communication times. *SIAM J. Comput.*, 18(2):244–257, 1989.

[61] Kamal Jain, Jitendra Padhye, Venkata N. Padmanabhan, and Lili Qiu. Impact of interference on multi-hop wireless network performance. In *Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 66–80. ACM Press, 2003.

[62] Klaus Jansen and Lorant Porkolab. Improved approximation schemes for scheduling unrelated parallel machines. In *STOC '99: Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 408–417, New York, NY, USA, 1999. ACM Press.

[63] Klaus Jansen and Roberto Solis-Oba. Approximation algorithms for scheduling jobs with chain precedence constraints. In *PPAM*, pages 105–112, 2003.

[64] Klaus Jansen, Roberto Solis-Oba, and Maxim Sviridenko. Makespan minimization in job shops: a polynomial time approximation scheme. In *STOC '99: Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 394–399, New York, NY, USA, 1999. ACM Press.

[65] B. Kalyanasundaram, K. Pruhs, and M. Velauthapillai. Scheduling broadcasts in wireless networks. In *Proc. European Symposium of Algorithms, LNCS 1879, Springer-Verlag*, pages 290–301, 2000.

[66] K. Kar, M. Kodialam, T. V. Lakshman, and L. Tassiulas. Routing for network capacity maximization in energy-constrained ad-hoc networks. In *IEEE INFOCOM*, 2003.

[67] D. Karger, C. Stein, and J. Wein. Scheduling algorithms. Chapter in the *Handbook of Algorithms*, CRC Press, 1997.

[68] Samir Khuller, Yoo-Ah Kim, and Yung-Chun (Justin) Wan. Algorithms for data migration with cloning. In *PODS '03: Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 27–36, New York, NY, USA, 2003. ACM Press.

[69] Hwangnam Kim and Jennifer C. Hou. Improving protocol capacity with model-based frame scheduling in ieee 802.11-operated wlans. In *MOBICOM*, pages 190–204, 2003.

[70] Jon Kleinberg, Yuval Rabani, and Éva Tardos. Fairness in routing and load balancing. *J. Comput. Syst. Sci.*, 63(1):2–20, 2001.

[71] K.R. Koch, R.S. Baker, and R.E. Alcouffe. A parallel algorithm for $3d\ s_n$ transport sweeps. Technical Report *LA-CP-92406*, Los Alamos National Laboratory, 1992.

[72] Murali Kodialam and Thyaga Nandagopal. Characterizing achievable rates in multi-hop wireless networks: the joint routing and scheduling problem. In *Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 42–54. ACM Press, 2003.

[73] Murali Kodialam and Thyaga Nandagopal. Characterizing achievable rates in multi-hop wireless mesh networks with orthogonal channels. *IEEE/ACM Trans. Netw.*, 13(4):868–880, 2005.

[74] Murali Kodialam and Thyaga Nandagopal. Characterizing the capacity region in multi-radio multi-channel wireless mesh networks. In *MobiCom '05: Proceedings of the 11th annual international conference on Mobile computing and networking*, pages 73–87, New York, NY, USA, 2005. ACM Press.

[75] Ulas C. Kozat and Leandros Tassiulas. Throughput capacity of random ad hoc networks with infrastructure support. In *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 55–65, New York, NY, USA, 2003. ACM Press.

[76] V. S. Anil Kumar, Madhav V. Marathe, Srinivasan Parthasarathy, and Aravind Srinivasan. End-to-end packet scheduling in ad hoc networks. In *ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1014–1023, 2004.

[77] V. S. Anil Kumar, Madhav V. Marathe, Srinivasan Parthasarathy, and Aravind Srinivasan. Algorithmic aspects of capacity in wireless networks. In *ACM SIGMETRICS*, pages 133–144, 2005.

[78] V. S. Anil Kumar, Madhav V. Marathe, Srinivasan Parthasarathy, and Aravind Srinivasan. Approximation algorithms for scheduling in multiple machines. In *IEEE Symposium on Foundations of Computer Systems (FOCS)*, 2005.

[79] V. S. Anil Kumar, Madhav V. Marathe, Srinivasan Parthasarathy, Aravind Srinivasan, and Sibylle Zust. Provable algorithms for parallel generalized sweep scheduling. In *Journal of Parallel and Distributed Computing (JPDC), to appear; preliminary version appears in IEEE Parallel and Distributed Computing Symposium (IPDPS), 2005.*

[80] V. S. Anil Kumar, Srinivasan Parthasarathy, Madhav V. Marathe, and Aravind Srinivasan. Scheduling on unrelated machines under tree-like precedence constraints. In *a special issue of Algorithmica, invited contribution – to appear; preliminary version appears in APPROX, 2005. Lecture Notes in Computer Science.*, pages 146–157.

[81] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. *Sequencing and scheduling: algorithms and complexity.* Elsevier, 1993.

[82] F. T. Leighton, B. M. Maggs, and A. W. Richa. Fast algorithms for finding o(congestion + dilation) packet routing schedules. *Combinatorica*, 19(2):1–27, 1999.

[83] T. Leighton, B. Maggs, and S. Rao. Packet routing and job shop scheduling in o(congestion+dilation) steps. *Combinatorica*, 14(2):167–180, 1994.

[84] J. K. Lenstra, D. B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, pages 259–271, 1990.

[85] Stefano Leonardi and Danny Raz. Approximating total flow time on parallel machines. In *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 110–119, New York, NY, USA, 1997. ACM Press.

[86] Jyh-Han Lin and Jeffrey Scott Vitter. e-approximations with minimum packing constraint violation (extended abstract). In *STOC '92: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 771–782, New York, NY, USA, 1992. ACM Press.

[87] Nathan Linial, Avner Magen, and Michael E. Saks. Trees and euclidean metrics.

In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 169–175, New York, NY, USA, 1998. ACM Press.

[88] Benyuan Liu, Zhen Liu, and Don Towsley. On the Capacity of Hybrid Wireless Networks. In *IEEE INFOCOM 2003*, San Francisco, CA, April 1–3 2003.

[89] Michael Luby. Removing randomness in parallel computation without a processor penalty. *Journal of Computer and System Sciences*, 47(2):250–286, 1993.

[90] Mark M. Mathis, Nancy M. Amato, and Marvin L. Adams. A general performance model for parallel sweeps on orthogonal grids for particle transport calculations. In *ICS '00: Proceedings of the 14th international conference on Supercomputing*, pages 255–263, New York, NY, USA, 2000. ACM Press.

[91] Mark M. Mathis, Nancy M. Amato, and Marvin L. Adams. A general performance model for parallel sweeps on orthogonal grids for particle transport calculations. In *ICS '00: Proceedings of the 14th international conference on Supercomputing*, pages 255–263, New York, NY, USA, 2000. ACM Press.

[92] Mark M. Mathis and Darren J. Kerbyson. A general performance model of structured and unstructured mesh particle transport computations. *J. Supercomput.*, 34(2):181–199, 2005.

[93] Friedhelm Meyer auf de Heide, Christian Schindelhauer, Klaus Volbert, and Matthias Grnewald. Energy, congestion and dilation in radio networks. In *Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*, pages 230–237. ACM Press, 2002.

[94] Adam Meyerson. Profit-earning facility location. In *STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 30–36, New York, NY, USA, 2001. ACM Press.

[95] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge University Press, New York, NY, USA, 1995.

[96] A. Munier and C. Hanen. An approximation algorithm for scheduling unitary tasks on m processors with communication delays. Internal Report LITP 12, Université P. et M. Curie.

[97] Rafail Ostrovsky and Yuval Rabani. Universal o(congestion + dilation + log1+&egr;n) local control packet switching algorithms. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 644–653. ACM Press, 1997.

[98] A. Panconesi and A. Srinivasan. Randomized distributed edge coloring via an extension of the chernoff-hoeffding bounds. *SIAM J. Comput.*, 26:350–368, 1997.

[99] S. D. Pautz. An algorithm for parallel sn sweeps on unstructures meshes. *Journal of Nuclear Science and Engineering*, 140:111–136, 2002.

[100] Christina Peraki and Sergio D. Servetto. On the maximum stable throughput problem in random networks with directional antennas. In *MobiHoc '03: Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, pages 76–87, New York, NY, USA, 2003. ACM Press.

[101] Charles E. Perkins and Pravin Bhagwat. Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers. In *Proceedings of ACM SIGCOM Conference*, pages 234–244, 1994.

[102] S. Plimpton, B. Hendrickson, S. Burns, W. McLendon III, and L. Rauchwerger. Parallel s_n sweeps on unstructured grids: Algorithms for prioritization, grid partitioning, and cycle detection. *Journal of American Nuclear Society*, 150(3):267–283, 2005.

[103] S. Plimpton, B. Hendrickson, S. Burns, W. McLendon III, and L. Rauchwerger. Parallel s_n sweeps on unstructured grids: Algorithms for prioritization, grid partitioning, and cycle detection. *Journal of American Nuclear Society*, 150(3):267–283, 2005.

[104] Steve Plimpton, Bruce Hendrickson, Shawn Burns, and III Will McLendon. Parallel algorithms for radiation transport on unstructured grids. In *Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, page 25, Washington, DC, USA, 2000. IEEE Computer Society.

[105] M. Queyranne and M. Sviridenko. Approximation algorithms for shop scheduling problems with minsum objective. *Journal of Scheduling*, 5:287–305, 2002.

[106] Yuval Rabani and Éva Tardos. Distributed packet switching in arbitrary networks. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 366–375. ACM Press, 1996.

[107] Bozidar Radunovic and Jean-Yves Le Boudec. Rate performance objectives of multihop wireless networks. *IEEE Trans. on Mobile Computing*, 3(4):334–349, 2004.

[108] Prabhakar Raghavan and Clark D. Tompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.

[109] Arjunan Rajeswaran and Rohit Negi. Capacity of power constrained ad-hoc networks. In *INFOCOM*, 2004.

[110] V.T. Rayward-Smith. Uet scheduling with interprocessor communication delays. *Discrete Applied Mathematics*, 18(1):55–71, 1987.

[111] Roofnet. `http://pdos.csail.mit.edu/roofnet/doku.php`.

[112] Siuli Roy, Dola Saha, S. Bandyopadhyay, Tetsuro Ueda, and Shinsuke Tanaka. A network-aware mac and routing protocol for effective load balancing in ad hoc wireless networks with directional antenna. In *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, 2003.

[113] E. Royer, S. Lee, and C. Perkins. The effects of mac protocols on ad hoc network communications. In *Proc. IEEE Wireless Communications and Networking Conference*, September 2000.

[114] C. Scheideler. *Universal routing strategies for interconnection networks*, volume 1390. Springer Verlag, 1998.

[115] A. Schulz and M. Skutella. The power of α-points in preemptive single machine scheduling. *Journal of Scheduling*, 5(2):121–133, 2002.

[116] P. Schuurman and G. J. Woeginger. Polynomial time approximation algorithms for machine scheduling: Ten open problems. *J. Scheduling*, pages 203–213, 1999.

[117] D. B. Shmoys and É. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, pages 461–474, 1993.

[118] David B. Shmoys, Clifford Stein, and Joel Wein. Improved approximation algorithms for shop scheduling problems. *SIAM Journal on Computing*, 23(3):617–632, 1994.

[119] M. Skutella. Convex quadratic and semidefinite relaxations in scheduling. *Journal of the ACM*, 46(2):206–242, 2001.

[120] W. E. Smith. Various optimizers for single-stage production. *Nav. Res. Log. Q.*, pages 59–66, 1956.

[121] A. Srinivas and E. Modiano. Minimum energy disjoint path routing in wireless

ad-hoc networks. In *Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 122–133, 2003.

[122] Aravind Srinivasan. Approximation algorithms via randomized rounding: a survey. *Lectures on Approximation and Randomized Algorithms (M. Karonski and H. J. Promel, editors), Series in Advanced Topics in Mathematics*, pages 9–71, 1999.

[123] C. Stein and J. Wein. On the existence of schedules that are near-optimal for both makespan and total weighted completion time. *Operations Research Letters*, 21, 1997.

[124] K. Tang, M. Correa, and M. Gerla. Effects of ad hoc mac layer medium access mechanisms under tcp. *MONET*, 6(4):317–329, 2001.

[125] Transims: Transportation analysis simulation system. `http://transims.tsasa.lanl.gov/`.

[126] Vijay V. Vazirani. *Approximation algorithms*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.

[127] Yaling Yang, Jun Wang, and Robin Kravets. Designing routing metrics for mesh networks. In *First IEEE Workshop on Wireless Mesh Networks (WiMesh)*, 2005.

[128] Su Yi, Yong Pei, and Shivkumar Kalyanaraman. On the capacity improvement of ad hoc wireless networks using directional antennas. In *MobiHoc*, pages 108–116, 2003.

[129] S.H. Yook, H. Jeong, and A. Barabasi. Modeling the internet's large-scale topology. *Proceedings of the National Academy of Sciences*, 99:13382–13386, 2002.