WILLIAM & MARY
CHARTERED 1693

W&M ScholarWorks

Dissertations, Theses, and Masters Projects

Theses, Dissertations, & Master Projects

2004

Lookahead scheduling in a real-time context: Models, algorithms, and analysis

Benjamin J. Coleman
College of William & Mary - Arts & Sciences

Follow this and additional works at: https://scholarworks.wm.edu/etd

Part of the Computer Sciences Commons, and the Operational Research Commons

Recommended Citation
Coleman, Benjamin J., "Lookahead scheduling in a real-time context: Models, algorithms, and analysis" (2004). *Dissertations, Theses, and Masters Projects.* Paper 1539623445.
https://dx.doi.org/doi:10.21220/s2-xrxe-y041

This Dissertation is brought to you for free and open access by the Theses, Dissertations, & Master Projects at W&M ScholarWorks. It has been accepted for inclusion in Dissertations, Theses, and Masters Projects by an authorized administrator of W&M ScholarWorks. For more information, please contact scholarworks@wm.edu.

Lookahead Scheduling in a Real-time Context: Models, Algorithms, and Analysis

---

A Dissertation

Presented to

The Faculty of the Department of Computer Science

The College of William & Mary in Virginia

In Partial Fulfillment

Of the Requirements for the Degree of

Doctor of Philosophy

---

by

Benjamin J. Coleman

2004

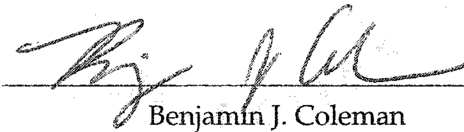
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPROVAL SHEET

This dissertation is submitted in partial fulfillment of

the requirements for the degree of

Doctor of Philosophy



Benjamin J. Coleman

Approved by the Committee, May 2004

Weizhen Mao, Chair

Gianfrano Ciardo

Phil Kearns

Rex Kincaid

Pinar Keskinocak
Georgia Institute of Technology

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

to my grandmother, Edna Bazura, who taught me the most important lesson in life:

how to be happy with myself.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table of Contents

Acknowledgments viii

List of Tables ix

List of Figures x

Abstract xvi

1 Introduction 2

1.1 Resource Assignment Problems 2

1.2 Algorithms 5

1.2.1 Online and Offline Algorithms 6

1.2.2 Semi-Online Algorithms 7

1.2.3 Real-Time Algorithms 8

1.3 Contributions 9

1.3.1 Results 9

2 Related Research 11

2.1 Motivation 11

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.2 Online and Semi-Online Algorithms 12

2.3 Algorithm Analysis 14

2.4 Job Scheduling 15

2.5 Lookahead Scheduling 18

2.6 Scheduling Models 19

2.6.1 Single Machine Scheduling 20

2.6.2 Identical Parallel Machine Scheduling 21

2.6.3 Unrelated Machine Scheduling 22

2.7 Discussion 22

3 A Model for Lookahead Algorithms **24**

3.1 Introduction 24

3.2 Queue Design 27

3.2.1 Wait Queue 27

3.2.2 Lookahead Queue 28

3.3 Control Processes 29

3.3.1 Queue Maintenance 29

3.3.2 Scheduler 30

3.4 Discussion 32

4 Lookahead Algorithm Design **33**

4.1 Overview 33

4.2 A Lookahead-1 Algorithm for Two Machines 36

4.2.1 Theoretical Justification of the LA1 Algorithm 39

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.2.2 LA1 Algorithm Time Complexity 52
4.3 A Lookahead-2 Algorithm for a Single Machine 56
4.3.1 Theoretical Justification of the LA2 Algorithm 61
4.3.2 LA2 Algorithm Time Complexity 86
4.4 Running Time of LA*k* Algorithm 86

5 Algorithm Analysis **88**
5.1 Introduction 88
5.2 Lower Bound for SJF 90
5.3 Lower Bound for LA1 92
5.4 Lower bound for LA*k* 94
5.5 Discussion 96

6 Simulation **99**
6.1 Overview / Introduction 99
6.2 Results 103
6.2.1 Transient Behavior 104
6.2.2 Steady-State Behavior 110
6.3 Discussion 115

7 Conclusion **117**
7.1 Summary of Results and Conclusions 117
7.2 Future Work 119

A Cross-docking Model **122**

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

A.1 Introduction 122
A.2 Relationship to Job Scheduling 123
A.3 Generating Solutions 125
A.3.1 Calculating the Waiting Schedule 125
A.3.2 Non-Idle Machines 129
A.3.3 Calculating the Non-Waiting Schedule 130
A.4 Discussion 132

Bibliography **133**

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ACKNOWLEDGMENTS

First, I would like to thank my advisor and the members of my committee for their insightful comments at important stages of my dissertation. In particular, Weizhen Mao has been both a supportive research advisor and a good friend. She has guided me through graduate school, both academically and socially, by always listening.

In addition, My wife Lauri and one-year-old son, Seth, have been remarkably supportive during this last year of work. Seth's presence is a source of extreme happiness in my life and, in his own special way, he helped me finish this massive project. As a historian, Lauri acknowledges that she does not fully comprehend the technical details of this work. However, she has read the entire document numerous times and given me important comments on grammar, word variety, and transitions. I hope I can be as helpful when she completes her dissertation.

In a similar way, Marcia Zangrilli read every paragraph, proof, footnote, and caption and gave me outstanding advice on how to make it all better. She has also been a wonderful friend, giving me motivation and encouragement as I labored on this document. I am indebted to her.

This whole process started when my undergraduate advisor at Ithaca College, Laurie King, strongly encouraged me to attend her alma mater for graduate school. Throughout my years in Virginia, she was an external source of support with her witty stories of her own graduate school experience.

Finally, I would like to acknowledge Vanessa Godwin who always knows the answers to the important administrative questions about completing a Ph.D., and Stephen Corbesero, whose help distributing the workload of my simulation allowed me to complete my last graph.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

List of Tables

4.1 We ran the simulation for one million scheduling decisions and tabulated all occurring sizes of the set A along with the frequencies. 54

6.1 The mean and standard deviation for each distribution used in the simulations. 102

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

List of Figures

3.1 Our model for lookahead scheduling in a real-time context consists of a lookahead queue and a wait queue plus two processes. One handles queue maintenance and the other make scheduling decisions. 25

4.1 When a decision must be made in the two machine system, the Lookahead-1 algorithm has access to jobs J_1 through J_l in the wait queue plus job J_{l+1} with its arrival time r_{l+1} in the lookahead queue. 36

4.2 In the waiting schedule, machine M_1 remains idle from time t to r_{l+1} and then executes J_{l+1}. After this job completes, the algorithm schedules the jobs in the wait queue in order. 36

4.3 In the non-waiting schedule, the algorithm chooses to schedule J_1 on M_1 immediately. During its execution, J_{l+1} arrives and is added to the wait queue. When J_1 completes execution, the algorithm schedules all of the jobs in the wait queue in order. 36

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.4 In Lemma 4.1, J_{l+1} completes before time t' in the waiting schedule. Depending on the lengths of J_1 and J_{l+1}, there are two possible non-waiting schedules, one where J_{l+1} completes before J_1, and one where it completes after J_1. . . . 43

4.5 In Lemma 4.2, J_{l+1} completes execution after time t' in the waiting schedule. Depending on the lengths of J_1 and J_{l+1}, there are two possible non-waiting schedules, one where J_{l+1} completes before J_1, and the other where it completes after J_1. . . . 45

4.6 In Lemma 4.3, J_{l+1} completes execution before time t' in the waiting schedule. Depending on the lengths of J_1 and J_{l+1}, there are two possible non-waiting schedules, one where both jobs are scheduled on the same machine, and one where they are on different machines. . . . 47

4.7 In Lemma 4.4, multiple jobs complete before time t' in the waiting schedule. Depending on the lengths of J_1 and J_{l+1}, there are two possible non-waiting schedules, one where the jobs are on the same machine as the waiting schedule, and one where a single job is on the other machine. . . . 50

4.8 The set A is defined to be all the jobs that begin execution after time r_{l+1} and complete before time t'. . . . 52

4.9 When a decision must be made in the single machine system, the Lookahead-2 algorithm has access to jobs J_1 through J_l in the wait queue plus jobs J_{l+1} and J_{l+2} with arrival times r_{l+1} and r_{l+2} in the lookahead queue. . . . 55

4.10 If the Lookahead-2 algorithm decides to wait at both decisions, there are two idle periods in the schedule. We call this the wait-wait schedule. . . . 55

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.11 If the Lookahead-2 algorithm decides to wait at time t and then schedule at the second decision, there is a single idle period at the beginning of the schedule. We call this the wait-nowait schedule. 56

4.12 If the Lookahead-2 algorithm decides to schedule at time t and then wait for the second lookahead job, there is a single idle period in the middle of the schedule. We call this the nowait-wait schedule. 56

4.13 If the Lookahead-2 algorithm decides to schedule at both decisions, then there are no idle periods in the schedule. We call this the nowait-nowait schedule. 57

5.1 In Theorem 5.1, the optimal schedule has all of the machines idle until time ϵ and then executes the short jobs followed by the long jobs. In the SJF schedule, the long jobs are executed immediately, and then the short jobs. . 90

5.2 In Theorem 5.2, the optimal schedule has all of the machines idle until time 2ϵ and then schedules the short jobs followed by both sets of long jobs. In the LA1 schedule, the first set of long jobs execute immediately, then the short jobs, and finally, the second set of long jobs. 92

5.3 In Theorem 5.3, the optimal schedule has all of the machines idle until time 2ϵ, then schedules the short jobs followed by all the long jobs. In the LAk schedule, the first set of long jobs is executed immediately, then the short jobs, and finally, the remaining long jobs. 94

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.4 For a fixed m and n, when the LA0 algorithm is used, the competitive ratio is large, implying the worst-case instance is far from optimal. By adding a single lookahead, the ratio is reduced significantly. As the size of the lookahead grows beyond one, the amount of improvement diminishes. 96

6.1 For two identical machines, we measured the average wait time for 25 jobs with service times drawn from various distributions. A single parameter was varied to produce average service times between 3 and 5 time units. Using 100,000 repetitions, each graph shows 95% confidence intervals on the percent improvement of the LA1 algorithm over the SJF algorithm. When the service times were drawn from distributions with small variance, little or no improvement is seen. However, when the variance is large, significant improvement occurs. 105

6.2 Using the traffic intensity and coefficient of variation as unifying metrics, this graph is a composite of the five graphs in Figure 6.1. For a given traffic intensity, as the coefficient of variation increases the variance also increases. This graph shows the correlation between amount of improvement and variance. 106

6.3 In the two identical machine system, we measured the size of the lookahead job and the first job in the wait queue each time the algorithm decided to wait. These graphs show that long jobs were delayed to wait for jobs that were 12 to 16 times shorter. 107

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

6.4 In the two unrelated machine system, we measured the percent improvement for 25 jobs with service times drawn from a Hyperexponential$(1, \mu, \alpha)$ distribution. As μ increases the variance increases, and the LA1 algorithm produces more improvement over the SJF algorithm. As α varies, the ratio of long jobs and short jobs changes. Although the variance increases as α decreases, the percent improvement has the opposite trend. 108

6.5 In the single machine system, we measured the percent improvement of the LA1 algorithm over the SJF algorithm and the LA2 algorithm over the SJF algorithm with service times drawn from Uniform, Exponential, and Hyperexponential distributions. Using 95% confidence intervals, the graphs demonstrate that higher variance corresponds to larger improvement. . . . 109

6.6 For two identical machines, we drew service times from a fixed Hyperexponential distribution and varied the number of jobs. As the number of jobs increases, the improvement of the LA1 algorithm over the SJF algorithm steadily decreases and eventually becomes negative. 110

6.7 In the single machine system, we drew service times from a Hyperexponential distribution and varied the number of jobs. Because the jobs arrive at a constant rate, the schedule has few idle period. Consequently, both the LA1 and LA2 algorithms perform poorly because waiting is rarely beneficial. . . 111

6.8 For two identical machines, we drew service times from a Markov-Modulated Poisson Process that caused jobs to arrive in bursts of approximately 25 jobs. Unlike Figure 6.6, the improvement stabilizes as the number of jobs increase. 112

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

6.9 For two unrelated machines, service times were drawn from a Markov-Modulated Poisson Process that caused jobs to arrive in bursts of approximately 25 jobs. For this system, the improvement is consistent regardless of the parameter values. 113

6.10 For the single machine system, when service times were drawn from a Markov-Modulated Poisson Process, the improvement stabilizes for both that LA1 and LA2 algorithms. 114

A.1 In the cross-docking model, the building is a long thin rectangle with receiving doors on one side and shipping doors on the other. The distance between two doors is one half and the distance across the facility is $\frac{1}{2}d$. Therefore the round-trip distances between doors is one and across the facility is d. 123

A.2 The assignment problem can be viewed as an m by n matrix or as a bipartite graph with m nodes on one side and n on the other. 126

A.3 When an instance of n jobs to be scheduled on m machines is reduced to an instance of the assignment problem, the result can be viewed as an mn by n matrix or a bipartite graph with mn nodes on one side and n nodes on the other. 128

A.4 When a machine is not initially idle, the instance of the assignment problem is augmented by adding the availability time of the machine to each job. . . 131

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ABSTRACT

Our research considers job scheduling, a special type of resource assignment problem. For example, at a cross-docking facility trucks must be assigned to doors where they will be unloaded. The cargo on each truck has various destinations within the facility, and the unloading time for a truck is dependent on the distance from the assigned door to these destinations. The goal is to assign the trucks to doors while minimizing the amount of time to unload all trucks.

We study scheduling algorithms for problems like the cross-docking example that are different from traditional algorithms in two ways. First, we utilize real-time, where the algorithm executes at the same time as when the jobs are handled. Because the time used by the algorithm to make decisions cannot be used to complete a job, these decisions must be made quickly. Second, our algorithms utilize lookahead, or partial knowledge of jobs that will arrive in the future.

The three goals of this research were to demonstrate that lookahead algorithms can be implemented effectively in a real-time context, to measure the amount of improvement gained by utilizing lookahead, and to explore the conditions in which lookahead is beneficial.

We present a model suitable for representing problems that include lookahead in a real-time context. Using this model, we develop lookahead algorithms for two important job scheduling systems and argue that these algorithms make decisions efficiently. We then study the performance of lookahead algorithms using mathematical analysis and simulation.

Our results provide a detailed picture of the behavior of lookahead algorithms in a real-time context. Our analytical study shows that lookahead algorithms produce schedules that are significantly better than those without lookahead. We also found that utilizing Lookahead-1, or knowledge of the next arriving job, produces substantial improvement while requiring the least effort to design. When more lookahead information is used, the solutions are better, but the amount of improvement is not significantly larger than a Lookahead-1 algorithm. Further, algorithms utilizing more lookahead are more complex to design, implement, and analyze. We conclude that Lookahead-1 algorithms are the best balance between improvement and design effort.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Lookahead Scheduling in a Real-time Context:

Models, Algorithms, and Analysis

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 1

Introduction

1.1 Resource Assignment Problems

We study the design and analysis of lookahead algorithms for resource assignment problems. The goal of these problems is to satisfy a sequence of requests for a limited resource. For a typical problem, the number of valid assignments is large, and only a portion of the potential solutions can realistically be considered. The challenge is to develop an efficient heuristic that produces quality solutions with high regularity. Traditionally, online algorithms have been used, however there has been interest recently in semi-online algorithms, specifically lookahead algorithms.

As an example of a resource assignment problem that could benefit from lookahead, consider a national shipping company with a cross-docking terminal. At this facility, trucks arrive carrying cargo destined for multiple locations. When a truck is unloaded, the cargo is sorted according to its destination and loaded onto another truck that will take the cargo toward to its destination. To maximize the number of unloading/loading doors, cross-docking terminals are typically long, thin rectangular buildings. As a result, the distance cargo travels within the building can be quite large, depending on the door

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

assignment of the in-coming and out-going trucks. We define the unloading time for a truck to be the amount of time necessary to move the cargo from the arriving door to the out-going doors. Because this time is determined by the distance between doors, we use the distance and time interchangeably.

In addition to this distance consideration, assigning trucks to unloading doors is a challenging task because there are typically more trucks to unload than unused doors. Thus, we have a resource assignment problem where the unloading doors are a limited resource and the trucks are the requests. This example is further complicated because the trucks arrive throughout the day, changing the available information when making scheduling decisions. Together, the dynamic unloading times and transient arrivals make it very difficult to create an assignment of trucks to doors that minimizes the time to unload the trucks for a given day.

In a real-world cross-docking environment, the assignments are usually performed using a simple algorithm such as First Come, First Serve (FCFS). Whenever a door becomes free, the first available truck is assigned to the door. This method is particularly problematic because the variable unloading times of the trucks are not considered. For example, if an unloading door at the extreme end of the facility becomes available, a waiting truck with cargo destined for that end should be selected rather than the first truck in line. Unlike FCFS, an algorithm such as the Shortest Job First (SJF) heuristic takes this type of optimization into account by considering all waiting trucks. When a door becomes available, the waiting truck with the shortest unloading time is assigned. However, the SJF algorithm can still make sub-optimal decisions because it does not consider trucks arriving in the future. If only a single truck is waiting, the SJF algorithm will assign it, even if a more

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

appropriate truck is about to arrive.

Although the use of future arrivals will improve the quality of the assignment, we must be careful to include this information in a realistic way. It is not possible to predict individual truck arrival times far into the future because the travel times are constrained by many external forces such as traffic. However, it is certainly realistic for a driver to contact the cross-docking facility when he is within a few minutes of arrival. Thus, a lookahead algorithm, using this partial knowledge of the future, can make better assignments than either the FCFS or SJF methods by considering both the dynamic and transient characteristics of the trucks.

The cross-docking example demonstrates a number of the key features of resource assignment problems. Specifically, we have a finite resource (the doors), demand on the resource (trucks to unload), and a goal to minimize some cost (the time to unload the trucks). These features also occur in a number of common problems in Computer Science:

- Paging: There is a system with fast memory of capacity k and slow memory of unlimited capacity. A set of $n > k$ pages is stored in this system with k pages in fast memory and the remaining $n-k$ pages in slow memory. A page request is satisfied if the page is in fast memory, otherwise a page fault occurs and we must remove one of the k pages from fast memory and replace it with the requested page. For a sequence of page requests, the goal is to minimize the number of page faults.

- List Update: Given a list of items in an unsorted list, the input is a sequence of requests. The cost to access any item is its position in the list. When the ith item is accessed, it can be moved to any position $j < i$ at no additional cost. The goal is to

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

minimize the total cost to access the items requested.

- The k-Server Problem: We have a set of k servers which move around a finite set of points in metric space in order to satisfy requests. Each request may only be satisfied if a server is at the location. Given the distances between each pair of points and a sequence of request locations, the goal is to minimize the total distance traveled by all servers.

- Job Scheduling: We are particularly interested in this broad area of problems and use examples within this field to explain our lookahead model and algorithms, such as the cross-docking problem. In a typical job scheduling problem, the goal is to assign n jobs to execute on m machines while minimizing a metric related to the completion times of one or more jobs. The model can be made arbitrarily complex with the addition of other job and machine characteristics. For example, it is common to give each job an arrival time or to specify performance qualities for the machines.

1.2 Algorithms

A solution to any of these resource assignment problems can be seen as a sequence of decisions. Any algorithm that assigns trucks to unloading doors in the cross-docking model must choose a door for each truck. Because there are more trucks than doors, at least one door will be assigned to more than a single truck. Therefore, the algorithm must also specify both a door and the order trucks will be unloaded at that door. It should be apparent that the number of possible assignments grows exponentially in the number of trucks.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

This explosive growth in number of possible solutions relates to each of the resource assignment problems described above, and means it is unrealistic to consider all possible solutions. At the same time, a fundamental result in NP-completeness theory states that an algorithm that does not consider all possibilities cannot guarantee that it always produces the optimal solution. Since we cannot calculate the best assignment, the typical approach is to utilize a clever local condition to make decisions that appear to be favorable. However, for any such heuristic there is at least one instance of the problem where these seemingly good decisions turn out to be bad ones.

Recall the goal in the cross-docking model is to assign trucks to doors in a manner that minimizes the time to unload all trucks. As discussed above, the SJF algorithm is one reasonable heuristic that considers the available trucks and assigns each to the door on which it can be unloaded the fastest. This approach, called a greedy method, can be shown to have very poor performance for certain sequences of trucks.

1.2.1 Online and Offline Algorithms

The SJF algorithm used in the cross-docking example is called an online algorithm because each request was handled without considering any of the future requests. One of the main benefits of online algorithms is that they can be used in situations where the requests become available over time. Although the algorithm makes decisions very quickly, considering each request in isolation does not produce an optimal solution because future trucks are not considered.

At the other extreme, an offline algorithm considers all of the requests when making a decision. One simple example is a method that considers the arrival time and unloading

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

time of each truck at each door and picks the truck and door pair that will unload the soonest. By repeating this process, all trucks are assigned to doors. It should be obvious that this algorithm will produce solutions that are, in general, better than an online technique. However, because this procedure does not consider all possible assignments, we know it cannot be optimal. Note, however, that the optimal algorithm is, by definition, an offline algorithm.

1.2.2 Semi-Online Algorithms

Although we would like to utilize offline algorithms, we believe that realism is a stronger requirement. In most resource assignment problems, it is inconceivable that the algorithm knows about all the requests. However, it is reasonable to assume that partial information is available. Using this information, a semi-online algorithm makes decisions that create better solutions that pure online algorithms.

Semi-online information takes on a wide variety of forms, not constrained to knowledge about individual requests. For example, in the paging problem, if we knew the last time each page would be referenced, a semi-online approach would be to evict each page immediately after its last reference. This algorithm still considers the requests in order, but it uses the additional information to make better decisions.

Lookahead is another type of semi-online approach that augments an online algorithm by considering a finite number of future requests in addition to the current request. In the paging example, when a page fault occurs, lookahead can be used to eliminate pages from consideration for eviction. That is, a page that will be referenced in the near future should not be evicted.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

We believe that lookahead is an important type of semi-online approach, however, it must be considered in a realistic way. Specifically, lookahead is most relevant when time is included in the model.

1.2.3 Real-Time Algorithms

In a real-world problem, requests do not occur all at once, but rather arrive over time. Further, decisions regarding these requests are made while other requests continue to arrive. For example, in the k-server problem, when a request occurs, if one or more servers are available, the algorithm will either pick a server to satisfy the request or hold the request to be satisfied later. This decision is made by considering the status of each server, the queue of other requests waiting to be satisfied, and, if lookahead is available, the requests that will arrive in the future.

Although the decision process is substantially more complex, we believe the inclusion of real-time is fundamentally important in the creation of useful algorithms for resource assignment problems. To study these real-time algorithms, we consider their performance using two metrics. First, as with all approximation algorithms, we want to know how well the algorithm creates solutions. This study is done through theoretical analysis and simulation. Second, it is important to consider the running time of these algorithms. By waiting until a server is available, the algorithm has more information and can make better decisions. However, time used to make decisions cannot be used to satisfy requests, and therefore, the running time of a real-time lookahead algorithm must be minimized.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

1.3 Contributions

Although lookahead has been used to improve solutions to a number of resource assignment problems, very few authors include time in the model, and no one has considered the running time of algorithms. Therefore, we are motivated to study lookahead algorithms that make decisions in a real-time context.

In the cross-docking model, if trucks call in shortly before arriving at the facility, a lookahead algorithm can use this information to create a better assignment. Specifically, as trucks arrive, they are either assigned to a door immediately, or they enter a queue to wait for an available door. Whenever a door becomes available, the algorithm considers the trucks waiting in the queue as well as the trucks that will arrive in the near future. To be consistent with our real-time constraint, these decisions have to be made quickly, without calculating all possible assignments.

1.3.1 Results

We consider the design and analysis of lookahead algorithms for real-time resource assignment problems like the cross-docking example. Our first major contribution to the field is a general model describing how to include lookahead to support the real-time concept and to facilitate fast decision making. This model, discussed in Chapter 3, works for any resource assignment problem where requests arrive over time.

In the remaining chapters, we use our general model and discuss lookahead algorithms for the specific resource assignment problem of job scheduling, of which the cross-docking problem is a special case. In Chapter 4, we develop two lookahead algorithms for im-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

portant scheduling models. The goal of this chapter is to consider whether lookahead algorithms that operate in a real-time context can be designed to make decisions quickly. We show that algorithms that always know about the next single job satisfy this time-constraint. However, for two lookahead jobs and beyond, the running time can be prohibitive to real-time execution.

Another important question regarding lookahead algorithms is the quality of solutions. Because we will not always produce the optimal schedule, the primary question is how far we will be from optimal. In Chapter 5, we consider this question with theoretical study. Borrowing the standard technique called competitive analysis from the study of online algorithms, we measure the worst-case performance of algorithms for a simple parallel machine scheduling model. Chapter 6 continues this study by using simulation to measure the average-case performance of lookahead algorithms. In both chapters, we demonstrate that algorithms with lookahead produce significantly better schedules than algorithms without lookahead.

This work is a collection of many medium-sized results rather than a small number of large results. Although the main chapter headings talk about algorithm design and analysis, we also spend time giving a qualitative measure of our algorithms. When discussing algorithm development in Chapter 4, we give a proof of correctness for our algorithm and then discuss the trade-off between amount of improvement and difficulty of developing the algorithm. In Chapter 6 on our simulation study, we give numerical measurements and then discuss the circumstances in which improvement will occur. By combining these quantitative and qualitative measures, our results give a detailed picture of the behavior of lookahead algorithms in a real-time context.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 2

Related Research

2.1 Motivation

As discussed in Chapter 1, our research builds upon previous work in the areas of online, offline, and semi-online algorithms. Of these three approaches, semi-online algorithms is the most relevant, because we consider lookahead algorithms. Although most of the work in this area has occurred in the last decade, Graham [42] studied one semi-online algorithm as early as 1969. His paper was quite innovative, as it also contained a rudimentary form of competitive analysis, one of the primary techniques used to study online algorithms.

Our research differs from previous work in two ways. First, time plays a central role in our model. That is, the algorithm makes decisions in the context of flowing time, and therefore the moments used to make decisions are lost. We maintain that for these algorithms to be truly useful, decisions must be made in the real-time context. To our knowledge, only Mao and Kincaid [79] consider the running time of the algorithm.

The other way our work is unique is our combination of theoretical and simulation study applied to lookahead algorithms. Most authors consider either theoretical analysis or simulation results, very few consider both. We feel that a combination of worst-case

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

theoretical analysis and average-case evaluation gives a better picture of the performance and behavior of an algorithm.

Our combination of lookahead and real-time is applicable to any resource assignment problem where time is a component. However, we focus our study on job scheduling problems because the field continues to receive significant research attention, and it is a perfect example of a resource assignment problem where lookahead in the real-time context is applicable. In addition, semi-online scheduling, and in particular lookahead scheduling has received only limited attention. Our research seeks to fill this gap while also considering algorithms in a real-time context.

In the remainder of this chapter, we survey the relevant previous work. The next section contains a review of the literature related to traditional online and semi-online algorithms. This is followed by a discussion of competitive analysis, the standard measure of an online algorithm. The chapter concludes with a survey of job scheduling algorithms, with particular attention to lookahead scheduling algorithms and the job scheduling models we consider in our research.

2.2 Online and Semi-Online Algorithms

The study of online algorithms [56, 61] can be traced back to the sixties [42], and since then research in the area has exploded. A representative example is online bin packing [25, 38, 39], for which a number of online algorithms were proposed and analyzed from the early seventies to the early nineties [31, 58, 78, 87, 52].

Since then, there has been a limited but gradually increasing interest in semi-online

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

algorithms. The most common type of semi-online algorithm is a lookahead algorithm [64, 65]. An algorithm utilizes lookahead if it examines one or more of the next requests when deciding how to make the current assignment. In the literature [2, 4], weak lookahead of size k means that the algorithm can foresee the next k requests following the current request being processed. On the other hand, strong lookahead with bound k means that the algorithm can foresee the next l requests, where this value is determined by a pre-selected function f that gives a close approximation to k. Most of the past research results use one of these two definitions of lookahead. For example, Grove [44] and Breslauer [16] studied a weak lookahead algorithm for bin packing which considered the current value as well as all the future items up to a certain cost. Other areas where lookahead has been utilized are graph coloring [54, 59, 66, 13, 14, 53], list update [5, 95, 88], the k-server problem [22], and paging [2, 4, 12, 70, 102, 67, 37, 55, 97, 104]. Note that in each of these cases, time is not a part of the model. The only exception is a study by Aksoy et al. [1] who considered lookahead in a data broadcast model.

Recently, some other variations of the semi-online algorithms have been studied. In situations where each request has a size larger than the next in the sequence, an algorithm is semi-online if it is aware of the fact that the input is sorted [93]. In conventional online and semi-online algorithms, requests must be processed in the order given in the input sequence, regardless of how much is known about the input ahead of time. When this restriction is removed, however, the decision to satisfy each request may be delayed by introducing a fixed-size buffer into the algorithm. When the algorithm is processing a request, it need not make an immediate decision, but instead may store the request in the buffer and proceed to the next [62]. Finally, Azar and Regev [8] studied a variant of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the bin packing problem called bin-stretching. In this problem, items were placed into a fixed number of bins that could stretch to accommodate more items. The authors studied algorithms that already knew the optimal size of the bins.

2.3 Algorithm Analysis

For both online and semi-online algorithms, the primary question is how well the algorithm performs. Mathematical analysis of these algorithms faces a unique problem when attempting to measure this performance. Traditional worst-case analysis leads to the conclusion that no matter how clever the algorithm, the result is always bad. Consider the problem of paging. Regardless of which paging algorithm is applied, there exists a sequence of page references such that a page fault occurs for every reference in the sequence. With such a measure, it is impossible to compare various algorithms. Motivated by this limitation, Sleator and Tarjan [96] developed a method to compare online algorithms. Competitive analysis [15] (named by Karlin et al. [60] and formalized by Manasse et al. [77]), compares the worst case performance of an algorithm with the performance of an optimal offline algorithm for the *same* sequence.

Formally, let A be an algorithm for a given problem. Further, let I be an instance for the problem and let $A(I)$ and $\mathrm{OPT}(I)$ be the values of the objective function given by the algorithm and the optimal algorithm, respectively. Then the algorithm A is said to be c-competitive if for all I,

$$A(I) \leq c \cdot \mathrm{OPT}(I) + a$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

When $a = 0$, the algorithm is said to be *strictly* c-competitive.

Another way to think about competitive analysis is as a game between two players. Player one is the algorithm. Player two is an adversary who produces the requests given to the algorithm. The adversary's goal is to create the highest cost (value of the objective function) possible and it has the benefit of knowing the method used by the algorithm to make decisions. Therefore, for each request, the adversary can determine the answer that will produce the highest cost.

When giving a proof of a competitive ratio, the typical approach is to give a proof of a lower bound and a proof of an upper bound. When these two values are equal, we have a tight ratio. As Karp discussed [61], lower bound proofs are frequently given with an adversary argument while proving upper bounds requires more advanced analysis. In general, it is easier to find a lower bound since all that is needed is an example that produces this bound.

As an alternative to competitive analysis, a number of sources discuss using randomized algorithms [84, 70, 103], where decisions are made based on a coin flip or some other chance event. In our research, we focus only on deterministic algorithms.

2.4 Job Scheduling

In the two previous sections, we surveyed general results related to online and semi-online algorithms. We now turn to our specific area of interest, job scheduling, one of the largest sources of resource assignment problems [43, 71, 34, 18, 85]. To define these problems, we use a three field classification $\alpha|\beta|\gamma$ where α specifies the machine environment, β denotes

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the job characteristics, and γ specifies the optimality criteria. This classification scheme was originally proposed by Graham et al. [43] and later extended to include provisions for multimachine jobs [99, 34].

Generally speaking, we have a set of n jobs, $J_1 \ldots J_n$, to schedule on m machines, $M_1 \ldots M_m$, and the goal is to minimize a time related criteria. Each job J_j has an associated processing time p_j and once a job is scheduled, we know its completion time C_j. The most general problem of interest is $P|r_j|\sum C_j$, where P means that we have m machines of the same speed, r_j designates that jobs have arrival times, or the earliest times the jobs may be scheduled, and $\sum C_j$ means we are trying to minimize the sum of all of the completion times. This example of the classification notation introduces some important types of α, β, and γ parameters. Other notation will be explained as necessary.

Although scheduling results exist for a variety of models [43, 71, 28, 72, 17, 24, 34, 46, 36, 18], our interests are in online and semi-online algorithms. The literature of scheduling algorithms parallels that of general online algorithms. Online job scheduling stems from the seminal paper by Graham [42], who studied the problem $P||C_{\max}$ and developed what he called list scheduling. For this problem, the goal is to minimize the maximum completion time, and since the jobs do not have an arrival time, Graham determined to create a list of the jobs and schedule them in order. Note that this is the first example of an online algorithm for job scheduling [94, 80]. The work of Graham is also the first study of semi-online algorithms. Given that the jobs are in a list, a natural step is to sort the list. Graham proposed to sort the list by processing times with $p_i \geq p_{i+1}$ for all $i \geq 1$. He showed an algorithm with a $\left(\frac{4}{3} - \frac{1}{3m}\right)$ competitive ratio.

Other semi-online scheduling algorithms include those of Seiden et al. [93] who looked

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

at a number of job scheduling problems for two and three identical machines. By assuming the jobs were sorted in decreasing size, the authors produced a number of tight competitive ratios for minimizing $C_{\max}$. Similarly, Zhang and He [49] studied $P2||C_{\max}$ for two different assumptions about the jobs. First, they gave a tight competitive ratio for the case when processing times were between p and rp ($p > 0, r \leq 1$). Second, they assumed the largest processing time was known. In this case, they showed that this additional knowledge does not change the ratio originally given by Graham [42].

Liu et al. [76] studied $P2||C_{\max}$ and $P3||C_{\max}$ under the assumption that processing times are not known until after the job is assigned to a machine. However, it is known that processing times decrease throughout the sequence of jobs. They showed that the competitive ratio is $\frac{4}{3}$ for 2 machines, and $\frac{7}{5}$ for 3 machines.

Another form of semi-online algorithm was given by Zhang [106] for the problem $P2||C_{\max}$. The algorithm processed the jobs in order, but instead of being forced to schedule each job on one of the machines, it could place the jobs in a fixed size buffer. In this way decisions could be delayed. Kellerer et al. [62] also considered this model and additionally studied an algorithm that produced two schedules and chose the better.

Note that for each of these studies, jobs do not have arrival times. Therefore, the problems represent the traditional framework where all requests are available at once rather than the real-time framework we advocate.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.5 Lookahead Scheduling

Within the larger field of job scheduling, we focus on lookahead algorithms in a real-time context. In the literature, a number of results use the term lookahead, however, these definitions differ from ours. For example, Sarkar et al. [89] studied lookahead as applied to a number of resource assignment problems. For these problems, the authors conceptualized a tree representing the possible decisions for an algorithm. In this context, lookahead was defined as traversing the tree a certain number of levels.

Motwani et al. [81] studied a scheduling problem that closely represented a copy system. Each job required multiple passes through the machines, where each pass needed the same processing time. The goal was to minimize the makespan, but all jobs in the schedule had to start and complete in the order they arrived. An offline algorithm for this problem could produce the optimal schedule and the authors showed that by using finite lookahead, a solution close to optimal could be created.

A number of authors have applied lookahead to alternative scheduling models. Gue [45] studied the cross-docking model discussed in Chapter 1 where jobs represented freight trucks. These trucks had to be assigned to unloading bays and the cargo on the trucks moved to out-going trucks. The bay assignment of the incoming truck determined the distance the cargo had to move to the out-going trucks. The goal was to minimize the distance the cargo moved. Gue defined lookahead to be examining the contents of each truck before assigning it to a bay.

Beaty [10, 11] studied a variant of job scheduling where it is possible to create non-feasible schedules. In this case, lookahead was used to increase the chance a feasible

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

schedule was produced. Yu et al. [105] studied a lookahead algorithm for pause-resume video-on-demand.

Many alternative models have been studied through simulation rather than analysis. For example, Winckler [101] studied a distributed algorithm for scheduling using lookahead, Koulamas and Smith [69] examined lookahead applied to a variant of a shop model, and Schaerf [90] and Cristofari et al. [30] studied various scheduling models using AI techniques with lookahead to create schedules.

In contrast to all these definitions of lookahead, in our research, jobs arrive over time. Therefore, lookahead is defined as having knowledge of a job before it arrives. Very few previous results have used this definition of lookahead. Li et al. [75] and Li et al. [74] studied lookahead algorithms for two different parallel machine models with communication costs through simulation. The work of Mao and Kincaid [79] is the only scheduling result using lookahead in a real-time context that gives an analytical study. For the problem $1|r_j|\sum C_j$, the authors analyzed an algorithm with knowledge of the next arriving job and presented a $\frac{1}{3}(n+1)$ competitive ratio. Our research expands on this work by including simulation and considering additional scheduling models.

2.6 Scheduling Models

In this last section, we survey the results for the specific models used in our research.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.6.1 Single Machine Scheduling

The problem $1|r_j|\sum C_j$ is one of the fundamental scheduling problems. Jobs arrive over time to be scheduled on a single machine. The goal is to minimize the total completion time. This is equivalent to minimizing the average wait time since $C_j = r_j + d_j + p_j$ where d_j is the delay for the job, or the time between its arrival and when it begins execution. Note that for any job, r_j and p_j are constant.

The offline version of this problem is known to be NP-hard [72] and many approximation algorithms have been proposed. Dessoukey and Deogun [33] gave a branch-and-bound algorithm, Deogun [32] showed a partitioning scheme, Chand et al. [20] offered an improved branch-and-bound method, Gazmuri [40] did a probabilistic analysis, and Ponser [86] showed that a greedy method gives optimal solutions under certain conditions. Chu [23] demonstrated algorithms making local optimal conditions. Finally, Mao et al. [80] analyzed the SJF and FCFS algorithms and showed that both have a competitive ratio of n.

The offline preemptive version of the problem can be solved in polynomial time [9]. Using this schedule, Kellerer et al. [63] showed how to construct a non-preemptive schedule, sometimes called a relaxation. The authors demonstrated this method has a competitive ration of $\mathcal{O}(\sqrt{n})$ ratio. Other relaxation methods were proposed by Chekuri et al. [21] and Hall et al. [48, 47].

In all preceding cases, the competitive ratio was a function of n. Hoogeveen and Vestjens [51] presented the first algorithm with a constant competitive ratio. With this algorithm, jobs are held until they could have finished and then they are scheduled. By de-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

laying jobs in this way, short jobs can be scheduled before long jobs to create a better schedule. The authors demonstrated this algorithm has a competitive ratio of 2. Anderson and Potts [6] extended this idea to weighted completion times while still retaining the ratio of 2. Goemans [41] used a similar waiting technique, however his algorithm only produced a 2.415 competitive ratio. van Stee and La Poutre [98] used the waiting technique while allowing restarts. Finally, Epstein and van Stee [35] discussed lower-bounds for the problem.

Although these recent results give a constant competitive ratio, the average case of the algorithm is significantly worse than the simple Shortest Job First rule. For any sequence of jobs, these algorithms produce approximately twice the amount of wait as the optimal because every job is delayed. As we discuss in Chapter 6, waiting is only beneficial when long jobs are delayed to wait for short jobs. These algorithms delay all jobs, causing unnecessary wait.

2.6.2 Identical Parallel Machine Scheduling

The second model we study is a generalization of $1|r_j|\sum C_j$ where we have m identical machines on which to schedule the jobs. This problem is denoted $P|r_j|\sum C_j$ and has been studied extensively both online and offline [71, 46] beginning with Bruno et al. [19]. More recently, Leonardi and Raz [73] showed that no approximation algorithm can achieve a competitive ratio better than $\mathcal{O}(n^{1/3-\epsilon})$. For this reason, much interest has been given to preemptive scheduling [83, 7, 3], although, as noted above, a number of semi-online algorithms have been considered recently [106, 49, 76, 93, 42, 62].

In general, research on this problem has considered variations such as communication

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

costs [94, 36]. In part, this is because the problem has so many real-world uses. We do note, however, that the Shortest Job First algorithm has not been studied either analytically or through simulation in the literature.

2.6.3 Unrelated Machine Scheduling

For the final model we study, we have the most general parallel scheduling problem. In this case, $R|r_j|\sum C_j$, each of the m machines P_i has a potentially different processing time p_{ij} for each job J_j. This means that each machine can execute any of the jobs, but the time required can vary. The cross-docking example discussed in Chapter 1 and described by Gue [45] is an excellent real-world example of this model. Appendix A discusses cross-docking in more detail.

For the general model, Phillips et al. [82] gave the first approximation algorithm with a competitive ratio of $\mathcal{O}(\log^2 n)$. Hall et al. [48] gave the first constant ratio of 16/3, and this was improved upon by Schulz and Skutella using randomization [92, 91]. Vredeveld and Hurkens [100] offered empirical studies of a number of these algorithms. In each case, the algorithms run offline and have super-linear running time. Jansen and Porkolab [57] developed a linear-time algorithm when m is fixed. Hoogeveen et al. [50] showed that no algorithm can have a competitive ratio arbitrarily close to 1.

2.7 Discussion

Although many of the concepts involved have been used previously, our research represents a new, more realistic approach to lookahead job scheduling. Our model is centered

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

around the real-time concept, and as a result, our definition of lookahead is more natural. In addition, our use of both theoretical, worst-case analysis and simulation-based average-case evaluation gives a more complete understanding of the behavior of our algorithms.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 3

A Model for Lookahead Algorithms

3.1 Introduction

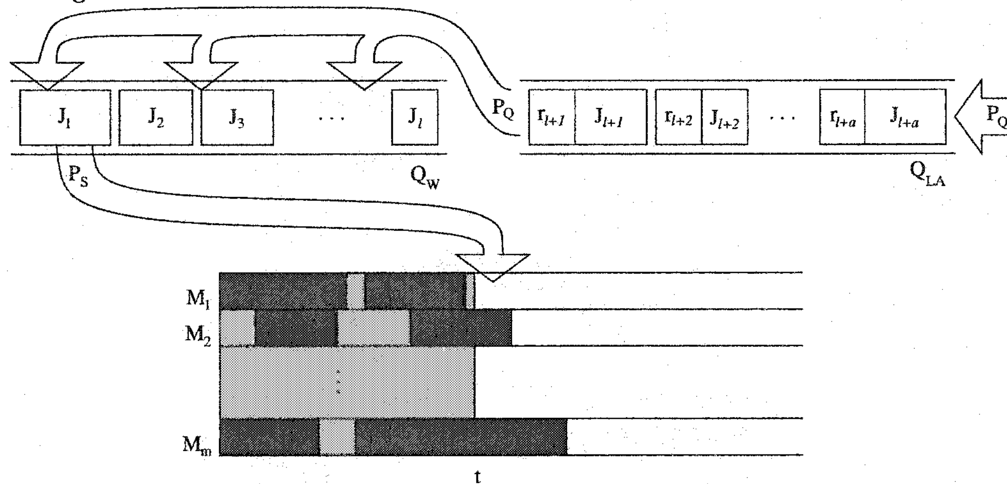
Model building is an important part of algorithm development. The model places an algorithm in the appropriate context by describing the under-lying assumptions. In this chapter, we present our first major result, a model suitable for any resource assignment problem to be solved by a lookahead algorithm in a real-time context.

The utilization of real-time in a lookahead algorithm causes our model to be more complex than those used with a traditional algorithm. For an offline algorithm that has access to all of the requests, the model is simply a data structure holding the requests that allows random access. Likewise, with an algorithm that considers each request in order, we simply have a queue of requests. However, with a lookahead algorithm in a real-time context, the requests arrive over time and the algorithm considers a combination of unsatisfied requests and future requests.

In this context, we are interested in job scheduling, a representative resource assignment problem. We will define our model in terms of a general job scheduling problem where we have n jobs to schedule on m machines. Associated with each job J_j, is a pro-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 3.1: Our model for lookahead scheduling in a real-time context consists of a lookahead queue and a wait queue plus two processes. One handles queue maintenance and the other make scheduling decisions.



cessing time p_j and an arrival time r_j, the earliest time the job may execute. The goal is to create a schedule that executes all n jobs that minimizes a time-related criteria such as the maximum completion time or total completion time. Although we have selected job scheduling to describe the lookahead model, it is applicable for any resource assignment problem in the real-time context.

In Figure 3.1, we have a snapshot for an instance of a job scheduling problem. In the schedule at the bottom of the figure, t represents the current moment in time, with the light gray area representing the past. Scheduling decisions have been made such that some machines are currently busy, designated by dark gray jobs. However, since we know the processing time of each job, we know when that machine will be available again.

Above the schedule are the two queues. An important consequence of making decisions in real-time is that requests arrive over time and might not be satisfied immediately. In this case, the request is placed in the wait queue, Q_W, and processed later. In the figure,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

we have a set of jobs J_1 through J_l that have arrived, but have not been scheduled. The order of the wait queue is an important property and is discussed in Section 3.2.

The second queue maintains information about lookahead. Because the requests have not formally occurred, they cannot be assigned. However, knowledge of these requests allows the algorithm to make a more informed decision. In the figure, Q_{LA} contains the set of jobs that have not arrived but have been predicted. Along with each of these jobs is its arrival time, r_j. The question of how jobs are predicted is discussed in section 3.2.2.

The existence of these two queues creates additional administrative concerns. For example, requests must be moved from the lookahead queue to the waiting queue at the appropriate time. Although these are important activities, they are, for the most part, disjoint from the task of creating a good assignment. For that reason, we have separated these tasks into a queue maintenance process. The details of this separation are discussed in Section 3.3.1.

The final portion of the model, and arguably the most important, is the algorithm that makes decisions about when to satisfy each request. Because we consider job scheduling, the algorithm uses the information available in both queues to create an assignment of jobs to machines. In the figure, this is represented by the movement of the first job in the wait queue to the schedule. In section 3.3.2, we discuss why the algorithm always selects the first job and other relevant design considerations.

Together, the two queues and the two processes describe a complete model for representing resource assignment problems to be solved by lookahead algorithms in a real-time context. In the remainder of this chapter, we discuss each component of the model in more detail, using the specific resource assignment problem of job scheduling as a motivating

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

example.

3.2 Queue Design

The wait queue, Q_W, and the lookahead queue, Q_{LA}, hold jobs before they are scheduled. The most important design issue concerning the wait queue is the order of the jobs. To build the lookahead queue, we must decide how future jobs are discovered.

3.2.1 Wait Queue

In Figure 3.1, Q_W is the wait queue that holds jobs that have arrived but have not been scheduled. By sorting the queue in certain ways, we can guarantee that the scheduling algorithm always selects the first job in the queue when it decides to schedule a job. This results in a scheduling algorithm with faster running time.

If the goal is to minimize the average wait time over all jobs, Conway [28] proved that for a given set of jobs that have already arrived, the smallest average wait time is produced by scheduling the jobs using the Shortest Job First rule. Therefore, the waiting queue will be sorted by processing time with the shortest at the head if the goal is to minimize the average wait time.

Conversely, if the goal is to minimize the total time necessary to complete all jobs, we will sort the wait queue so that the largest job is at the head. Graham [42] showed that this will be optimal for a given set of jobs.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.2.2 Lookahead Queue

As Figure 3.1 depicts, jobs enter the wait queue by first passing through the lookahead queue, Q_{LA}. This auxiliary queue contains jobs J_{l+1} through J_{l+a} that will arrive in the future. These jobs cannot be scheduled because they have not arrived. Therefore, it makes sense to sort this queue by arrival time. When a job is predicted, it is added to the end of the queue, and by the time its arrival time occurs, it will have moved to the head.

The prediction of jobs must be done in a realistic way. For example, in the cross-docking system introduced in Chapter 1, it is reasonable that trucks can use radio communication to announce their arrival time when they are within a short distance of the cross-docking warehouse. Likewise, a facility that uses a combination of an appointment book and "walk-in," or spontaneous sessions, can also be viewed as utilizing lookahead. In this case, the appointments are predicted before they occur and the others are never predicted.

We consider two mathematical methods for predicting jobs. In the first approach, the lookahead queue always contains the next k incoming jobs for some constant $k \geq 1$. Let t be the current time and let $J_{l+1}, \ldots, J_n$ be the jobs which have not arrived. Further, assume that $r_{l+1} \leq r_{l+2} \leq \ldots \leq r_n$. In this case, the jobs $J_{l+1} \ldots J_{l+k}$ form the lookahead queue. For any sequence of n jobs, Q_{LA} always holds k jobs until job J_{n-k} arrives. At this point, there are no additional jobs to add to Q_{LA} and the queue size will decrease until the last job arrives.

Another method for creating the lookahead queue is to lookahead Δ time units, where Δ is a pre-selected constant similar to k. Let the current time be t. Then the lookahead

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

queue will contain all the jobs arriving between time t and time $t + \Delta$. For a sequence of n jobs, the number of jobs in the lookahead queue can be as large as n when all jobs have arrival times less than $t+\Delta$, or it can be as small as zero in the case that the earliest arriving future job has an arrival time greater than $t + \Delta$.

No matter which of the two methods is used to define the lookahead queue, the choice for the constants k or Δ determines the amount of future information available to the scheduler.

3.3 Control Processes

Our discussion of the design for the wait and lookahead queues talked about the maintenance that must occur for each queue. The other important aspect of our model is the two processes that perform this work. We have a process P_Q that handles the tasks related to queue maintenance, and a second process, P_S, that makes scheduling decisions. This is a natural separation because the tasks are essentially disjoint. As discussed in section 3.2, by sorting the wait queue in a way appropriate to the goal, the scheduler always chooses the first job in the queue. Thus, the scheduler only needs read access to the queues, and it directs the queue maintenance process to remove the job when it makes a scheduling decision.

3.3.1 Queue Maintenance

The task of moving jobs in and out of the two queues and keeping the queues in the correct order is the task of process P_Q, represented in Figure 3.1 as arrows into Q_{LA} and between

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Q_{LA} and Q_W. Specifically, P_Q moves jobs out of Q_W for execution, moves jobs out of Q_{LA} and adds them to Q_W when the arrival times of the jobs occur, and adds jobs to Q_{LA} when this information becomes available. Process P_Q is also responsible for keeping the sorted orders in the queues. Typically, the lookahead queue is ordered by increasing arrival times and the wait queue may be ordered by processing times, arrival times, or priorities of the jobs in the queue, depending on the algorithm adopted by the scheduler.

3.3.2 Scheduler

The scheduler is called each time a scheduling decision must be made. This occurs in two situations, when a job arrives and at least one machine is idle, and when a job completes execution and the wait queue is not empty. As a result, the decision-making is partitioned into multiple executions of the same algorithm. The purpose of this algorithm is to choose whether to schedule a particular job or to wait for the next job to arrive, based on the information available in Q_W and Q_{LA}.

Making these decisions effectively is a very difficult task. From a mathematical perspective, the problem is NP-hard regardless of the optimality criteria because the lookahead jobs have arrival times [72]. We are further challenged by the fact that the algorithm only knows about a portion of the jobs that will arrive in the future. Together, these factors mean that any real-time algorithm will only produce an approximation of the optimal schedule.

A good approximation is a greedy approach because the goal is to minimize the objective function over the entire schedule. This type of algorithm makes locally optimal decisions with the idea that the cumulative schedule will be close to the true optimal. In

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the case of our job scheduling model, we define the local optimal in terms of the known jobs in Q_W and Q_{LA}. We assume that no further jobs will arrive. Under this assumption, our algorithm has the same information as an offline algorithm that knows the entire sequence of jobs.

Using this information, the algorithm must decide whether the local-optimal schedule is produced by executing a job in the wait queue or waiting for a job in the lookahead queue to arrive. Discovering a technique to make this decision was a major goal of our research.

A naive but straight-forward method is to generate all possible schedules of the jobs in Q_W and Q_{LA}, evaluate the objective function for each schedule, and then identify the one with the minimum value. The scheduler then makes its decision consistent with this local-optimal schedule. The scheduler chooses to execute a job from Q_W at time t if the best schedule executes and it decides to wait if the best schedule waits.

Unfortunately, this brute-force method is not realistic. The machine remains idle, waiting for the algorithm to decide whether to wait or schedule, resulting in exponential running time. We are therefore motivated to develop an algorithm smarter than the naive method which makes optimal local decisions in linear, logarithmic or constant time. Our ultimate goal is to achieve constant time by developing a local condition containing predicates derived from the characteristics of the jobs in Q_W and Q_{LA}. The satisfaction of the local condition implies that waiting at t is always better than executing a job in Q_W or vice versa. If such a local condition exists, the algorithm can make the optimal local decision without calculating all possible schedules.

For some job scheduling problems, the sub-problem of deciding whether to wait or

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

schedule may be NP-hard. Because it will be impossible to find the optimal local solution quickly, we will be forced to develop an approximation algorithm for the local optimal decisions. In either case, the technique used to decide whether to wait or schedule is our primary interest.

3.4 Discussion

The lookahead model we have described is a new, more realistic approach to job scheduling because scheduling decisions are made in the context of time. By holding unscheduled jobs in a wait queue, we can alter the order of assignment and consequently create a better schedule. In addition, lookahead is a middle ground between the all-knowing offline algorithm and the short-sighted online algorithm.

Using this model, we consider two fundamental research questions. First, how can we use the information in Q_W and Q_{LA} to create better schedules? We use a greedy method to make the decision that appears to be the best at each given time instant. Second, can we implement this decision process in constant time? The execution time of the algorithm is an important consideration because the amount of time used to make scheduling decisions affects the execution of the jobs. We consider both of these questions in the following chapters.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 4

Lookahead Algorithm Design

4.1 Overview

In the previous chapter, we presented a general model for utilizing lookahead algorithms to solve resource assignment problems in a real-time context. In this chapter, we focus on the scheduling process in the lookahead model. Specifically, we demonstrate how to implement a lookahead algorithm efficiently for two fundamental job scheduling problems.

For both of the scheduling problems, there are n jobs, and associated with each job, J_j, is a processing time, p_j, and arrival time, r_j. We assume that, through the lookahead queue, we always know the arrival time and processing time of one or more of the next arriving jobs, called the lookahead jobs. For both problems, the goal is to minimize the sum of completion times.

As discussed in the previous chapter, it is not possible to create the optimal schedule because a lookahead algorithm used to make each decision does not have knowledge of all of the jobs. Instead, we use a greedy heuristic to incrementally create a schedule that is an approximation of the optimal. When a job arrives and a machine is idle, the algorithm executes and decides whether to schedule that job or to hold it until another job arrives.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Likewise, when a job completes and there are jobs waiting, the algorithm must decide whether to execute the first job in the wait queue or wait.

To construct a complete schedule for n jobs, the decision whether to wait or schedule is made multiple times. It is important to point out that the algorithms we present execute once for each of these decision. This is a significant departure from a traditional algorithm that executes outside of time and creates the entire schedule in one execution. Our algorithm executes in real-time and must complete execution before any job may begin. Therefore, each decision must be made quickly.

For a conservative algorithm that never holds a machine idle when there is a job waiting, the execution time of the algorithm is typically not long. For example, the Shortest Job First (SJF) algorithm always executes the job with the smallest processing time and never decides to wait. Therefore, scheduling decisions are made in constant time, assuming the wait queue is sorted appropriately. In the offline version of the problem where the jobs do not have arrival times, this method produces the optimal schedule [28].

Although the SJF algorithm works well when all jobs are available at once, it can easily be fooled into making bad decisions when the jobs arrive over time. In a one machine environment, if the machine becomes idle and a very long job is the only one in the wait queue, the SJF algorithm executes this long job. However, if a short job is about to arrive, a better schedule is to wait for the short job to arrive, schedule it, and then schedule the long job. In this example, the SJF algorithm's lack of foresight causes it to make a poor scheduling decision.

A lookahead algorithm, by comparison, creates a better schedule because it has knowledge of jobs that arrive in the near future. This complicates the decision process because

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the scheduler must determine whether it is more beneficial to wait for a job or to schedule a job in the wait queue. For our model, this decision process is broken down into three steps. First, the algorithm assumes that no jobs will arrive other than those already in the lookahead queue. With this assumption, the problem is reduced to an offline problem because the scheduler has knowledge of all jobs. Second, the algorithm calculates the objective function for "waiting" and "non-waiting" schedules. The former is the optimal schedule of all unscheduled jobs produced after a job is scheduled immediately and the latter is the optimal schedule produced after waiting for the first lookahead job to arrive. In the final step of the process, our algorithm determines which schedule produces the smaller objective value. If the waiting schedule is better, the algorithm decides to wait, otherwise the algorithm schedules the smallest job in the queue[1].

For this process, the second step requires the most explanation. Given a set of jobs, there are an exponential number of feasible schedules, even after the first decision to wait or schedule is specified. Because the algorithm executes in real-time, it is not reasonable to calculate all these schedules for both the waiting and non-waiting cases. Instead, we present conditions that can be evaluated quickly by inferring the optimal waiting and not-waiting schedules, usually without considering each of the jobs in the two queues.

In the next section, we discuss a Lookahead-1 algorithm for a two machine system and the running time of this algorithm. In the following section, we develop a Lookahead-2 algorithm for the single machine environment. Finally, we discuss the exponential running time of a Lookahead-k algorithm.

[1]Conway's result can easily be extended to show that the shortest job will be executed even though some of the jobs have arrival times.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 4.1: When a decision must be made in the two machine system, the Lookahead-1 algorithm has access to jobs J_1 through J_l in the wait queue plus job J_{l+1} with its arrival time r_{l+1} in the lookahead queue.

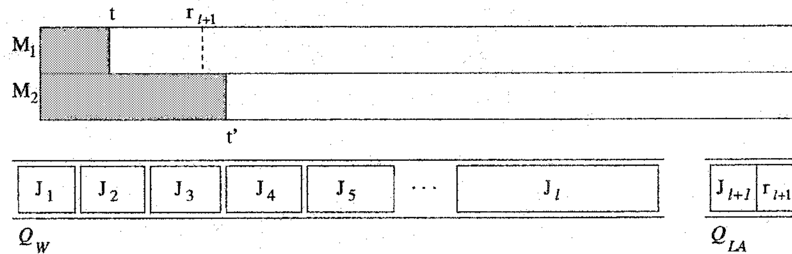


Figure 4.2: In the waiting schedule, machine M_1 remains idle from time t to r_{l+1} and then executes J_{l+1}. After this job completes, the algorithm schedules the jobs in the wait queue in order.

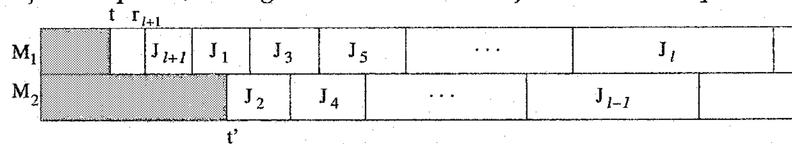
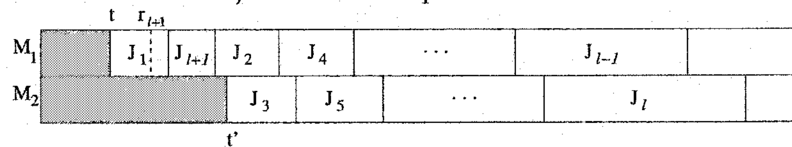


Figure 4.3: In the non-waiting schedule, the algorithm chooses to schedule J_1 on M_1 immediately. During its execution, J_{l+1} arrives and is added to the wait queue. When J_1 completes execution, the algorithm schedules all of the jobs in the wait queue in order.



4.2 A Lookahead-1 Algorithm for Two Machines

In the job scheduling problem $P2|r_j|\sum C_j$, we have two machines, M_1 and M_2 on which we want to schedule n jobs, each of which has an associated arrival time r_j, and a processing time p_j. The goal is to minimize $\sum C_j$, the sum of the completion times for all n jobs, also called the total completion time.

In addition to the queue of waiting jobs, our algorithm always has access to the arrival time and processing requirements of the next job to arrive. In Section 4.2.1, we describe a method to decide between waiting and scheduling that typically does not consider every job in the wait queue.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 4.1 shows a representative instance of the problem. At time t, machine M_1 is idle and M_2 will become idle at time t'. We have a set of jobs J_1 through J_l in the wait queue, and a job J_{l+1} in the lookahead queue that will arrive at time $r_{l+1} > t$. The task of the algorithm is to decide whether to execute job J_1 immediately or to wait for job J_{l+1} to arrive.

To understand how the algorithm works, consider Figures 4.2 and 4.3 that show the schedules produced as a result of each of the possible decisions. Comparing these two schedules, we notice that the even indexed jobs are scheduled in order on one machine and the odd indexed jobs (with the exception of J_1) are scheduled in order on the other machine. This observation is the key to our algorithm, and allows it to make decisions without calculating either schedule entirely.

This helpful pattern occurs because the jobs are scheduled in order from shortest to longest. In the non-waiting schedule in Figure 4.3, job J_2 starts the alternating sequence because it is the job that begins execution before time t' and completes after. Because $p_3 > p_2$, when J_3 is scheduled on M_2, its completion time is necessarily after the completion time of J_2. Likewise, J_4 is scheduled on M_1 and completes after J_3 and, because each job is longer than the previous, the alternating pattern continues through job J_l.

This pattern occurs in both the waiting and non-waiting schedules, although it may be that the groups are on opposite machines in each schedule. It is also possible for a single job at the beginning of the sequence to remain on the same machine in both schedules. Regardless, the details of the alternation can be determined without calculating the entire schedule, and using this information, we can decide which schedule creates a smaller total completion time. Consequently, our algorithm can quickly determine whether to wait or

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

schedule, an important consideration in a real-time context.

Throughout the algorithm presented below, we use t to refer to the current time and, without loss of generality, refer to the idle machine as M_1. The time when machine M_2 becomes available is t'. In the pseudo-code, "schedule" means that J_1 is executed on M_1 starting at time t. Accordingly, "wait" means that M_1 is idle from t to r_{l+1}. Further definition will be given in the next section.

Procedure Lookahead

  if Q_W is empty

    then return

  if Q_{LA} is empty

    then schedule

  else if one of the following conditions is true

    (1) $t \geq t'$

    (2) $t < t'$ and $p_1 \leq \Delta$

    (3) $t < t', p_1 > \Delta$, and $r_{l+1} \geq t'$

    (4) $t < t', p_1 > \Delta, r_{l+1} < t'$, and $p_{l+1} \geq p_1$

    (5) $t < t', p_1 > \Delta, r_{l+1} < t', p_{l+1} < p_1$, and $\Delta + p_{l+1} \geq p_1$

    then schedule

  else if $t + p_1 > t'$

    then if $t + \Delta + p_{l+1} < t'$ //case $\boxed{1}$

      then if $(1 + |B|)\Delta + (|B| - |C| - 1)\min\{t' - t, p_1 - p_{l+1}\}$

        $+(|B| - |C|)(t + p_{l+1} - t') \geq 0$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

then schedule

else wait

else if $t + \Delta + p_{l+1} \geq t'$ //case $\boxed{2}$

then if $|B|\Delta + (|C| - |B|) \min\{t' - t, p_1 - p_{l+1}\} \geq 0$

then schedule

else wait

else if $t + p_1 \leq t'$

then if $t + \Delta + p_{l+1} + p_1 \geq t'$ //case $\boxed{3}$

then if $(1 + |B|)\Delta + p_{l+1} - p_1 + (|C| - |B| + 1) \max\{0, t' - t - p_{l+1} - p_b\} \geq 0$

then schedule

else wait

else if $t + \Delta + p_{l+1} + p_1 < t'$ //case $\boxed{4}$

then if $(|A| + |B|)\Delta + p_{l+1} - p_1 + (|C| - |B| + 1) \max\{0, t' - t - \sum_A p_j - p_b\} \geq 0$

then schedule

else wait

4.2.1 Theoretical Justification of the LA1 Algorithm

The goal of this section is to demonstrate that the various conditions presented in the Lookahead-1 algorithm are both necessary and sufficient when making scheduling decisions. That is, we want to show that the algorithm can successfully determine to wait or schedule without considering every possible schedule. To do so, we break the overall

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

structure of the algorithm into three parts. The first two conditions address the algorithm's decision when one of the queues is empty. The next condition, broken into five cases, specifies decisions we call "obvious." In the last four conditions, the algorithm must analyze the alternating pattern to make its decision. In this section, we consider these three possibilities in turn.

Our algorithm is executed when a machine becomes idle or when a job arrives and there is already an idle machine. Obviously, if Q_W is empty, then we must wait and if Q_{LA} is empty, then we should schedule. The first two conditions of our algorithm represent these actions.

Because there is no work to do when either queue is empty, from now on, we make the following assumptions, without loss of generality, about the state of the queues:

- The wait queue Q_W contains l jobs, $J_1, \dots, J_l$, all of which have arrived but have not been scheduled. These jobs are ordered by non-decreasing processing times: $p_1 \leq \cdots \leq p_l$.
- The lookahead queue Q_{LA} contains the next in-coming job J_{l+1} with processing time p_{l+1} and arrival time $r_{l+1} > t$.

Using this notation, we next consider some cases in which the choice to schedule J_1 is obvious. Without loss of generality, assume M_1 is idle at time t and we need to make a scheduling decision. Recall that this can occur either because a job completes execution or because a job arrives. Also, let t' be the time when M_2 becomes idle and let $\Delta = r_{l+1} - t > 0$ be the amount of time before the lookahead job arrives, called the waiting period.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- $t \geq t'$: When both machines are idle, it does not make sense to make both wait. The lookahead algorithm executes J_1.

- $t < t'$ and $p_1 \leq \Delta$: When the processing time of J_1 can fit into the waiting period without increasing the completion times of other jobs, scheduling J_1 on M_1 is an obvious choice.

- $t < t', p_1 > \Delta$, and $r_{l+1} \geq t'$: In this case, J_{l+1} arrives after t'. If M_1 is kept idle, then at time t' both machines are idle and we have the first case. Therefore, J_1 is executed at t to yield a smaller completion time.

- $t < t', p_1 > \Delta, r_{l+1} < t'$, and $p_{l+1} \geq p_1$: Because $p_{l+1} \geq p_1$, if the algorithm waits, then when J_{l+1} arrives, it is placed in the queue somewhere behind J_1. Therefore, the jobs are scheduled in the same order regardless of whether or not the algorithm waits. It follows that starting the sequence of jobs at time t rather than r_{l+1} produces a schedule with a smaller total completion time. Therefore, the algorithm executes J_1.

- $t < t', p_1 > \Delta, r_{l+1} < t', p_{l+1} < p_1$, and $\Delta + p_{l+1} \geq p_1$: The sum of the completion times for J_1 and J_{l+1} in the waiting and non-waiting schedules is $2 \cdot (t + \Delta + p_{l+1}) + p_1$ and $2 \cdot (t + p_1) + p_{l+1}$ respectively. If $\Delta + p_{l+1} > p_1$, it follows that the non-waiting schedule has a smaller total completion time. When $\Delta + p_{l+1} = p_1$, the machines next becomes idle at time $\min\{t + p_1, t'\}$ in both schedules. However, in the waiting schedule, J_1 is at the head of the queue while in the non-waiting schedule J_{l+1} is the shortest available job. It follows that we produce a smaller total completion time with the smaller set of waiting jobs. Therefore, when $\Delta + p_{l+1} \geq p_1$, we execute J_1.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

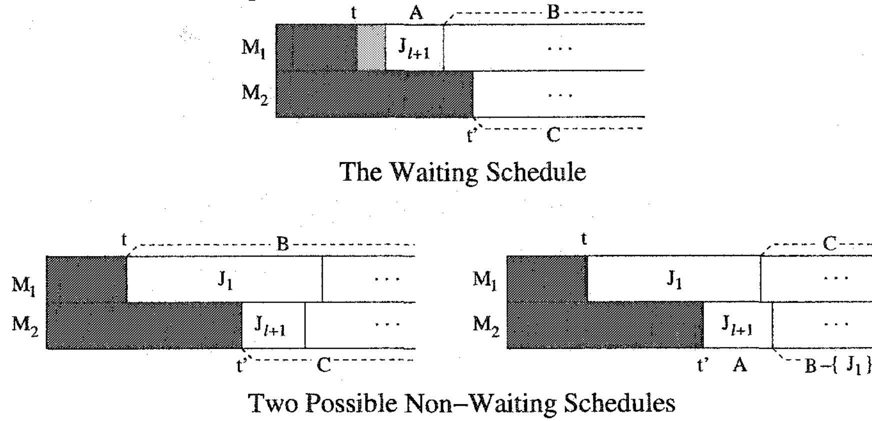
When $t < t', p_1 > \Delta, r_{l+1} < t', p_{l+1} < p_1$, and $\Delta + p_{l+1} < p_1$, the decision whether or not to wait requires further consideration. Using information about the relationship between J_1, J_{l+1}, and t', we further break the remaining possibilities into four cases:

- $t + p_1 > t'$ and $t + \Delta + p_{l+1} < t'$: In the non-waiting schedule, J_1 completes after t' and in the waiting schedule, J_{l+1} completes before t'.
- $t + p_1 > t'$ and $t + \Delta + p_{l+1} \geq t'$: In the non-waiting schedule, J_1 completes after t' and in the waiting schedule, J_{l+1} completes after t'.
- $t + p_1 \leq t'$ and $t + \Delta + p_{l+1} + p_1 \geq t'$: Note that the conditions in this case imply that $t + \Delta + p_{l+1} < t'$. In the non-waiting schedule, J_1 completes before t' and in the waiting schedule, J_{l+1} completes before t' and J_1 also completes on M_1 but after t'.
- $t + p_1 \leq t'$ and $t + \Delta + p_{l+1} + p_1 < t'$: In the non-waiting schedule, J_1 completes before t' and in the waiting schedule, both J_1 and J_{l+1} complete before t'.

Note that these four cases cover all the remaining possibilities. We consider each possibility in a lemma using the following definitions. First, we define three job sets based on the waiting schedule. Set A contains all jobs on M_1 which start after r_{l+1} and complete before t'. Note that A may be empty. Set B contains jobs on M_1 following A. Let J_b be the first, and therefore the shortest job in B. Finally, set C contains jobs on M_2 which start after t'. Note that because the jobs are scheduled in the order of non-decreasing processing times, jobs $B = \{J_b, J_{b+2}, J_{b+4} \ldots\}$ are scheduled on M_1 and jobs $C = \{J_{b+1}, J_{b+3}, J_{b+5} \ldots\}$ are scheduled on M_2. It follows that we have $|B| = |C|$ or $|B| = |C| + 1$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 4.4: In Lemma 4.1, J_{l+1} completes before time t' in the waiting schedule. Depending on the lengths of J_1 and J_{l+1}, there are two possible non-waiting schedules, one where J_{l+1} completes before J_1, and one where it completes after J_1.



Define a function *diff*, which gives the difference between the total wait times of a set of jobs in the waiting schedule and the non-waiting schedule (under consideration). Thus, if $\mathit{diff}(\{J_1, \ldots, J_l, J_{l+1}\}) \geq 0$, the lookahead algorithm chooses to schedule, and if $\mathit{diff}(\{J_1, \ldots, J_l, J_{l+1}\}) < 0$, the algorithm chooses to wait.

Lemma 4.1 *Suppose $t < t'$, $p_1 > \Delta$, $r_{l+1} < t'$, $p_{l+1} < p_1$, and $\Delta + p_{l+1} < p_1$. In addition, $t + p_1 > t'$ and $t + \Delta + p_{l+1} < t'$. The lookahead algorithm chooses to schedule J_1 on M_1 at t if and only if $(1+|B|)\Delta + (|B|-|C|-1)\min\{t'-t, p_1 - p_{l+1}\} + (|B|-|C|)(t+p_{l+1}-t') \geq 0$.*

Proof: To prove the lemma, all we need to show is that

$$\mathit{diff}(\{J_1, \ldots, J_l, J_{l+1}\}) = (1+|B|)\Delta + (|B|-|C|-1)\min\{t'-t, p_1-p_{l+1}\} + (|B|-|C|)(t+p_{l+1}-t').$$

Figure 4.4 shows the waiting schedule and two possible non-waiting schedules. In the waiting schedule, after waiting from t to r_{l+1}, M_1 executes the shortest job available, J_{l+1}. Because $t + \Delta + p_{l+1} < t'$, J_{l+1} finishes before t'. Note that because $t + p_1 > t'$, J_1 finishes after t'. Also, $|A| = 1$ containing J_{l+1}.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

If $t' + p_{l+1} \leq t + p_1$, then the first possible non-waiting schedule occurs and we have the following equations for the *diff* function.

$$\mathit{diff}(A) = t + \Delta - t',$$

$$\mathit{diff}(B) = |B|(\Delta + p_{l+1}), \text{ and}$$

$$\mathit{diff}(C) = |C|(-p_{l+1}).$$

Summing up, we get

$$\mathit{diff}(\{J_1, \ldots, J_l, J_{l+1}\}) = (1 + |B|)\Delta - (t' - t) + (|B| - |C|)p_{l+1}.$$

Because $t' + p_{l+1} \leq t + p_1$, $\min\{t' - t, p_1 - p_{l+1}\} = t' - t$. Also, $(|B| - |C|)p_{l+1} = (|B| - |C|)(t + p_{l+1} - t') + (|B| - |C|)(t' - t)$. Therefore,

$$\mathit{diff}(\{J_1, \ldots, J_l, J_{l+1}\}) = (1+|B|)\Delta + (|B| - |C| - 1)\min\{t' - t, p_1 - p_{l+1}\} + (|B| - |C|)(t + p_{l+1} - t').$$

If $t' + p_{l+1} > t + p_1$, then the second possible non-waiting schedule occurs, and we have the following equations for the *diff* function.

$$\mathit{diff}(A) = t + \Delta - t',$$

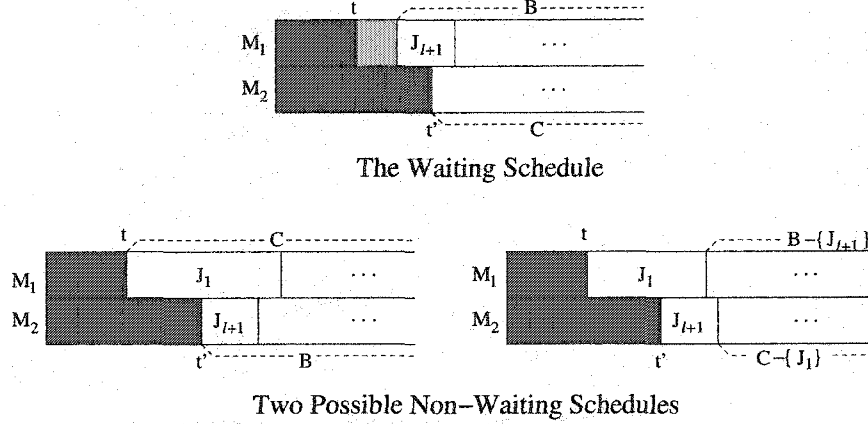
$$\mathit{diff}(\{J_1\}) = \Delta + p_{l+1},$$

$$\mathit{diff}(B - \{J_1\}) = (|B| - 1)(t + \Delta + p_1 - t'), \text{ and}$$

$$\mathit{diff}(C) = |C|(t' - t - p_1).$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 4.5: In Lemma 4.2, J_{l+1} completes execution after time t' in the waiting schedule. Depending on the lengths of J_1 and J_{l+1}, there are two possible non-waiting schedules, one where J_{l+1} completes before J_1, and the other where it completes after J_1.



Summing up, we get

$$diff(\{J_1, \ldots, J_l, J_{l+1}\}) = (1 + |B|)\Delta - (p_1 - p_{l+1}) + (|B| - |C|)(t + p_1 - t').$$

Because $t' + p_{l+1} > t + p_1$, $\min\{t' - t, p_1 - p_{l+1}\} = p_1 - p_{l+1}$. Also, $(|B| - |C|)(t + p_1 - t') = (|B| - |C|)(t + p_{l+1} - t') + (|B| - |C|)(p_1 - p_{l+1})$. Therefore,

$$diff(\{J_1, \ldots, J_l, J_{l+1}\}) = (1+|B|)\Delta + (|B|-|C|-1)\min\{t'-t, p_1-p_{l+1}\} + (|B|-|C|)(t+p_{l+1}-t').$$

■

Lemma 4.2 *Suppose $t < t'$, $p_1 > \Delta$, $r_{l+1} < t'$, $p_{l+1} < p_1$, and $\Delta + p_{l+1} < p_1$. In addition, $t + p_1 > t'$ and $t + \Delta + p_{l+1} \geq t'$. The lookahead algorithm chooses to schedule J_1 on M_1 at t if and only if $|B|\Delta + (|C| - |B|)\min\{t' - t, p_1 - p_{l+1}\} \geq 0$.*

Proof: To prove the lemma, all we need to show is that

$$diff(\{J_1, \ldots, J_l, J_{l+1}\}) = |B|\Delta + (|C| - |B|)min\{t' - t, p_1 - p_{l+1}\}.$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 4.5 shows the waiting schedule and two possible non-waiting schedules. In the waiting schedule, after waiting from t to r_{l+1}, M_1 executes the shortest job available, J_{l+1}. Because $t + \Delta + p_{l+1} \geq t'$, J_{l+1} finishes after t'. Also, A is empty and J_{l+1} is the first (also the shortest) in B.

If $t' + p_{l+1} \leq t + p_1$, where the first possible non-waiting schedule occurs, we have the following equations for the *diff* function.

$$diff(B) = |B|(t + \Delta - t'), \text{ and}$$

$$diff(C) = |C|(t' - t).$$

Summing up, we get

$$diff(\{J_1, \ldots, J_l, J_{l+1}\}) = |B|\Delta + (|C| - |B|)(t' - t).$$

Because $t' + p_{l+1} \leq t + p_1$, $\min\{t' - t, p_1 - p_{l+1}\} = t' - t$. Therefore,

$$diff(\{J_1, \ldots, J_l, J_{l+1}\}) = |B|\Delta + (|C| - |B|)min\{t' - t, p_1 - p_{l+1}\}.$$

If $t' + p_{l+1} > t + p_1$, where the second possible non-waiting schedule occurs, we have the following equations for the *diff* function.

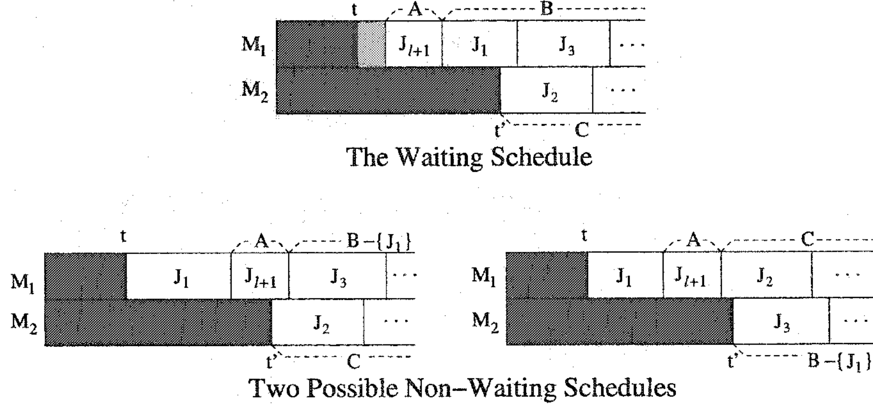
$$diff(\{J_1\}) = t' - t,$$

$$diff(\{J_{l+1}\}) = t + \Delta - t',$$

$$diff(B - \{J_{l+1}\}) = (|B| - 1)(\Delta + p_{l+1} - p_1), \text{ and}$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 4.6: In Lemma 4.3, J_{l+1} completes execution before time t' in the waiting schedule. Depending on the lengths of J_1 and J_{l+1}, there are two possible non-waiting schedules, one where both jobs are scheduled on the same machine, and one where they are on different machines.



$$diff(C - \{J_1\}) = (|C| - 1)(p_1 - p_{l+1}).$$

Summing up, we get

$$diff(\{J_1, \ldots, J_l, J_{l+1}\}) = |B|\Delta + (|C| - |B|)(p_1 - p_{l+1}).$$

Because $t' + p_{l+1} > t + p_1$, $\min\{t' - t, p_1 - p_{l+1}\} = p_1 - p_{l+1}$. Therefore,

$$diff(\{J_1, \ldots, J_l, J_{l+1}\}) = |B|\Delta + (|C| - |B|)min\{t' - t, p_1 - p_{l+1}\}.$$

■

Lemma 4.3 *Suppose $t < t'$, $p_1 > \Delta$, $r_{l+1} < t'$, $p_{l+1} < p_1$, and $\Delta + p_{l+1} < p_1$. In addition, $t + p_1 \leq t'$ and $t' \leq t + \Delta + p_{l+1} + p_1$. The lookahead algorithm chooses to schedule J_1 on M_1 at t if and only if $(1 + |B|)\Delta + p_{l+1} - p_1 + (|C| - |B| + 1)\max\{0, t' - t - p_{l+1} - p_b\} \geq 0$.*

Proof: To prove the lemma, all we need to show is that

$$diff(\{J_1, \ldots, J_l, J_{l+1}\}) = (1 + |B|)\Delta + p_{l+1} - p_1 + (|C| - |B| + 1)\max\{0, t' - t - p_{l+1} - p_b\}.$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 4.6 shows the waiting schedule and two possible non-waiting schedules. In the waiting schedule, J_{l+1} is the only job in A because $t + \Delta + p_{l+1} < t' \leq t + \Delta + p_{l+1} + p_1$. Also, J_1 is scheduled on M_1 and completes after t'.

If $t + p_1 + p_{l+1} > t'$, where the first possible non-waiting schedule occurs, we have the following equations for the *diff* function.

$$\mathit{diff}(\{J_1\}) = \Delta + p_{l+1},$$

$$\mathit{diff}(\{J_{l+1}\}) = \Delta - p_1,$$

$$\mathit{diff}(B - \{J_1\}) = (|B| - 1)\Delta, \text{ and}$$

$$\mathit{diff}(C) = 0.$$

Summing up, we get

$$\mathit{diff}(\{J_1, \ldots, J_l, J_{l+1}\}) = (1 + |B|)\Delta + p_{l+1} - p_1.$$

From the non-waiting schedule, we have $t' - t - p_{l+1} - p_1 < 0$. So $(|C| - |B| + 1)\max\{0, t' - t - p_{l+1} - p_1\} = 0$. Therefore,

$$\mathit{diff}(\{J_1, \ldots, J_l, J_{l+1}\}) = (1 + |B|)\Delta + p_{l+1} - p_1 + (|C| - |B| + 1)\max\{0, t' - t - p_{l+1} - p_b\}.$$

If $t + p_1 + p_{l+1} \leq t'$, where the second possible non-waiting schedule occurs, we have the following equations for the *diff* function.

$$\mathit{diff}(\{J_1\}) = \Delta + p_{l+1},$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$$diff(\{J_{l+1}\}) = \Delta - p_1,$$

$$diff(B - \{J_1\}) = (|B| - 1)(t + \Delta + p_{l+1} + p_1 - t'), \text{ and}$$

$$diff(C) = |C|(t' - t - p_{l+1} - p_1).$$

Summing up, we get

$$diff(\{J_1, \ldots, J_l, J_{l+1}\}) = (1 + |B|)\Delta + p_{l+1} - p_1 + (|C| - |B| + 1)(t' - t - p_{l+1} - p_1).$$

Because $t' - t - p_{l+1} - p_1 \geq 0$ from the non-waiting schedule, therefore,

$$diff(\{J_1, \ldots, J_l, J_{l+1}\}) = (1 + |B|)\Delta + p_{l+1} - p_1 + (|C| - |B| + 1)\max\{0, t' - t - p_{l+1} - p_b\}.$$

■

Lemma 4.4 *Suppose $t < t'$, $p_1 > \Delta$, $r_{l+1} < t'$, $p_{l+1} < p_1$, and $\Delta + p_{l+1} < p_1$. In addition, $t + p_1 \leq t'$ and $t + \Delta + p_{l+1} + p_1 < t'$. The lookahead algorithm chooses to schedule J_1 on M_1 at t if and only if $(|A| + |B|)\Delta + p_{l+1} - p_1 + (|C| - |B| + 1)\max\{0, t' - t - \sum_A p_j - p_b\} \geq 0$.*

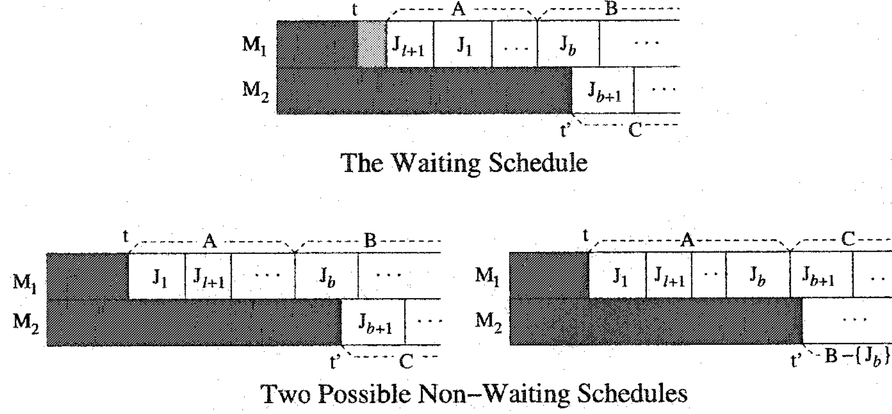
Proof: To prove the lemma, all we need to show is that

$$diff(\{J_1, \ldots, J_l, J_{l+1}\}) = (|A| + |B|)\Delta + p_{l+1} - p_1 + (|C| - |B| + 1)\max\{0, t' - t - \sum_A p_j - p_b\}.$$

Figure 4.7 shows the waiting schedule and two possible non-waiting schedules. In the waiting schedule, the set A contains both J_{l+1} and J_1 because $t + \Delta + p_{l+1} + p_1 < t'$. It may also be the case that more jobs are in A.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 4.7: In Lemma 4.4, multiple jobs complete before time t' in the waiting schedule. Depending on the lengths of J_1 and J_{l+1}, there are two possible non-waiting schedules, one where the jobs are on the same machine as the waiting schedule, and one where a single job is on the other machine.



If $t + \sum_A p_i + p_b > t'$, where the first possible non-waiting schedule occurs, we have the following equations for the *diff* function.

$$diff(\{J_1\}) = \Delta + p_{l+1},$$

$$diff(\{J_{l+1}\}) = \Delta - p_1,$$

$$diff(A - \{J_1, J_{l+1}\}) = (|A| - 2)\Delta,$$

$$diff(B) = |B|\Delta, \text{ and}$$

$$diff(C) = 0.$$

Summing up, we get

$$diff(\{J_1, \ldots, J_l, J_{l+1}\}) = (|A| + |B|)\Delta + p_{l+1} - p_1.$$

From the non-waiting schedule, we have $t' - t - \sum_A p_j - J_b < 0$. So $(|C| - |B| + 1)\max\{0, t' -$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$t - \sum_A p_j - p_b\} = 0$. Therefore,

$$diff(\{J_1, \ldots, J_l, J_{l+1}\}) = (|A|+|B|)\Delta + p_{l+1} - p_1 + (|C|-|B|+1)\max\{0, t' - t - \sum_A p_j - p_b\}.$$

If $t + \sum_A p_i + p_b \leq t'$, where the second possible non-waiting schedule occurs, we have the following equations for the *diff* function.

$$diff(\{J_1\}) = \Delta + p_{l+1},$$

$$diff(\{J_{l+1}\}) = \Delta - p_1,$$

$$diff(A - \{J_1, J_{l+1}\}) = (|A| - 2)\Delta,$$

$$diff(\{J_b\}) = \Delta,$$

$$diff(B - \{J_b\}) = (|B| - 1)(t + \Delta + \sum_A p_j + p_b - t'), \text{ and}$$

$$diff(C) = |C|(t' - t - \sum_A p_j - p_b).$$

Summing up, we get

$$diff(\{J_1, \ldots, J_l, J_{l+1}\}) = (|A| + |B|)\Delta + p_{l+1} - p_1 + (|C| - |B| + 1)(t' - t - \sum_A p_j - p_b).$$

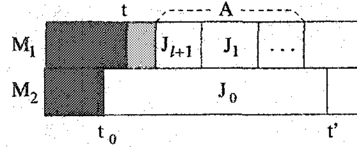
Because $t' - t - \sum_A p_j - p_b \geq 0$ from the non-waiting schedule, therefore,

$$diff(\{J_1, \ldots, J_l, J_{l+1}\}) = (|A|+|B|)\Delta + p_{l+1} - p_1 + (|C|-|B|+1)\max\{0, t' - t - \sum_A p_j - p_b\}.$$

■

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 4.8: The set A is defined to be all the jobs that begin execution after time r_{l+1} and complete before time t'.



4.2.2 LA1 Algorithm Time Complexity

Although the proofs for each of the last four cases require significant effort, the resulting condition can be easily evaluated. Together, the conditions demonstrate that scheduling decisions can be made without completely calculating the waiting and non-waiting schedules. In constant time, we can evaluate nearly all the conditions. In the cases 1, 2, and 3, $|A|$ is either 0 or 1. Knowing $|A|$, we can then easily calculate $|B|$ and $|C|$. Therefore, in these cases, we can decide whether to schedule or wait in $\mathcal{O}(1)$ time.

In case 4, $t' - t$ is large compared to the size of the jobs in the wait queue. This allows multiple jobs to complete on M_1 between t and t'. As a result, $|A| > 1$. Therefore, to compute $|A|$ and $\sum_A p_j$ the algorithm considers the jobs in Q_W in order until it finds the first job that completes after t'. In the worst case, we may have to check the entire wait queue.

Notice, however, that this check only occurs in one of four cases. Further, the time needed to calculate the schedule is proportional to the number of jobs in A that fit in the interval $[t, t']$. Therefore, if we show that the average size of A is small, then we can conclude that the running time of the algorithm is near-constant.

Formally, assume that the jobs arrive with an average inter-arrival time of μ_a and these jobs have an average processing time of μ_p. Without loss of generality, assume machine M_1

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

is idle at time t. Let J_0 be the last job scheduled on M_2 with starting time t^0 and completion time $t' = t^0 + p_0$. Let Q_W be the wait queue containing jobs $J_1 \ldots J_l$ with processing times $p_1 \leq \ldots \leq p_l$. Finally, let J_{l+1} with arrival time $r_{l+1} > t$ and processing time p_{l+1} be the next job to arrive. We are interested in the schedule where we wait for job J_{l+1} to arrive and then immediately schedule it on M_1. The jobs in the wait queue are then scheduled on M_1 using the SJF rule. In this context, we define A to be the set of jobs scheduled on M_1 that complete execution between t and t'. See Figure 4.8. Note that $0 \leq |A| \leq l+1$.

Lemma 4.5 *In case* $\boxed{4}$, $E[|A|] \leq 3$

Proof First, note that $t' > t$. We know that if A contains any jobs, J_{l+1} is in the set. So let $A' - A - J_{l+1}$ be the remaining jobs. At time t^0, J_0 is the smallest job in the wait queue. Because $t > t'$, none of the jobs in the queue at t^0 are in A. Therefore, all the jobs in A' must arrive between times t^0 and t'. Further, each of these jobs must have a processing time less than or equal to $t' - t^0$. Therefore,

$$E[|A'|] \leq Pr\{p_j < t' - t^0\} \cdot E[\text{num jobs arrive during } t' - t^0]$$

Because J_0 can be any job, on average $\mu_p = p_0 = t' - t^0$. Thus,

$$E[|A'|] \leq Pr\{p_j < \mu_p\} \cdot \frac{\mu_p}{\mu_a}$$

Now consider J_{l+1} and the original set of interest, A. Note that if $r_{l+1} + p_{l+1} - t \geq t' - t$, then $|A| = 0$. Otherwise, we have at most $1 + |A'|$ jobs that fit in the time span $t' - t$. Therefore,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$$E[|A|] \leq Pr\{r_{l+1} + p_{l+1} - t < \mu_p\} \left[1 + Pr\{p_j < \mu_p\} \cdot \frac{\mu_p}{\mu_a} \right]$$

This equation says that the average size of A is less than the probability that J_{l+1} fits in the time span $t' - t^0$ plus a fraction of the traffic intensity, $\rho = \mu_p/\mu_a$. For a two-machine system, the traffic intensity must remain less than 2, otherwise the wait queue tends to grow toward infinity. Therefore, we assume $\rho \leq 2$ and $E[|A|] \leq 3$. □

Table 4.1: We ran the simulation for one million scheduling decisions and tabulated all occurring sizes of the set A along with the frequencies.

$\|A\|$	count	percent
0	352220	0.3529
1	235921	0.2364
2	154590	0.1549
3	95025	0.0952
4	58075	0.0582
5	35864	0.0359
6	22843	0.0229
7	14937	0.0150
8	9963	0.0100
9	6534	0.0065
10	4345	0.0044
⋮		
25	1	0.0000

As further evidence, we observed $|A|$ under a variety of conditions using simulation. Table 4.1 shows the range of sizes which occurred with the associated frequency. For sizes 0 and 1, we evaluate one of the first 3 cases. This represents nearly 59% of the samples. Another 40% of the samples fall in the range 2-10. The pattern of decreasing frequency continues until reaching the maximum size of 25, which occurred once in nearly one million trials. Considering all the samples, the mean of $|A|$ was 1.766 with a standard deviation of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 4.9: When a decision must be made in the single machine system, the Lookahead-2 algorithm has access to jobs J_1 through J_l in the wait queue plus jobs J_{l+1} and J_{l+2} with arrival times r_{l+1} and r_{l+2} in the lookahead queue.

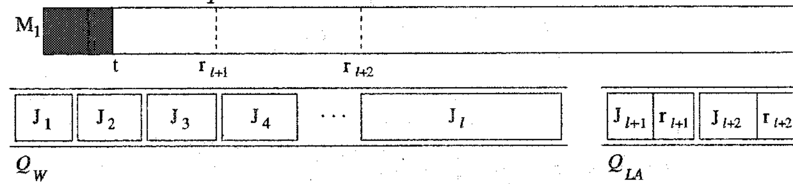
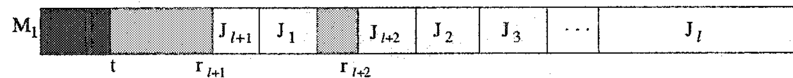


Figure 4.10: If the Lookahead-2 algorithm decides to wait at both decisions, there are two idle periods in the schedule. We call this the wait-wait schedule.



2.201. Based on these measurements, we conclude the algorithm runs in nearly constant time in case [4].

Waiting for the lookahead job to arrive affects the completion time of every job in the queue and therefore can dramatically improve the total completion time. We have shown that we do not need to consider the completion time of each individual job. In most cases, we can decide whether or not lookahead is beneficial by using $t, t', p_1, \Delta, p_{l+1}$, and l, which are all constants for any given decision. In the case where we do have to consider the jobs in A, we have shown that this set is small. Therefore, the computation necessary to decide whether to wait or schedule requires only a few constant time comparisons. A standard Shortest Job First algorithm without lookahead would use only the information about the first job in the queue to make scheduling decisions. We can apply a more powerful lookahead heuristic in near constant time and gain improvements. Thus, these improvements come at almost no additional cost.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 4.11: If the Lookahead-2 algorithm decides to wait at time t and then schedule at the second decision, there is a single idle period at the beginning of the schedule. We call this the wait-nowait schedule.

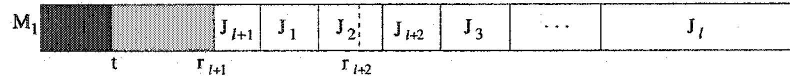
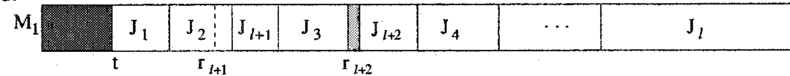


Figure 4.12: If the Lookahead-2 algorithm decides to schedule at time t and then wait for the second lookahead job, there is a single idle period in the middle of the schedule. We call this the nowait-wait schedule.



4.3 A Lookahead-2 Algorithm for a Single Machine

In this section we discuss a lookahead-2 algorithm for a single machine system. Formally, $1|r_j|\sum C_j$ is the problem where we have jobs arriving over time to be processed on a single machine. The goal, as before, is to minimize the sum of completion times.

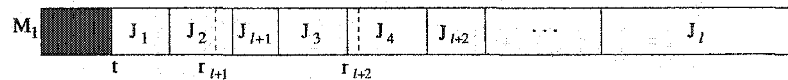
The scheduling system is nearly identical to the two machine model discussed in the previous section, with the exception that the schedule always has access to the next two arriving jobs. Figure 4.9 shows a representative case where jobs J_1 through J_l are in the wait queue and jobs J_{l+1} with arrival time r_{l+1} and J_{l+2} with arrival time r_{l+2} are in the lookahead queue.

Our algorithm is broken down into two major parts. The first considers cases when both J_{l+1} and J_{l+2} would arrive during the execution of J_1. In this case, a decision to wait either means that the algorithm intends to wait until r_{l+1} and schedule J_{l+1} or wait until r_{l+2} and schedule J_{l+2}.

In the second, more general situation, r_{l+2} occurs far enough in the future that one or more jobs can complete execution before J_{l+2} arrives. Here, the decision of the algorithm is dependent on where r_{l+2} falls on the time-line compared to the completion times of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 4.13: If the Lookahead-2 algorithm decides to schedule at both decisions, then there are no idle periods in the schedule. We call this the nowait-nowait schedule.



scheduled jobs. Therefore, the decision to wait or schedule at time t has an effect on the later decision because it determines when the completion times occur. This means that in the general case, we have four schedules:

- W/W – wait-wait (Figure 4.10) – The algorithm decides to wait for the first lookahead job at time t and then wait again for the second lookahead job at the appropriate time.
- $W/\overline{W}$ – wait-nowait (Figure 4.11) – The algorithm waits for the first lookahead job but then does not wait for the second job. Instead, the second job arrives during the execution of another job and is added to the wait queue to be scheduled later.
- $\overline{W}/W$ – nowait-wait (Figure 4.12) – The algorithm does not wait for the first job, but rather schedules J_1. When J_{l+1} arrives, it is added to the wait queue. Jobs are scheduled in sorted order until immediately before J_{l+2} would arrive and then the scheduler waits for this job to arrive.
- $\overline{W}/\overline{W}$ – nowait-nowait (Figure 4.13) – For both jobs, the algorithm does not wait and simply adds the jobs to the wait queue when they arrive.

In many cases, we can eliminate one or more of these possibilities by arguing that they always produce a worse schedule than other decisions.

In our algorithm that follows, we break the decision process down into multiple cases based on the following parameters:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- t (the current time)
- $p_1, \ldots, p_l$ (processing times of the jobs in Q_W)
- p_{l+1} and r_{l+1} (attributes of the first lookahead job)
- p_{l+2} and r_{l+2} (attributes of the second lookahead job)

For notational simplicity, define $\Delta = r_{l+1} - t$ and $\Delta' = r_{l+2} - t$. Definitions of the other terms will be given in the next section.

Procedure Lookahead-2

if Q_W is empty

then return

if Q_{LA} is empty

then schedule

else if $(l+2)\Delta' \leq \max\{0, p_1 - p_{l+1}\} + \max\{0, p_1 - p_{l+2}\}$ and any of the following:

- $p_1 \leq \Delta$,
- $p_1 > \Delta$ and $r_{l+1} = r_{l+2}$
- $p_1 \geq \Delta'$ and $r_{l+2} > r_{l+1}$ and $p_1 \leq p_{l+1}$
- $p_1 \geq \Delta'$ and $r_{l+2} > r_{l+1}$ and $p_1 > p_{l+1}$ and $(l+2)(r_{l+2} - r_{l+1}) \leq p_{l+1} - p_{l+2}$
- $\Delta < p_1 < \Delta'$ and $r_{l+2} > r_{l+1}$ and $p_1 \leq p_{l+1}$

then wait

else if $p_1 \leq \Delta$

then schedule

else if $p_1 > \Delta$ and $r_{l+1} = r_{l+2}$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

if $p_1 \leq p_{l+1}$

then schedule

else if $(l+2)\Delta' \leq \max\{0, p_1 - p_{l+1}\} + \max\{0, p_1 - p_{l+2}\}$

then wait

else schedule

else if $p_1 \geq \Delta'$ and $r_{l+2} > r_{l+1}$ and $p_1 \leq p_{l+1}$

if $p_1 \leq p_{l+2}$

then schedule

else if $(l+2)\Delta' \leq \max\{0, p_1 - p_{l+1}\} + \max\{0, p_1 - p_{l+2}\}$

then wait

else schedule

else if $p_1 \geq \Delta'$ and $r_{l+2} > r_{l+1}$ and $p_1 > p_{l+1}$ and $(l+2)(r_{l+2} - r_{l+1}) \leq p_{l+1} - p_{l+2}$

if $(l+2)\Delta' \leq \max\{0, p_1 - p_{l+1}\} + \max\{0, p_1 - p_{l+2}\}$

then wait

else schedule

else if $\Delta < p_1 < \Delta'$ and $r_{l+2} > r_{l+1}$ and $p_1 \leq p_{l+1}$

then execute

else if $p_1 \geq \Delta'$ and $r_{l+2} > r_{l+1}$ and $p_1 > p_{l+1}$ and $(l+2)(r_{l+2} - r_{l+1}) > p_{l+1} - p_{l+2}$

if $(l+2)\Delta + \min\{\max\{0, p_1 - p_{l+2}\}, (l+1)\max\{0, r_{l+2} - r_{l+1} - p_{l+1}\}\} \leq$

$$\max\{0, p_1 - p_{l+1}\} + \max\{0, p_1 - p_{l+2}\} - \max\{0, p_{l+1} - p_{l+2}\}.$$

then wait

else schedule

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

else if $\Delta < p_1 < \Delta'$ and $r_{l+2} > r_{l+1}$ and $p_1 > p_{l+1}$ and $(l+1)(\Delta' - p_1) \leq p_{l+1} - p_{l+2}$

if $(l+2)\Delta - (l+1)(\Delta' - p_1) \leq \max\{0, p_1 - p_{l+1}\} - \max\{0, p_{l+1} - p_{l+2}\}$

then wait

else schedule

else if $\Delta < p_1 < \Delta'$ and $r_{l+2} > r_{l+1}$ and $p_1 > p_{l+1}$ and $(l+1)(\Delta' - p_1) > p_{l+1} - p_{l+2}$

and $p_{l+1} \geq \Delta' - p_1$

if $(l+2)\Delta + \min\{\max\{0, p_1 - p_{l+2}\}, (l+1)\max\{0, r_{l+2} - r_{l+1} - p_{l+1}\}\} \leq$

$\max\{0, p_1 - p_{l+1}\} + \max\{0, p_1 - p_{l+2}\}$

then wait

else schedule

else if $\Delta < p_1 < \Delta'$ and $r_{l+2} > r_{l+1}$ and $p_1 > p_{l+1}$ and $(l+1)(\Delta' - p_1) > p_{l+1} - p_{l+2}$

and $p_{l+1} < \Delta' - p_1$ and $g = h + 1$

if $(h+2)\Delta + (l-h)(\Delta + p_{l+1} + \sum_{j=1}^{h+1} p_j - \Delta') + \max\{(l-h)(\Delta' - p_{l+1} - \sum_{j=1}^{h+1} p_j,)$

$p_{h+2} - p_{l+2}\} \leq \max\{0, p_1 - p_{l+1}\} + \max\{0, p_{h+2} - p_{l+2}\}$

then wait

else schedule

else if $\Delta < p_1 < \Delta'$ and $r_{l+2} > r_{l+1}$ and $p_1 > p_{l+1}$ and $(l+1)(\Delta' - p_1) > p_{l+1} - p_{l+2}$

and $p_{l+1} < \Delta' - p_1$ and $g = h$

if $(l+2)\Delta + \min\{(l-h+1)(\Delta' - \Delta - p_{l+1} - \sum_{j=1}^{h} p_j), p_{h+1} - p_{l+2}\}$

$- \min\{(l-h+1)(\Delta' - p_{l+1} - \sum_{j=1}^{h} p_j), p_{h+1} - p_{l+2}\} \leq \max\{0, p_1 - p_{l+1}\}$

then wait

else schedule

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.3.1 Theoretical Justification of the LA2 Algorithm

Just as we proved the correctness for the LA1 algorithm on the two machine system in Section 4.2.1, the goal of this section is to demonstrate that the conditions in the Lookahead-2 algorithm allow it to make local optimal decisions. That is, we want to show that the evaluation of these conditions is sufficient for the algorithm to conclude whether to wait or schedule.

Although the algorithm appears more complex than the LA1 algorithm, many of the conditions relate to "trivial" cases where the decision of the algorithm is obvious. For instance, Lemmas 4.6 and 4.7 consider the case when J_1 is shorter than both J_{l+1} and J_{l+2} and all the cases when J_1 is longer than Δ'.

In the remainder of the cases, the four general schedule types discussed in the previous section are now feasible. We break these cases down by two factors. First, in Lemmas 4.8 and 4.9, we consider schedules where the lookahead jobs arrive close together and, after the first lookahead job arrives, we can immediately use the single machine LA1 algorithm of Mao and Kincaid [79] to decide whether to wait or schedule.

The final two lemmas, numbers 4.10 and 4.11, consider the cases when the two lookahead jobs have arrival times significantly separated. As a result, more than a single job can be scheduled before J_{l+2} arrives, regardless of whether the algorithm decides to wait or schedule.

Lemma 4.6 *For any of the following cases,*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

1. $p_1 \leq \Delta$,

2. $p_1 > \Delta$ *and* $r_{l+1} = r_{l+2}$,

3. $p_1 \geq \Delta'$ *and* $r_{l+2} > r_{l+1}$ *and* $p_1 \leq p_{l+1}$,

4. $p_1 \geq \Delta'$ *and* $r_{l+2} > r_{l+1}$ *and* $p_1 > p_{l+1}$ *and* $(l+2)(r_{l+2} - r_{l+1}) \leq p_{l+1} - p_{l+2}$, *or*

5. $\Delta < p_1 < \Delta'$ *and* $r_{l+2} > r_{l+1}$ *and* $p_1 \leq p_{l+1}$,

the algorithm chooses to wait until r_{l+1} *if and only if*

$$(l+2)\Delta' \leq \max\{0, p_1 - p_{l+1}\} + \max\{0, p_1 - p_{l+2}\}. \tag{4.1}$$

Note that the "if and only if" condition also implies that if the above inequality is evaluated false the algorithm chooses to execute J_1 *at time* t.

Proof We prove the lemma for each case listed.

1. $p_1 \leq \Delta$: Clearly, it is more beneficial to execute J_1 than waiting because the job's length p_1 fits into the waiting period Δ. The choice for the algorithm is to execute. The evaluation of inequality (4.1) also suggests the same action for the algorithm because $(l+2)\Delta' \geq (l+2)\Delta \geq (l+2)p_1 > 2p_1 > \max\{0, p_1 - p_{l+1}\} + \max\{0, p_1 - p_{l+2}\}$.

2. $p_1 > \Delta$ and $r_{l+1} = r_{l+2}$: Recall our assumption $p_{l+1} \leq p_{l+2}$ because $r_{l+1} = r_{l+2}$. Also note that $\Delta' = \Delta$ because $r_{l+1} = r_{l+2}$. The fact that $p_1 > \Delta'$ implies that in the $\overline{W}$ schedule in which J_1 is scheduled from t to $t + p_1$, when J_1 is completed the two lookahead jobs, J_{l+1} and J_{l+2}, will have already arrived. Consider two possibilities of p_1 when compared with p_{l+1}: (2.a) $p_1 \leq p_{l+1}$ and (2.b) $p_1 > p_{l+1}$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(2.a) When $p_1 \leq p_{l+1}$, J_1 remains to be the shortest in Q_W at time r_{l+1} $(= r_{l+2})$ once both J_{l+1} and J_{l+2} have arrived and been added to Q_W. This means that in the W schedule J_1 is executed at time r_{l+2}, while in the $\overline{W}$ schedule J_1 is executed at an earlier time of t. Obviously, the $\overline{W}$ schedule has a smaller total completion time than the W schedule. The choice for the algorithm is to execute. The evaluation of inequality (4.1) also suggests the same action for the algorithm because $(l+2)\Delta' > 0 = \max\{0, p_1 - p_{l+1}\} + \max\{0, p_1 - p_{l+2}\}$.

(2.b) When $p_1 > p_{l+1}$, we compute the total completion times, denoted as $\sum C_j(W)$ and $\sum C_j(\overline{W})$, of $J_1, \ldots, J_l, J_{l+1}, J_{l+2}$ in the W and $\overline{W}$ schedules, respectively and compare the two quantities. We have

$$\sum C_j(W) = (l+2)r_{l+2} + (l+2)p_{l+1} + (l+1)\min\{p_1, p_{l+2}\} + l\min\{\max\{p_1, p_{l+2}\}, p_2\} + \cdots$$

and

$$\sum C_j(\overline{W}) = (l+2)t + (l+2)p_1 + (l+1)p_{l+1} + l\min\{p_2, p_{l+2}\} + \cdots.$$

Notice that the "$\cdots$" parts in the above equations represent the same quantity. Considering the difference, we get

$$\sum C_j(W) - \sum C_j(\overline{W}) = (l+2)\Delta' - (p_1 - p_{l+1}) - \max\{0, p_1 - p_{l+2}\}.$$

Therefore, the algorithm chooses to wait if and only if $\sum C_j(W) - \sum C_j(\overline{W}) \leq 0$ if and only if $(l+2)\Delta' \leq (p_1 - p_{l+1}) + \max\{0, p_1 - p_{l+2}\}$ if and only if $(l+2)\Delta' \leq \max\{0, p_1 - p_{l+1}\} + \max\{0, p_1 - p_{l+2}\}$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3. $p_1 \geq \Delta'$ and $r_{l+2} > r_{l+1}$ and $p_1 \leq p_{l+1}$: First notice that because $p_1 \geq \Delta'$, in the $\overline{W}$ schedule the two lookahead jobs, J_{l+1} and J_{l+2}, have arrived when J_1 is completed at time $t + p_1$. For the W schedule, we further classify it into the W_2 schedule (wait until r_{l+2}) and the W_1 schedule (wait until r_{l+1} and then execute). Because $p_1 \leq p_{l+1}$, the W_1 schedule has the machine idle from t to r_{l+1} and then executes the shortest job available at r_{l+1}, which is J_1. This schedule certainly is no better than the $\overline{W}$ schedule which executes J_1 at an earlier time of t. Thus, the W_1 schedule is out of consideration. We only need to compare the W_2 schedule and the $\overline{W}$ schedule. Consider two possibilities of p_1 when compared with p_{l+2}: (3.a) $p_1 \leq p_{l+2}$ and (3.b) $p_1 > p_{l+2}$.

(3.a) When $p_1 \leq p_{l+2}$, the W_2 schedule waits from t to r_{l+2} and then executes J_1, which is no better than the $\overline{W}$ schedule. The choice for the algorithm is to execute. The evaluation of inequality (4.1) also suggests the same action for the algorithm because $(l+2)\Delta' > 0 = \max\{0, p_1 - p_{l+1}\} + \max\{0, p_1 - p_{l+2}\}$.

(3.b) When $p_1 > p_{l+2}$, we compute and compare $\sum C_j(W_2)$ and $\sum C_j(\overline{W})$. We have

$$\sum C_j(W_2) = (l+2)r_{l+2} + (l+2)p_{l+2} + (l+1)p_1 + \cdots$$

and

$$\sum C_j(\overline{W}) = (l+2)t + (l+2)p_1 + (l+1)p_{l+2} + \cdots .$$

Considering the difference, we get

$$\sum C_j(W_2) - \sum C_j(\overline{W}) = (l+2)\Delta' - (p_1 - p_{l+2}).$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Therefore, the algorithm chooses to wait if and only if $\sum C_j(W_2) - \sum C_j(\overline{W}) \leq 0$ if and only if $(l+2)\Delta' \leq p_1 - p_{l+2}$ if and only if $(l+2)\Delta' \leq \max\{0, p_1 - p_{l+1}\} + \max\{0, p_1 - p_{l+2}\}$.

4. $p_1 \geq \Delta'$ and $r_{l+2} > r_{l+1}$ and $p_1 > p_{l+1}$ and $(l+2)(r_{l+2} - r_{l+1}) \leq p_{l+1} - p_{l+2}$: Similar to the previous case, we further classify the W schedule into the W_2 schedule (wait until r_{l+2}) and the W_1 schedule (wait until r_{l+1} and then execute). Because both schedules wait from t to r_{l+1} the problem becomes lookahead-1 at time r_{l+1} with $Q_W = \{J_{l+1}, J_1, \ldots, J_l\}$ and $Q_{LA} = \{J_{l+2}\}$. Using the result developed for lookahead-1, we get $\sum C_j(W_2) \leq \sum C_j(W_1)$ if and only if $(l+2)(r_{l+2} - r_{l+1}) \leq p_{l+1} - p_{l+2}$. Because the latter inequality is given, $\sum C_j(W_2) \leq \sum C_j(W_1)$, implying that the W_1 schedule is out of consideration. We only need to compare the W_2 schedule and the $\overline{W}$ schedule. Recall that $p_1 > p_{l+1}$. The given $(l+2)(r_{l+2} - r_{l+1}) \leq p_{l+1} - p_{l+2}$ also suggests $p_{l+1} > p_{l+2}$. We compute and compare $\sum C_j(W_2)$ and $\sum C_j(\overline{W})$. We have

$$\sum C_j(W_2) = (l+2)r_{l+2} + (l+2)p_{l+2} + (l+1)p_{l+1} + lp_1 + \cdots$$

and

$$\sum C_j(\overline{W}) = (l+2)t + (l+2)p_1 + (l+1)p_{l+2} + lp_{l+1} + \cdots .$$

Considering the difference, we get

$$\sum C_j(W_2) - \sum C_j(\overline{W}) = (l+2)\Delta' - (p_1 - p_{l+1}) + (p_1 - p_{l+2}).$$

Therefore, the algorithm chooses to wait if and only if $\sum C_j(W_2) - \sum C_j(\overline{W}) \leq 0$ if and only if $(l+2)\Delta' \leq (p_1 - p_{l+1}) + (p_1 - p_{l+2})$ if and only if $(l+2)\Delta' \leq \max\{0, p_1 - p_{l+1}\} +$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$\max\{0, p_1 - p_{l+2}\}$.

5. $\Delta < p_1 < \Delta'$ and $r_{l+2} > r_{l+1}$ and $p_1 \leq p_{l+1}$: Unlike the previous cases, if the algorithm decides to schedule at time t, then when J_1 completes execution at $t + j_1$, the second lookahead job has not arrived. Therefore, in the W_2 schedule there is a period from t to r_{l+2} idle where J_1 could complete execution. Clearly, it is not beneficial to wait until r_{l+2}; the W_2 schedule is out of consideration.

If the algorithm decides to wait until r_{l+1} and then schedule, then J_{l+1} is added to the queue somewhere behind J_1 and therefore J_1 is scheduled. However, the algorithm could have simply scheduled J_1 at time t and reduced the waiting times for all jobs. Therefore, the algorithm should always decide to schedule. The evaluation of inequality (4.1) also suggests the same action for the algorithm because $(l+2)\Delta' > (l+2)p_1 > 2p_1 > \max\{0, p_1 - p_{l+1}\} + \max\{0, p_1 - p_{l+2}\}$. □

Lemma 4.6 covers many cases except the following two cases:

- $p_1 \geq \Delta'$ and $r_{l+2} > r_{l+1}$ and $p_1 > p_{l+1}$ and $(l+2)(r_{l+2} - r_{l+1}) > p_{l+1} - p_{l+2}$
- $\Delta < p_1 < \Delta'$ and $r_{l+2} > r_{l+1}$ and $p_1 > p_{l+1}$

We have Lemma 4.7 to handle the first uncovered case while the second case requires more detailed study later.

Lemma 4.7 *When $p_1 \geq \Delta'$ and $r_{l+2} > r_{l+1}$ and $p_1 > p_{l+1}$ and $(l+2)(r_{l+2} - r_{l+1}) > p_{l+1} - p_{l+2}$, the algorithm chooses to wait until r_{l+1} if and only if*

$$(l+2)\Delta + \min\{\max\{0, p_1 - p_{l+2}\}, (l+1)\max\{0, r_{l+2} - r_{l+1} - p_{l+1}\}\}$$
$$\leq \max\{0, p_1 - p_{l+1}\} + \max\{0, p_1 - p_{l+2}\} - \max\{0, p_{l+1} - p_{l+2}\}. \quad (4.2)$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Proof In the $\overline{W}$ schedule, the given $p_1 \geq \Delta'$ implies that when J_1 is completed at time $t + p_1$, the two lookahead jobs, J_{l+1} and J_{l+2}, have arrived. In the waiting schedule, we apply the lookahead-1 result to time r_{l+1} with $Q_W = \{J_{l+1}, J_1, \ldots, J_l\}$ and $Q_{LA} = \{J_{l+2}\}$. Because $(l+2)(r_{l+2} - r_{l+1}) > p_{l+1} - p_{l+2}$, $\sum C_j(W_2) > \sum C_j(W_1)$. So we only need to compare the W_1 schedule and the $\overline{W}$ schedule. We consider three possibilities of p_{l+1}: (a) $p_{l+1} \geq r_{l+2} - r_{l+1}$, (b) $p_{l+1} < r_{l+2} - r_{l+1}$ and $(l+1)(r_{l+2} - r_{l+1} - p_{l+1}) \leq p_1 - p_{l+2}$, and (c) $p_{l+1} < r_{l+2} - r_{l+1}$ and $(l+1)(r_{l+2} - r_{l+1} - p_{l+1}) > p_1 - p_{l+2}$.

(a) When $p_{l+1} \geq r_{l+2} - r_{l+1}$, the second lookahead job J_{l+2} has arrived after J_{l+1} is completed at time $r_{l+1} + p_{l+1}$ in the W_1 schedule. We compute and compare $\sum C_j(W_1)$ and $\sum C_j(\overline{W})$. We have

$$\sum C_j(W_1) = (l+2)r_{l+1} + (l+2)p_{l+1} + (l+1)\min\{p_1, p_{l+2}\} + l\min\{p_2, \max\{p_1, p_{l+2}\}\} + \cdots$$

and

$$\sum C_j(\overline{W}) = (l+2)t + (l+2)p_1 + (l+1)\min\{p_{l+1}, p_{l+2}\} + l\min\{p_2, \max\{p_{l+1}, p_{l+2}\}\} + \cdots .$$

Considering the difference, we get

$$\sum C_j(W_1) - \sum C_j(\overline{W}) = (l+2)\Delta - (p_1 - p_{l+1}) - \max\{0, p_1 - p_{l+2}\} + \max\{0, p_{l+1} - p_{l+2}\}.$$

Therefore, the algorithm chooses to wait if and only if $\sum C_j(W_1) - \sum C_j(\overline{W}) \leq 0$ if and only if $(l+2)\Delta \leq (p_1 - p_{l+1}) + \max\{0, p_1 - p_{l+2}\} - \max\{0, p_{l+1} - p_{l+2}\}$ if and only if

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$(l+2)\Delta + \min\{\max\{0, p_1 - p_{l+2}\}, (l+1)\max\{0, r_{l+2} - r_{l+1} - p_{l+1}\}\} \leq \max\{0, p_1 - p_{l+1}\} + \max\{0, p_1 - p_{l+2}\} - \max\{0, p_{l+1} - p_{l+2}\}$.

(b) When $p_{l+1} < r_{l+2} - r_{l+1}$ and $(l+1)(r_{l+2} - r_{l+1} - p_{l+1}) \leq p_1 - p_{l+2}$, the second lookahead job J_{l+2} has not arrived when J_{l+1} is completed at time $t + p_{l+1}$ in the W_1 schedule. To analyze the W_1 schedule we further classify it into the W_1/W and $W_1/\overline{W}$ schedules, with the former being the schedule in which the machine waits from t to r_{l+1}, executes J_{l+1}, and then waits again until r_{l+2}, and the latter being the schedule in which the machine waits from t to r_{l+1}, executes J_{l+1}, and then executes J_1. Because $(l+1)(r_{l+2} - r_{l+1} - p_{l+1}) \leq p_1 - p_{l+2}$, by the lookahead-1 result applied to the W_1 schedule at time $r_{l+1} + p_{l+1}$ with $Q_W = \{J_1, \ldots, J_l\}$ and $Q_{LA} = \{J_{l+2}\}$, we have $\sum C_j(W_1/W) \leq \sum C_j(W_1/\overline{W})$. So the $W_1/\overline{W}$ schedule is out of consideration. We only need to compare the W_1/W schedule and the $\overline{W}$ schedule. Recall that $p_1 > p_{l+1}$. The given $(l+1)(r_{l+2} - r_{l+1} - p_{l+1}) \leq p_1 - p_{l+2}$ also implies that $p_1 > p_{l+2}$. We have

$$\sum C_j(W_1/W) = (l+2)r_{l+1} + (l+2)p_{l+1} + (l+1)(r_{l+2} - r_{l+1} - p_{l+1}) + (l+1)p_{l+2} + lp_1 + \cdots$$

and

$$\sum C_j(\overline{W}) = (l+2)t + (l+2)p_1 + (l+1)\min\{p_{l+1}, p_{l+2}\} + l\max\{p_{l+1}, p_{l+2}\} + \cdots.$$

Considering the difference, we get

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$$\sum C_j(W_1/W) - \sum C_j(\overline{W}) = (l+2)\Delta + (l+1)(r_{l+2} - r_{l+1} - p_{l+1}) - (p_1 - p_{l+1}) - (p_1 - p_{l+2}) + \max\{0, p_{l+1} - p_{l+2}\}.$$

Therefore, the algorithm chooses to wait if and only if $\sum C_j(W_1/W) - \sum C_j(\overline{W}) \leq 0$ if and only if $(l+2)\Delta + (l+1)(r_{l+2} - r_{l+1} - p_{l+1}) \leq (p_1 - p_{l+1}) + (p_1 - p_{l+2}) - \max\{0, p_{l+1} - p_{l+2}\}$ if and only if $(l+2)\Delta + \min\{\max\{0, p_1 - p_{l+2}\}, (l+1)\max\{0, r_{l+2} - r_{l+1} - p_{l+1}\}\} \leq \max\{0, p_1 - p_{l+1}\} + \max\{0, p_1 - p_{l+2}\} - \max\{0, p_{l+1} - p_{l+2}\}$.

(c) When $p_{l+1} < r_{l+2} - r_{l+1}$ and $(l+1)(r_{l+2} - r_{l+1} - p_{l+1}) > p_1 - p_{l+2}$, this is a case symmetric to the previous case (b). So we only need to compare the $W_1/\overline{W}$ schedule and the $\overline{W}$ schedule because the given $(l+1)(r_{l+2} - r_{l+1} - p_{l+1}) > p_1 - p_{l+2}$ suggests that $\sum C_j(W_1/W) > \sum C_j(W_1/\overline{W})$. We compute and compare $\sum C_j(W_1/\overline{W})$ and $\sum C_j(\overline{W})$ We have

$$\sum C_j(W_1/\overline{W}) = (l+2)r_{l+1} + (l+2)p_{l+1} + (l+1)p_1 + l\min\{p_2, p_{l+2}\} + \cdots$$

and

$$\sum C_j(\overline{W}) = (l+2)t + (l+2)p_1 + (l+1)\min\{p_{l+1}, p_{l+2}\} + l\min\{p_2, \max\{p_{l+1}, p_{l+2}\}\} + \cdots.$$

Considering the difference, we get

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$$\sum C_j(W_1/\overline{W}) - \sum C_j(\overline{W}) = (l+2)\Delta - (p_1 - p_{l+1}) + \max\{0, p_{l+1} - p_{l+2}\}.$$

Therefore, the algorithm chooses to wait if and only if $\sum C_j(W_1/\overline{W}) - \sum C_j(\overline{W}) \leq 0$ if and only if $(l+2)\Delta \leq (p_1 - p_{l+1}) - \max\{0, p_{l+1} - p_{l+2}\}$ if and only if $(l+2)\Delta + \min\{\max\{0, p_1 - p_{l+2}\}, (l+1)\max\{0, r_{l+2} - r_{l+1} - p_{l+1}\}\} \leq \max\{0, p_1 - p_{l+1}\} + \max\{0, p_1 - p_{l+2} - \max\{0, p_{l+1} - p_{l+2}\}$. □

Now with Lemmas 4.6 and 4.7, the only case left is when $\Delta < p_1 < \Delta'$ and $r_{l+2} > r_{l+1}$ and $p_1 > p_{l+1}$. We further divide this case into the following sub-cases:

- $\Delta < p_1 < \Delta'$ and $r_{l+2} > r_{l+1}$ and $p_1 > p_{l+1}$ and $(l+1)(\Delta' - p_1) \leq p_{l+1} - p_{l+2}$
- $\Delta < p_1 < \Delta'$ and $r_{l+2} > r_{l+1}$ and $p_1 > p_{l+1}$ and $(l+1)(\Delta' - p_1) > p_{l+1} - p_{l+2}$ and $p_{l+1} \geq \Delta' - p_1$
- $\Delta < p_1 < \Delta'$ and $r_{l+2} > r_{l+1}$ and $p_1 > p_{l+1}$ and $(l+1)(\Delta' - p_1) > p_{l+1} - p_{l+2}$ and $p_{l+1} < \Delta' - p_1$

The first two sub-cases are studied in the next two lemmas while the last one requires special attention and is studied later.

Notice that because $p_1 < \Delta'$ for all of the remaining conditions, it is not beneficial to have the machine wait from t to r_{l+2} while J_1 can fit into the idle period. So the W_2 schedule where the algorithm waits until time r_{l+2} is out of consideration. Given this fact, for simplicity, we drop the subscript on the first wait decision assuming it is understood that the algorithm waits until r_{l+1}. This also implies that J_{l+1} is scheduled at time r_{l+1}.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

With this new notation, we have four schedules to consider. In the W/W schedule, the algorithm waits until time r_{l+1} and then schedules J_{l+1} when it arrives. Next, the algorithm schedules zero or more jobs from the wait queue. Specifically, from Lemma 4.6, we know that if $p_1 < \Delta$ then the algorithm schedules the job. This means that if a job can complete execution before r_{l+2}, then the algorithm should schedule that job. When no more jobs in the queue meet this criteria, the algorithm holds the machine idle until time r_{l+2}. At that time, J_{l+2} is scheduled, followed by the remaining jobs in the wait queue.

Similarly, in the $W/\overline{W}$, the algorithm waits until r_{l+1}, schedules J_{l+1} and then schedules all jobs that fit before r_{l+2}. In this case however, the algorithm schedules the next job in the queue without waiting. J_{l+2} is simply added to the queue when it arrives.

The last two schedules, $\overline{W}/W$ and $\overline{W}/\overline{W}$ are defined in analogous fashion. In general, the algorithm schedules all jobs that will complete before the next arrives and then makes its decision.

Lemma 4.8 *When $\Delta < p_1 < \Delta'$ and $r_{l+2} > r_{l+1}$ and $p_1 > p_{l+1}$ and $(l+1)(\Delta' - p_1) \leq p_{l+1} - p_{l+2}$, the algorithm chooses to wait until r_{l+1} if and only if*

$$(l+2)\Delta - (l+1)(\Delta' - p_1) \leq \max\{0, p_1 - p_{l+1}\} - \max\{0, p_{l+1} - p_{l+2}\}. \qquad (4.3)$$

Proof First, consider the the pair of non-waiting schedules, $\overline{W}/W$ and $\overline{W}/\overline{W}$. We apply the lookahead-1 result to the non-waiting schedule at time $t+p_1$ with $Q_W = \{J_{l+1}, J_2, \ldots, J_l\}$ and $Q_{LA} = \{J_{l+2}\}$. The given $(l+1)(\Delta' - p_1) \leq p_{l+1} - p_{l+2}$ ensures that $\sum C_j(\overline{W}/W) \leq \sum C_j(\overline{W}/\overline{W})$. So the $\overline{W}/\overline{W}$ schedule is out of consideration. Therefore, we must only consider the W/W, $W/\overline{W}$, and $\overline{W}/W$ schedules. Note that the given $(l+1)(\Delta' - p_1) \leq$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$p_{l+1} - p_{l+2}$ implies that $p_{l+1} > p_{l+2}$. We consider two possibilities of p_{l+1} when compared with $r_{l+2} - r_{l+1}$: (a) $p_{l+1} \geq r_{l+2} - r_{l+1}$ and (b) $p_{l+1} < r_{l+2} - r_{l+1}$.

(a) When $p_{l+1} \geq r_{l+2} - r_{l+1}$, J_{l+1} completes after the second lookahead job J_{l+2} has arrived in the $W/\overline{W}$ schedule. Thus the W/W schedule is not possible. We compute and compare $\sum C_j(W/\overline{W})$ and $\sum C_j(\overline{W}/W)$. We have

$$\sum C_j(W/\overline{W}) = (l+2)r_{l+1} + (l+2)p_{l+1} + (l+1)p_{l+2} + lp_1 + \cdots$$

and

$$\sum C_j(\overline{W}/W) = (l+2)t + (l+2)p_1 + (l+1)(\Delta' - p_1) + (l+1)p_{l+2} + lp_{l+1} + \cdots .$$

Considering the difference, we get

$$\sum C_j(W/\overline{W}) - \sum C_j(\overline{W}/W) = (l+2)\Delta - (l+1)(\Delta' - p_1) + 2p_{l+1} - 2p_1.$$

We notice that the above difference is always greater than zero because $(l+2)\Delta - (l+1)(\Delta' - p_1) + 2p_{l+1} - 2p_1 = (l+2)r_{l+1} - (l+1)r_{l+2} - t + (l-1)p_1 + 2p_{l+1} > (l+2)r_{l+1} - (l+1)r_{l+2} - t + (l+1)p_{l+1} = (l+1)(p_{l+1} + r_{l+1} - r_{l+2}) + r_{l+1} - t > 0$. The choice for the algorithm is to execute. The evaluation of inequality (4.3) also suggests the same action for the algorithm because $(l+2)\Delta - (l+1)(\Delta' - p_1) \geq (l+2)\Delta - (p_{l+1} - p_{l+2}) > \Delta - (p_{l+1} - p_{l+2}) > (p_1 - p_{l+1}) - (p_{l+1} - p_{l+2}) \geq \max\{0, p_1 - p_{l+1}\} - \max\{0, p_{l+1} - p_{l+2}\}$.

(b) When $p_{l+1} < r_{l+2} - r_{l+1}$, the second lookahead job J_{l+2} has not arrived when J_{l+1} is completed at time $r_{l+1} + p_{l+1}$ if the algorithm waits at time t. However, $\sum C_j(W/W) =$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$(r_{l+1}+p_{l+1})+(l+1)r_{l+2}+(l+1)p_{l+2}+lp_1+\cdots$ and $\sum C_j(\overline{W}/W) = (t+p_1)+(l+1)r_{l+2}+(l+1)p_{l+2}+lp_{l+1}+\cdots$, thus the difference is $\sum C_j(W/W) - \sum C_j(\overline{W}/W) = (r_{l+1}-t)+(l-1)(p_1-p_{l+1}) > 0$. So the W/W schedule is out of consideration. We only need to compare the $W/\overline{W}$ schedule and the $\overline{W}/W$ schedule. We have

$$\sum C_j(W/\overline{W}) = (l+2)r_{l+1}+(l+2)p_{l+1}+(l+1)p_1+lp_{l+2}+\cdots$$

and

$$\sum C_j(\overline{W}/W) = (l+2)t+(l+2)p_1+(l+1)(\Delta'-p_1)+(l+1)p_{l+2}+lp_{l+1}+\cdots.$$

Considering the difference, we get

$$\sum C_j(W/\overline{W}) - \sum C_j(\overline{W}/W) = (l+2)\Delta-(l+1)(\Delta'-p_1)-(p_1-p_{l+1})+(p_{l+1}-p_{l+2}).$$

Therefore, the algorithm chooses to wait if and only if $\sum C_j(W/\overline{W}) - \sum C_j(\overline{W}/W) \leq 0$ if and only if $(l+2)\Delta-(l+1)(\Delta'-p_1) \leq (p_1-p_{l+1})-(p_{l+1}-p_{l+2})$ if and only if $(l+2)\Delta-(l+1)(\Delta'-p_1) \leq \max\{0,p_1-p_{l+1}\}-\max\{0,p_{l+1}-p_{l+2}\}$. □

Lemma 4.9 *When $\Delta < p_1 < \Delta'$ and $r_{l+2} > r_{l+1}$ and $p_1 > p_{l+1}$ and $(l+1)(\Delta'-p_1) > p_{l+1}-p_{l+2}$ and $p_{l+1} \geq \Delta'-p_1$, the algorithm chooses to wait until r_{l+1} if and only if*

$$(l+2)\Delta+\min\{\max\{0,p_1-p_{l+2}\},(l+1)\max\{0,r_{l+2}-r_{l+1}-p_{l+1}\}\}$$
$$\leq \max\{0,p_1-p_{l+1}\}+\max\{0,p_1-p_{l+2}\}. \quad (4.4)$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Proof As with lemma 4.8, we first rule out one possible schedule. The given $(l+1)(\Delta' - p_1) > p_{l+1} - p_{l+2}$ ensures that $\sum C_j(\overline{W}/W) > \sum C_j(\overline{W}/\overline{W})$ due to the lookahead result applied to the $\overline{W}$ schedule at time $t + p_1$ with $Q_W = \{J_{l+1}, J_2, \ldots, J_l\}$ and $Q_{LA} = \{J_{l+2}\}$. So the $\overline{W}/W$ schedule is out of consideration. We only need to compare the W/W, $W/\overline{W}$, and $\overline{W}/\overline{W}$ schedules. We consider three possibilities: (a) $p_{l+1} \geq r_{l+2} - r_{l+1}$, (b) $p_{l+1} < r_{l+2} - r_{l+1}$ and $(l+1)(r_{l+2} - r_{l+1} - p_{l+1}) \leq p_1 - p_{l+2}$, and (c) $p_{l+1} < r_{l+2} - r_{l+1}$ and $(l+1)(r_{l+2} - r_{l+1} - p_{l+1}) > p_1 - p_{l+2}$.

(a) When $p_{l+1} \geq r_{l+2} - r_{l+1}$, J_{l+1} is completed after the second lookahead job J_{l+2} has arrived in the $W/\overline{W}$ schedule. Therefore the W/W schedule is not possible. We compute and compare $\sum C_j(W/\overline{W})$ and $\sum C_j(\overline{W}/\overline{W})$. We have

$$\sum C_j(W/\overline{W}) = (l+2)r_{l+1} + (l+2)p_{l+1} + (l+1)\min\{p_1, p_{l+2}\} + l\min\{p_2, \max\{p_1, p_{l+2}\}\} + \cdots$$

and

$$\sum C_j(\overline{W}/\overline{W}) = (l+2)t + (l+2)p_1 + (l+1)p_{l+1} + l\min\{p_2, p_{l+2}\} + \cdots.$$

Considering the difference, we get

$$\sum C_j(W/\overline{W}) - \sum C_j(\overline{W}/\overline{W}) = (l+2)\Delta - (p_1 - p_{l+1}) - \max\{0, p_1 - p_{l+2}\}.$$

Therefore, the algorithm chooses to wait if and only if $\sum C_j(W/\overline{W}) - \sum C_j(\overline{W}/\overline{W}) \leq 0$ if and only if $(l+2)\Delta \leq (p_1 - p_{l+1}) + \max\{0, p_1 - p_{l+2}\}$ if and only if $(l+2)\Delta +$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$\min\{\max\{0, p_1 - p_{l+2}\}, (l+1)\max\{0, r_{l+2} - r_{l+1} - p_{l+1}\}\} \leq \max\{0, p_1 - p_{l+1}\} + \max\{0, p_1 - p_{l+2}\}$.

(b) When $p_{l+1} < r_{l+2} - r_{l+1}$ and $(l+1)(r_{l+2} - r_{l+1} - p_{l+1}) \leq p_1 - p_{l+2}$, the second lookahead job J_{l+2} has not arrived when J_{l+1} is completed at time $r_{l+1} + p_{l+1}$ when the algorithm decides to wait for J_{l+1}. Because $(l+1)(r_{l+2} - r_{l+1} - p_{l+1}) \leq p_1 - p_{l+2}$, $\sum C_j(W/W) \leq \sum C_j(W/\overline{W})$ due to the lookahead-1 result applied to the $W/\overline{W}$ schedule at time $r_{l+1} + p_{l+1}$ with $Q_W = \{J_1, \ldots, J_l\}$ and $Q_{LA} = \{J_{l+2}\}$. So the $W/\overline{W}$ schedule is out of consideration. We only need to compare the W/W schedule and the $\overline{W}/\overline{W}$ schedule. Note that the given $(l+1)(r_{l+2} - r_{l+1} - p_{l+1}) \leq p_1 - p_{l+2}$ also implies that $p_{l+2} \leq p_1$. We have

$$\sum C_j(W/W) = (l+2)r_{l+1} + (l+2)p_{l+1} + (l+1)(r_{l+2} - r_{l+1} - p_{l+1}) + (l+1)p_{l+2} + lp_1 + \cdots$$

and

$$\sum C_j(\overline{W}/\overline{W}) = (l+2)t + (l+2)p_1 + (l+1)p_{l+1} + lp_{l+2} + \cdots.$$

Considering the difference, we get

$$\sum C_j(W/W) - \sum C_j(\overline{W}/\overline{W}) = (l+2)\Delta + (l+1)(r_{l+2} - r_{l+1} - p_{l+1}) - (p_1 - p_{l+1}) - (p_1 - p_{l+2}).$$

Therefore, the algorithm chooses to wait if and only if $\sum C_j(W/W) - \sum C_j(\overline{W}/\overline{W}) \leq 0$ if and only if $(l+2)\Delta + (l+1)(r_{l+2} - r_{l+1} - p_{l+1}) \leq (p_1 - p_{l+1}) + (p_1 - p_{l+2})$ if and only if

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$(l+2)\Delta + \min\{\max\{0, p_1 - p_{l+2}\}, (l+1)\max\{0, r_{l+2} - r_{l+1} - p_{l+1}\}\} \leq \max\{0, p_1 - p_{l+1}\} +$ $\max\{0, p_1 - p_{l+2}\}$.

(c) When $p_{l+1} < r_{l+2} - r_{l+1}$ and $(l+1)(r_{l+2} - r_{l+1} - p_{l+1}) > p_1 - p_{l+2}$, this is a case symmetric to the previous case (b). We only need to compare the $W/\overline{W}$ schedule and the $\overline{W}/\overline{W}$ schedule because the given $(l+1)(r_{l+2} - r_{l+1} - p_{l+1}) > p_1 - p_{l+2}$ suggests that $\sum C_j(W/W) > \sum C_j(W/\overline{W})$. We have

$$\sum C_j(W/\overline{W}) = (l+2)r_{l+1} + (l+2)p_{l+1} + (l+1)p_1 + \cdots$$

and

$$\sum C_j(\overline{W}/\overline{W}) = (l+2)t + (l+2)p_1 + (l+1)p_{l+1} + \cdots.$$

Considering the difference, we get

$$\sum C_j(W/\overline{W}) - \sum C_j(\overline{W}/\overline{W}) = (l+2)\Delta - (p_1 - p_{l+1}).$$

Therefore, the algorithm chooses to wait if and only if $\sum C_j(W/\overline{W}) - \sum C_j(\overline{W}/\overline{W}) \leq 0$ if and only if $(l+2)\Delta \leq p_1 - p_{l+2}$ if and only if $(l+2)\Delta + \min\{\max\{0, p_1 - p_{l+2}\}, (l+1)\max\{0, r_{l+2} - r_{l+1} - p_{l+1}\}\} \leq \max\{0, p_1 - p_{l+1}\} + \max\{0, p_1 - p_{l+2}\}$. □

For all the cases studied in Lemmas 4.6 – 4.9, the algorithm's decision whether to wait or to execute is based on the evaluation of an inequality, (4.1), (4.2), (4.3), or (4.4), which can be done in constant time. However, for the final case the time needed is no longer constant. Recall that the final unsolved case is $\Delta < p_1 < \Delta'$ and $r_{l+2} > r_{l+1}$ and $p_1 > p_{l+1}$ and $(l+1)(\Delta' - p_1) > p_{l+1} - p_{l+2}$ and $p_{l+1} < \Delta' - p_1$. Examining the conditions in this

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

case, there are no upper bounds imposed on Δ', or equivalently, r_{l+2}, suggesting that the second lookahead job J_{l+2} may arrive at an arbitrarily late time, thus allowing multiple jobs to be scheduled in the gap Δ'. Because of this, the algorithm requires linear time in the worst case to make each decision.

As with Lemmas 4.8 and 4.9, we do not need to consider the W_2 schedule where we wait until time r_{l+2}. Here especially, such a decision simply does not make sense. Referring again to Lemma 4.6, waiting only makes sense if the smallest (first) job in the queue cannot complete execution before the arrival of the next job. In this case, our former definition of the schedules W/W, $W/\overline{W}$, $\overline{W}/W$ and $\overline{W}/\overline{W}$ are well defined. The algorithm waits or schedules (the first W or $\overline{W}$) and then the next important scheduling decision (the second W or $\overline{W}$) occurs after any intermediate jobs are scheduled.

First, we define notation for the number of jobs that are scheduled between the two "important" decisions. In the waiting schedule, the machine is idle from t to r_{l+1}, and then executes J_{l+1} at r_{l+1}. Note that $r_{l+1}+p_{l+1} < r_{l+1}+(\Delta'-p_1) < r_{l+1}+\Delta'-\Delta = r_{l+2}$. From $r_{l+1}+p_{l+1}$ to r_{l+2}, there may be jobs $J_1, J_2, \ldots$ that fit into the period. Let S_W be the largest set of all jobs, $J_1, \ldots, J_h$, that can be executed and completed in the period from $r_{l+1}+p_{l+1}$ to r_{l+2}. Similarly in the non-waiting schedule, the machine executes J_1 at t and then J_{l+1} at $t+p_1$. Note that $t+p_1+p_{l+1} < t+p_1+(\Delta'-p_1) = t+\Delta' = r_{l+2}$. From $t+p_1+p_{l+1}$ to r_{l+2}, there may be jobs, $J_2, J_3, \ldots$ that fit into the period. Let $S_{\overline{W}}$ be the largest set of all jobs, $J_2, \ldots, J_g$, that can be executed and completed in the period from $t+p_1+p_{l+1}$ to r_{l+2}. We notice that sets S_W and $S_{\overline{W}}$ are closely related in that either $g = h+1$ or $g = h$, and furthermore we have the following possibilities.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- When $g = h+1$, $S_W = \{J_1, \ldots, J_h\}$ and $S_{\overline{W}} = \{J_2, \ldots, J_{h+1}\}$ with $h \geq 0$. (When $h = 0$, $S_W = S_{\overline{W}} = \emptyset$.)
- When $g = h$, $S_W = \{J_1, \ldots, J_h\}$ and $S_{\overline{W}} = \{J_2, \ldots, J_h\}$ with $h \geq 1$. (When $h = 1$, $S_{\overline{W}} = \emptyset$.)

Therefore, our final unsettled case, $\Delta < p_1 < \Delta'$ and $r_{l+2} > r_{l+1}$ and $p_1 > p_{l+1}$ and $(l+1)(\Delta' - p_1) > p_{l+1} - p_{l+2}$ and $p_{l+1} < \Delta' - p_1$ is further divided into the following two sub-cases, each of which is studied separately.

- $\Delta < p_1 < \Delta'$ and $r_{l+2} > r_{l+1}$ and $p_1 > p_{l+1}$ and $(l+1)(\Delta' - p_1) > p_{l+1} - p_{l+2}$ and $p_{l+1} < \Delta' - p_1$ and $g = h+1$
- $\Delta < p_1 < \Delta'$ and $r_{l+2} > r_{l+1}$ and $p_1 > p_{l+1}$ and $(l+1)(\Delta' - p_1) > p_{l+1} - p_{l+2}$ and $p_{l+1} < \Delta' - p_1$ and $g = h$

Lemma 4.10 *When $\Delta < p_1 < \Delta'$ and $r_{l+2} > r_{l+1}$ and $p_1 > p_{l+1}$ and $(l+1)(\Delta' - p_1) > p_{l+1} - p_{l+2}$ and $p_{l+1} < \Delta' - p_1$ and $g = h+1$, the algorithm chooses to wait until r_{l+1} if and only if*

$$(h+2)\Delta + (l-h)(\Delta + p_{l+1} + \sum_{j=1}^{h+1} p_j - \Delta') + \max\{(l-h)(\Delta' - p_{l+1} - \sum_{j=1}^{h+1} p_j), \\ p_{h+2} - p_{l+2}\} \leq \max\{0, p_1 - p_{l+1}\} + \max\{0, p_{h+2} - p_{l+2}\}, \quad (4.5)$$

where h and g are as defined earlier based on the contents in sets S_W and $S_{\overline{W}}$.

Proof We first argue that the W/W schedule yields a total completion time larger than that in the $\overline{W}/W$ schedule and, therefore, is out of consideration because

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$$\sum C_j(W/W) \geq (h+1)r_{l+1} + (h+1)p_{l+1} + hp_1 + \cdots + p_h + (r_{l+2} + p_{h+1}) + \cdots$$

and

$$\sum C_j(\overline{W}/W) = (h+2)t + (h+2)p_1 + (h+1)p_{l+1} + hp_2 + \cdots + p_{h+1} + \cdots ,$$

where the last "$\cdots$" in each formula represents the total completion time of the jobs $J_{l+2}, J_{h+2}, \ldots, J_l$ using the shortest-job-first order starting at time r_{l+2}. Considering the difference, we get

$$\begin{aligned}
&\sum C_j(W/W) - \sum C_j(\overline{W}/W) \\
&\geq (h+1)\Delta - t - 2p_1 - p_2 - \cdots - p_h + r_{l+2} \\
&= (h+1)\Delta - 2p_1 - p_2 - \cdots - p_h + \Delta' \\
&\geq (h+1)\Delta - 2p_1 - p_2 - \cdots - p_h + p_{l+1} + p_1 + \cdots + p_{h+1} \\
&= (h+1)\Delta + p_{l+1} + p_{h+1} - p_1 \\
&\geq (h+1)\Delta + p_{l+1} \\
&> 0.
\end{aligned}$$

So we only need to compare the $W/\overline{W}$ schedule with one of the $\overline{W}/W$ and the $\overline{W}/\overline{W}$ schedules, depending on which yields a smaller total completion time. We consider two

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

possibilities: (a) $(l-h)(\Delta' - p_{l+1} - \sum_{j=1}^{h+1} p_j) \leq p_{h+2} - p_{l+2}$ and (b) $(l-h)(\Delta' - p_{l+1} - \sum_{j=1}^{h+1} p_j) > p_{h+2} - p_{l+2}$.

(a) When $(l-h)(\Delta' - p_{l+1} - \sum_{j=1}^{h+1} p_j) \leq p_{h+2} - p_{l+2}$, the $\overline{W}/W$ schedule has a smaller total completion time than the $\overline{W}/\overline{W}$ schedule according to the lookahead-1 result applied to time $t + p_{l+1} + \sum_{j=1}^{h+1} p_j$ in the non-waiting schedule with $Q_W = \{J_{h+2}, \ldots, J_l\}$ and $Q_{LA} = \{J_{l+2}\}$. So the $\overline{W}/\overline{W}$ schedule is out of consideration. Note that the given$(l-h)(\Delta' - p_{l+1} - \sum_{j=1}^{h+1} p_j) \leq p_{h+2} - p_{l+2}$ also implies that $p_{h+2} \geq p_{l+2}$. We compute and compare $\sum C_j(W/\overline{W})$ and $\sum C_j(\overline{W}/W)$. We have

$$\sum C_j(W/\overline{W}) = (h+2)r_{l+1} + (h+2)p_{l+1} + (h+1)p_1 + hp_2 + \cdots + p_{h+1}$$
$$+(l-h)(r_{l+1} + p_{l+1} + \sum_{j=1}^{h+1} p_j) + \cdots$$

and

$$\sum C_j(\overline{W}/W)$$
$$= (h+2)t + (h+2)p_1 + (h+1)p_{l+1} + hp_2 + \cdots + p_{h+1} + (l-h)r_{l+2} + \cdots,$$

where the last "$\cdots$" in each formula represents the total completion time of the jobs $J_{l+2}, J_{h+2}, \ldots, J_l$ using the shortest-job-first order starting at time 0. Considering the difference, we get

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$$\begin{aligned}
&\sum C_j(W/\overline{W}) - \sum C_j(\overline{W}/W) \\
&= (h+2)\Delta - (p_1 - p_{l+1}) + (l-h)(r_{l+1} + p_{l+1} + \sum_{j=1}^{h+1} p_j - r_{l+2}) \\
&= (h+2)\Delta - (p_1 - p_{l+1}) + (l-h)(\Delta + p_{l+1} + \sum_{j=1}^{h+1} p_j - \Delta').
\end{aligned}$$

Therefore, the algorithm chooses to wait if and only if $\sum C_j(W/\overline{W}) - \sum C_j(\overline{W}/W)$ ≤ 0 if and only if $(h+2)\Delta + (l-h)(\Delta + p_{l+1} + \sum_{j=1}^{h+1} p_j - \Delta') \leq p_1 - p_{l+1}$ if and only if $(h+2)\Delta + (l-h)(\Delta + p_{l+1} + \sum_{j=1}^{h+1} p_j - \Delta') + \max\{(l-h)(\Delta' - p_{l+1} - \sum_{j=1}^{h+1} p_j), p_{h+2} - p_{l+2}\} \leq \max\{0, p_1 - p_{l+1}\} + \max\{0, p_{h+2} - p_{l+2}\}$.

(b) When $(l-h)(\Delta' - p_{l+1} - \sum_{j=1}^{h+1} p_j) > p_{h+2} - p_{l+2}$, the $\overline{W}/W$ schedule has a larger total completion time than the $\overline{W}/\overline{W}$ schedule according to the lookahead-1 result applied to time $t + p_{l+1} + \sum_{j=1}^{h+1} p_j$ in the non-waiting schedule with $Q_W = \{J_{h+2}, \ldots, J_l\}$ and $Q_{LA} = \{J_{l+2}\}$. So the $\overline{W}/W$ schedule is out of consideration. We compute and compare $\sum C_j(W/\overline{W})$ and $\sum C_j(\overline{W}/\overline{W})$. We have

$$\begin{aligned}
&\sum C_j(W/\overline{W}) \\
&= (l+2)r_{l+1} + (l+2)p_{l+1} + (l+1)p_1 + \cdots + (l-h+1)p_{h+1} + (l-h)\min\{p_{h+2}, p_{l+2}\} \\
&+(l-h-1)\min\{p_{h+3}, \max\{p_{h+2}, p_{l+2}\}\} + \cdots
\end{aligned}$$

and

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$$\sum C_j(\overline{W}/\overline{W})$$
$$= (l+2)t + (l+2)p_1 + (l+1)p_{l+1} + lp_2 + \cdots + (l-h)p_{h+2}$$
$$+(l-h-1)\min\{p_{h+3}, p_{l+2}\} + \cdots ,$$

Considering the difference, we get

$$\sum C_j(W/\overline{W}) - \sum C_j(\overline{W}/\overline{W})$$
$$= (l+2)\Delta - (p_1 - p_{l+1}) - \max\{0, p_{h+2} - p_{l+2}\}.$$

Therefore, the algorithm chooses to wait if and only if $\sum C_j(W/\overline{W}) - \sum C_j(\overline{W}/\overline{W}) \leq 0$ if and only if $(l+2)\Delta \leq (p_1 - p_{l+1}) + \max\{0, p_{h+2} - p_{l+2}\}$ if and only if $(h+2)\Delta + (l-h)(\Delta + p_{l+1} + \sum_{j=1}^{h+1} p_j - \Delta') + \max\{(l-h)(\Delta' - p_{l+1} - \sum_{j=1}^{h+1} p_j), p_{h+2} - p_{l+2}\} \leq \max\{0, p_1 - p_{l+1}\} + \max\{0, p_{h+2} - p_{l+2}\}$. □

Lemma 4.11 *When $\Delta < p_1 < \Delta'$ and $r_{l+2} > r_{l+1}$ and $p_1 > p_{l+1}$ and $(l+1)(\Delta' - p_1) > p_{l+1} - p_{l+2}$ and $p_{l+1} < \Delta' - p_1$ and $g = h$, the algorithm chooses to wait until r_{l+1} if and only if*

$$(l+2)\Delta + \min\{(l-h+1)(\Delta' - \Delta - p_{l+1} - \sum_{j=1}^{h} p_j), p_{h+1} - p_{l+2}\}$$
$$- \min\{(l-h+1)(\Delta' - p_{l+1} - \sum_{j=1}^{h} p_j), p_{h+1} - p_{l+2}\}$$
$$\leq \max\{0, p_1 - p_{l+1}\}, \tag{4.6}$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

where h and g are as defined earlier based on the contents in sets S_W and $S_{\overline{W}}$.

Proof Applying the lookahead-1 result to time $r_{l+1} + p_{l+1} + \sum_{j=1}^{h} p_j$ in the waiting schedule with $Q_W = \{J_{h+1}, \dots, J_l\}$ and $Q_{LA} = \{J_{l+2}\}$, we have that $\sum C_j(W/W) \leq \sum C_j(W/\overline{W})$ if and only if $(l - h + 1)(\Delta' - \Delta - p_{l+1} - \sum_{j=1}^{h} p_j) \leq p_{h+1} - p_{l+2}$. Similarly, applying the lookahead-1 result to time $t + p_{l+1} + \sum_{j=1}^{h} p_j$ in the non-waiting schedule with $Q_W = \{J_{h+1}, \dots, J_l\}$ and $Q_{LA} = \{J_{l+2}\}$, we have that $\sum C_j(\overline{W}/W) \leq \sum C_j(\overline{W}/\overline{W})$ if and only if $(l - h + 1)(\Delta' - p_{l+1} - \sum_{j=1}^{h} p_j) \leq p_{h+1} - p_{l+2}$. Note that $(l - h + 1)(\Delta' - \Delta - p_{l+1} - \sum_{j=1}^{h} p_j) < (l - h + 1)(\Delta' - p_{l+1} - \sum_{j=1}^{h} p_j)$. We have the following three possibilities: (a) $(l - h + 1)(\Delta' - p_{l+1} - \sum_{j=1}^{h} p_j) \leq p_{h+1} - p_{l+2}$, (b) $(l - h + 1)(\Delta' - \Delta - p_{l+1} - \sum_{j=1}^{h} p_j) > p_{h+1} - p_{l+2}$, and (c) $(l - h + 1)(\Delta' - \Delta - p_{l+1} - \sum_{j=1}^{h} p_j) \leq p_{h+1} - p_{l+2}$ and $(l - h + 1)(\Delta' - p_{l+1} - \sum_{j=1}^{h} p_j) > p_{h+1} - p_{l+2}$.

(a) When $(l - h + 1)(\Delta' - p_{l+1} - \sum_{j=1}^{h} p_j) \leq p_{h+1} - p_{l+2}$, which also implies that $(l - h + 1)(\Delta' - \Delta - p_{l+1} - \sum_{j=1}^{h} p_j) < p_{h+1} - p_{l+2}$, the $W/\overline{W}$ and the $\overline{W}/\overline{W}$ are out of consideration. We only need to compute and compare $\sum C_j(W/W)$ and $\sum C_j(\overline{W}/W)$. We have

$$\sum C_j(W/W) = (h+1)r_{l+1} + (h+1)p_{l+1} + hp_1 + \cdots + p_h + \cdots$$

and

$$\sum C_j(\overline{W}/W) = (h+1)t + (h+1)p_1 + hp_{l+1} + \cdots + p_h + \cdots,$$

where the last "$\cdots$" in each formula represents the total completion time of the jobs $J_{l+2}, J_{h+1}, \dots, J_l$ using the shortest-job-first order starting at time r_{l+2}. Considering the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

difference, we get

$$\sum C_j(W/W) - \sum C_j(\overline{W}/W) = (h+1)\Delta - (p_1 - p_{l+1}).$$

Therefore, the algorithm chooses to wait if and only if $\sum C_j(W/W) - \sum C_j(\overline{W}/W) \leq 0$ if and only if $(h+1)\Delta \leq p_1 - p_{l+1}$ if and only if $(l+2)\Delta + \min\{(l-h+1)(\Delta' - \Delta - p_{l+1} - \sum_{j=1}^{h} p_j), p_{h+1} - p_{l+2}\} - \min\{(l-h+1)(\Delta' - p_{l+1} - \sum_{j=1}^{h} p_j), p_{h+1} - p_{l+2}\} \leq \max\{0, p_1 - p_{l+1}\}$.

(b) When $(l-h+1)(\Delta' - \Delta - p_{l+1} - \sum_{j=1}^{h} p_j) > p_{h+1} - p_{l+2}$, which also implies that $(l-h+1)(\Delta' - p_{l+1} - \sum_{j=1}^{h} p_j) > p_{h+1} - p_{l+2}$, the W/W and the $\overline{W}/W$ schedules are out of consideration. We only need to compute and compare $\sum C_j(W/\overline{W})$ and $\sum C_j(\overline{W}/\overline{W})$. We have,

$$\sum C_j(W/\overline{W}) = (l+2)r_{l+1} + (l+2)p_{l+1} + (l+1)p_1 + \cdots$$

and

$$\sum C_j(\overline{W}/\overline{W}) = (l+2)t + (l+2)p_1 + (l+1)p_{l+1} + \cdots.$$

Considering the difference, we get

$$\sum C_j(W/\overline{W}) - \sum C_j(\overline{W}/\overline{W}) = (l+2)\Delta - (p_1 - p_{l+1}).$$

Therefore, the algorithm chooses to wait if and only if $\sum C_j(W/\overline{W}) - \sum C_j(\overline{W}/\overline{W}) \leq 0$ if and only if $(l+2)\Delta \leq p_1 - p_{l+1}$ if and only if $(l+2)\Delta + \min\{(l-h+1)(\Delta' - \Delta - p_{l+1} - \sum_{j=1}^{h} p_j), p_{h+1} - p_{l+2}\} - \min\{(l-h+1)(\Delta' - p_{l+1} - \sum_{j=1}^{h} p_j), p_{h+1} - p_{l+2}\} \leq \max\{0, p_1 - p_{l+1}\}$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(c) When $(l-h+1)(\Delta'-\Delta-p_{l+1}-\sum_{j=1}^{h}p_j) \leq p_{h+1}-p_{l+2}$ and $(l-h+1)(\Delta'-p_{l+1}-\sum_{j=1}^{h}p_j) > p_{h+1}-p_{l+2}$, the $W/\overline{W}$ and the $\overline{W}/W$ schedules are out of consideration. Note that the above inequalities imply that $p_{h+1} \geq p_{l+2}$. We only need to compute and compare $\sum C_j(W/W)$ and $\sum C_j(\overline{W}/\overline{W})$. We have

$$\begin{aligned}
&\sum C_j(W/W) \\
&= (l+2)r_{l+1} + (l+2)p_{l+1} + (l+1)p_1 + \cdots + (l-h+1)(\Delta' - \Delta - p_{l+1} - \sum_{j=1}^{h} p_j) \\
&+(l-h+1)p_{l+2} + (l-h)p_{h+1} + + \cdots
\end{aligned}$$

and

$$\begin{aligned}
&\sum C_j(\overline{W}/\overline{W}) \\
&= (l+2)t + (l+2)p_1 + (l+1)p_{l+1} + \cdots + (l-h+1)p_{h+1} + (l-h)p_{l+2} + \cdots .
\end{aligned}$$

Considering the difference, we get

$$\begin{aligned}
&\sum C_j(W/W) - \sum C_j(\overline{W}/\overline{W}) \\
&= (l+2)\Delta - (p_1 - p_{l+1}) + (l-h+1)(\Delta' - \Delta - p_{l+1} - \sum_{j=1}^{h} p_j) - (p_{h+1} - p_{l+2}).
\end{aligned}$$

Therefore, the algorithm chooses to wait if and only if $\sum C_j(W/W) - \sum C_j(\overline{W}/\overline{W}) \leq 0$ if and only if $(l+2)\Delta + (l-h+1)(\Delta' - \Delta - p_{l+1} - \sum_{j=1}^{h} p_j) \leq (p_1 - p_{l+1}) + (p_{h+1} - p_{l+2})$ if and only if $(l+2)\Delta + \min\{(l-h+1)(\Delta' - \Delta - p_{l+1} - \sum_{j=1}^{h} p_j), p_{h+1} - p_{l+2}\} - \min\{(l-h+1)(\Delta' - p_{l+1} - \sum_{j=1}^{h} p_j), p_{h+1} - p_{l+2}\} \leq \max\{0, p_1 - p_{l+1}\}$. □

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.3.2 LA2 Algorithm Time Complexity

As shown in Lemmas 4.10 and 4.11, the running time for a LA2 algorithm is linear. This occurs because the algorithm must know when r_{l+2} occurs relative to the previous completion time. If the arrival occurs close to a completion time, waiting may be beneficial. However, given the sets of waiting jobs and lookahead jobs, this fact can only be determined by direct calculation, considering the jobs in order. Therefore, although there are many cases where the LA2 algorithm makes decisions in constant time, there exists instances requiring linear time.

4.4 Running Time of LAk Algorithm

In this chapter, we considered the implementation of two lookahead algorithms. In both cases, the worst-case running time was linear, although we argued that the actual running time would be near-constant for the Lookahead-1 algorithm for two machines. In this section, we consider the running time of a Lookahead-k algorithm for a single machine.

Our approach to pick the shortest job in the wait queue is based on the problem $1||\sum C_j$ where the n jobs are available at time 0. Conway [28] showed that the optimal schedule is produced by sorting the jobs and scheduling them from shortest to longest (SJF). Note that this solution has no idle periods because all the jobs are available at once.

Unfortunately, the SJF algorithm does not produce the optimal schedule when jobs arrive over time. There can be time periods in the optimal schedule where the machine is idle even though one or more jobs are available, and the SJF algorithm, called a conservative approach, does not allow this type of idleness. A lookahead algorithm creates a

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

better schedule by finding some instances when conservative decisions are not optimal. The amount of improvement increases as the size of the lookahead queue grows.

The limit of this growth is when the algorithm knows about every job to schedule, and as $k \to n$, we have the offline version of $1|r_j|\sum C_j$. Unfortunately, this problem has been shown to be strongly NP-hard [72], and consequently, any method that guarantees finding the local optimal requires exponential time (unless $\mathcal{P} = \mathcal{NP}$).

Thus, there is a tension between quality of schedule and running time. If we use an algorithm with a large lookahead, it creates very good schedules, but the execution time of the algorithm is large. On the other hand, if we use a small amount of lookahead, the algorithm runs quickly, but the quality of the schedule is reduced. As we will discuss further in our simulation study of lookahead algorithms, Chapter 6, the amount of improvement actually tapers off as the size of the lookahead grows. Lookahead-1 (LA1) results in a significant improvement over SJF, whereas Lookahead-2 is only slightly better than LA1. We therefore conclude that a lookahead of one is the appropriate balance between the conflicting metrics of execution time and schedule quality.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 5

Algorithm Analysis

5.1 Introduction

Beginning in this chapter, we turn from the design of efficient algorithms to a discussion of the effectiveness of our algorithms. We study algorithms in two ways. In this chapter, we use mathematical analysis to measure the worst-case performance, and in the next chapter, we use simulation to consider the average case. For both approaches, we show that our algorithm produces schedules that are significantly better than schedules from an equivalent algorithm without lookahead.

Our approach in this chapter is a technique called competitive analysis, which produces a ratio describing how far from optimal a solution is in the worst case. Formally, let A be an algorithm of interest. Further, let I be an instance for the problem and let $A(I)$ and $\mathrm{OPT}(I)$ be the values of the objective function given by the algorithm and the optimal algorithm, respectively. Then the algorithm A is said to be c-competitive if for all I,

$$A(I) \leq c \cdot \mathrm{OPT}(I)$$

Notice that this approach is a shift in the way we think about an algorithm. When we

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

considered the running time in Chapter 4, we were interested in how long it would take to make each local-optimal decision. Here, we are interested in cumulative performance of the algorithm, not the quality of individual decisions.

We apply this technique to algorithms for the general problem $P|r_j|\sum C_j$, where we have n jobs to schedule on m machines. Each job has an arrival time r_j and a processing time p_j, and the goal is to schedule all of the jobs while minimizing the sum of completion times. This is the same problem studied in Chapter 4 except we have generalized the number of machines to m.

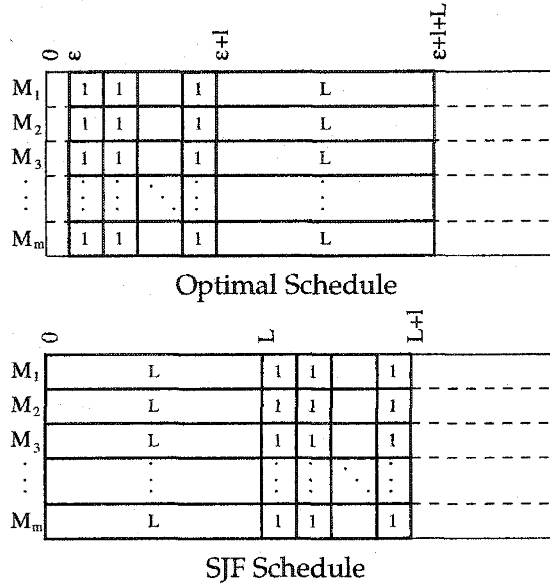
For this problem, we study three algorithms, the Shortest Job First (SJF) algorithm, the Lookahead-1 (LA1) algorithm, and the Lookahead-k (LAk) algorithm, each based on the model described in Chapter 3. In our model, we described a three step method to make scheduling decisions. The first two steps are to assume that no other jobs will arrive and then to calculate the optimal schedule of the jobs in the wait queue and lookahead queue. In the third step, if the algorithm decides to schedule it always chooses the shortest job in the wait queue. Because this is the same rule that the SJF algorithm uses, we consider the SJF algorithm to be the Lookahead-0 (LA0) algorithm.

Using this alternative view of the SJF algorithm, the difference between any two algorithms is the size of the lookahead queue, Q_{LA}. For the SJF algorithm, the size is zero; for the LA1 algorithm, the size is one; and for LAk, the size is k. Thus, for each algorithm, the wait queue, Q_W, holds jobs that have arrived but have not been scheduled and the lookahead queue, Q_{LA}, contains zero or more jobs that arrive in the future. The algorithm makes scheduling decisions using the information in these two queues.

Within this framework, we consider the performance of each algorithm. In section

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 5.1: In Theorem 5.1, the optimal schedule has all of the machines idle until time ϵ and then executes the short jobs followed by the long jobs. In the SJF schedule, the long jobs are executed immediately, and then the short jobs.



5.2, we give a proof of the lower bound on the competitive ratio for the SJF algorithm. Likewise, in Section 5.3 we give a lower bound proof for the LA1 algorithm and in Section 5.4, a proof for the LAk algorithm. Finally, in Section 5.5, we discuss the significance of these ratios.

5.2 Lower Bound for SJF

Theorem 5.1 *The competitive ratio for SJF is at least* $\frac{n}{m}$.

Proof: Consider the instance where m long jobs with processing time L arrive at time 0 and $l \cdot m$ short jobs with processing time 1 arrive at time ϵ. Note that $n = m + l \cdot m = (l+1) \cdot m$. In the optimal schedule OPT, all m machines wait from time 0 to time ϵ. Then, all $l \cdot m$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

short jobs are executed followed by the m long jobs. See Figure 5.1. Let C_{OPT} be the total completion time in the optimal schedule. We have

$$\begin{aligned} C_{\text{OPT}} &= m((\epsilon+1)+(\epsilon+2)+\ldots+(\epsilon+l)+(\epsilon+l+L)) \\ &= m\left(L+\frac{1}{2}l^2+\left(\frac{3}{2}+\epsilon\right)l+\epsilon\right). \end{aligned}$$

In the SJF schedule, all m long jobs are executed before any short jobs. See Figure 5.1. Let C_{SJF} be the total completion time in the SJF schedule. We have

$$\begin{aligned} C_{\text{SJF}} &= m(L+(L+1)+(L+2)+\ldots+(L+l)) \\ &= m\left((l+1)L+\frac{1}{2}l^2+\frac{1}{2}l\right). \end{aligned}$$

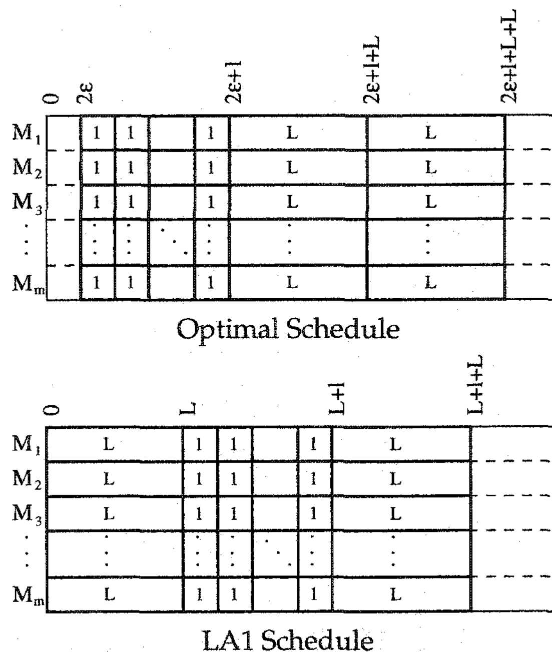
Therefore, the ratio is

$$\begin{aligned} \frac{C_{\text{SJF}}}{C_{\text{OPT}}} &= \frac{m\left((l+1)L+\frac{1}{2}l^2+\frac{1}{2}l\right)}{m\left(L+\frac{1}{2}l^2+\left(\frac{3}{2}+\epsilon\right)l+\epsilon\right)} \\ &= \frac{\frac{n}{m}L+\frac{1}{2}l^2+\frac{1}{2}l}{L+\frac{1}{2}l^2+\left(\frac{3}{2}+\epsilon\right)l+\epsilon} \\ &\to \frac{n}{m} \qquad \text{as } L\to\infty. \end{aligned}$$

■

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 5.2: In Theorem 5.2, the optimal schedule has all of the machines idle until time 2ϵ and then schedules the short jobs followed by both sets of long jobs. In the LA1 schedule, the first set of long jobs execute immediately, then the short jobs, and finally, the second set of long jobs.



5.3 Lower Bound for LA1

Theorem 5.2 *The competitive ratio for LA1 is at least* $\frac{1}{3}\left(\frac{n}{m}+1\right)$.

Proof: Consider the instance where m long jobs with processing time L arrive at time 0 followed by another m long jobs with processing time L arriving at time ϵ, and finally, $l \cdot m$ short jobs with processing time 1 arriving at time 2ϵ. Note that $n = m+m+l \cdot m = (l+2)m$.

In the optimal schedule, all m machines wait from time 0 until time 2ϵ, and then execute all the short jobs before executing any long jobs. See Figure 5.2. Let C_{OPT} be the total completion time in the optimal schedule. We have

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$$\begin{aligned} C_{\mathrm{OPT}} &= m((2\epsilon+1)+(2\epsilon+2)+\ldots+(2\epsilon+l)+(2\epsilon+l+L)+(2\epsilon+l+L+L)) \\ &= m\left(3L+\frac{1}{2}l^2+\left(\frac{5}{2}+2\epsilon\right)l+4\epsilon\right). \end{aligned}$$

In the LA1 schedule, all m long jobs with arrival time 0 are executed before any short jobs. The second batch of long jobs are executed last. See Figure 5.2. Let C_{LA1} be the total completion time of the LA1 schedule. We have

$$\begin{aligned} C_{\mathrm{LA1}} &= m(L+(L+1)+(L+2)+\ldots+(L+l)+(L+l+L)) \\ &= m\left((l+3)L+\frac{1}{2}l^2+\frac{3}{2}l\right). \end{aligned}$$

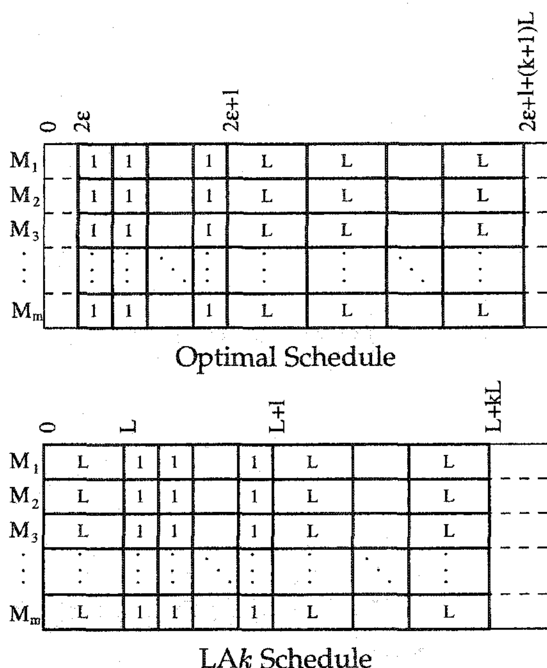
Considering the ratio, we have

$$\begin{aligned} \frac{C_{\mathrm{LA1}}}{C_{\mathrm{OPT}}} &= \frac{m\left((l+3)L+\frac{1}{2}l^2+\frac{3}{2}l\right)}{m\left(3L+\frac{1}{2}l^2+\left(\frac{5}{2}+2\epsilon\right)l+4\epsilon\right)} \\ &= \frac{\left(\frac{n}{m}+1\right)L+\frac{1}{2}l^2+\frac{3}{2}l}{3L+\frac{1}{2}l^2+\left(\frac{5}{2}+2\epsilon\right)l+4\epsilon} \\ &\to \frac{1}{3}\left(\frac{n}{m}+1\right) \qquad \text{as } L\to\infty. \end{aligned}$$

■

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 5.3: In Theorem 5.3, the optimal schedule has all of the machines idle until time 2ϵ, then schedules the short jobs followed by all the long jobs. In the LAk schedule, the first set of long jobs is executed immediately, then the short jobs, and finally, the remaining long jobs.



5.4 Lower bound for LAk

Theorem 5.3 *The competitive ratio for LAk is at least* $\frac{2}{(k+1)(k+2)}\left(\frac{n}{m}+\frac{1}{2}k(k+1)\right)$.

Proof: Consider the instance where m long jobs with processing time L arrive at time 0, another $k \cdot m$ long jobs with processing time L arrive at time ϵ and $l \cdot m$ short jobs with processing time 1 arrive at the 2ϵ. Note $n = m + k \cdot m + l \cdot m = (k + l + 1) \cdot m$.

In the optimal schedule, all m machines wait from time 0 to 2ϵ and then execute all short jobs before executing any long jobs. See Figure 5.3. Let C_{OPT} be the total completion time in the optimal schedule. We have

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$$\begin{aligned} C_{\mathrm{OPT}} &= m((2\epsilon+1)+(2\epsilon+2)+\ldots+(2\epsilon+l)+(2\epsilon+l+L)+ \\ &\quad (2\epsilon+l+2L)+\ldots+(2\epsilon+l+(k+1)L) \\ &= m\left(\frac{1}{2}(k+1)(k+2)L+\frac{1}{2}l^2+\left(\frac{3}{2}+k+2\epsilon\right)l+(k+1)2\epsilon\right). \end{aligned}$$

In the LAk schedule, all m long jobs with arrival time 0 are executed before any short jobs. The second batch of long jobs are scheduled last. See Figure 5.3. Let $C_{\mathrm{LA}k}$ be the total completion time in the LAk schedule. We have

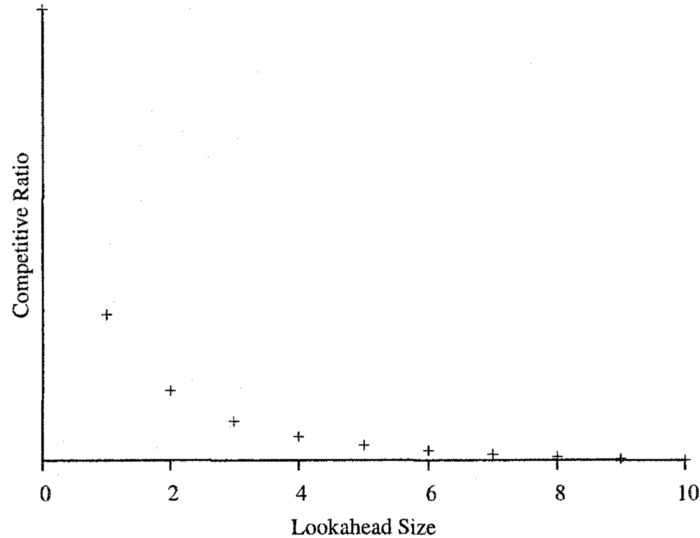
$$\begin{aligned} C_{\mathrm{LA}k} &= m(L+(L+1)+(L+2)+\ldots+(L+l)+(L+l+L)+ \\ &\quad (L+l+2L)+\ldots+(L+l+kL)) \\ &= m\left((l+k+1+\frac{1}{2}k(k+1))L+\frac{1}{2}l^2+\left(\frac{1}{2}+k\right)l\right). \end{aligned}$$

Considering the ratio, we have

$$\begin{aligned} \frac{C_{\mathrm{LA}k}}{C_{\mathrm{OPT}}} &= \frac{m\left((l+k+1+\frac{1}{2}k(k+1))L+\frac{1}{2}l^2+\left(\frac{1}{2}+k\right)l\right)}{m\left(\frac{1}{2}(k+1)(k+2)L+\frac{1}{2}l^2+\left(\frac{3}{2}+k+2\epsilon\right)l+(k+1)2\epsilon\right)} \\ &= \frac{\left(\frac{n}{m}+\frac{1}{2}k(k+1)\right)L+\frac{1}{2}l^2+\left(\frac{1}{2}+k\right)\cdot l}{\frac{1}{2}(k+1)(k+2)L+\frac{1}{2}l^2+\left(\frac{3}{2}+k+2\epsilon\right)l+(k+1)2\epsilon} \\ &\to \frac{2}{(k+1)(k+2)}\left(\frac{n}{m}+\frac{1}{2}k(k+1)\right) \qquad \text{as } L\to\infty. \end{aligned}$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 5.4: For a fixed m and n, when the LA0 algorithm is used, the competitive ratio is large, implying the worst-case instance is far from optimal. By adding a single lookahead, the ratio is reduced significantly. As the size of the lookahead grows beyond one, the amount of improvement diminishes.



■

5.5 Discussion

We suspect that the lower bounds from the previous sections are all tight bounds. That is, they are also upper bounds for the competitive ratios of the corresponding algorithms. It has been shown that when $m = 1$, the tight competitive ratio is $n = \frac{n}{m}$ for the SJF algorithm [80]. In addition, when $m = 1$, we know that the tight competitive ratio for LA1 is $\frac{1}{3}(n+1) = \frac{1}{3}\left(\frac{n}{m}+1\right)$[79]. Because our lower bounds cover the previous tight bounds for the special case of $m = 1$, we are encouraged that our ratios may also be tight.

The ratio for the LAk algorithm unifies our three results. When $k = 0$, we have LA0, which is SJF. The bound for LA0 is $\frac{2}{(k+1)(k+2)}\left(\frac{n}{m}+\frac{1}{2}k(k+1)\right) = \frac{2}{1\cdot 2}\left(\frac{n}{m}+\frac{1}{2}\cdot 0\right) = \frac{n}{m}$, which matches the bound we developed for SJF. When $k = 1$, we have LA1. The bound

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

for LAk when $k = 1$ is $\frac{2}{(k+1)(k+2)}\left(\frac{n}{m} + \frac{1}{2}k(k+1)\right) = \frac{2}{2\cdot 3}\left(\frac{n}{m} + \frac{1}{2}\cdot 1\cdot 2\right) = \frac{1}{3}\left(\frac{n}{m}+1\right)$, which matches our bound for LA1. Although these consistencies do not prove any of the bounds are upper bounds, it does show that there is a relationship between the various lower bounds. The existence of this relationship and our ratios that are consistent with previous tight ratios imply the lower bounds may be tight.

Assuming the bound for LAk is tight, an interesting fact comes out of the mathematics. First, note that in general, the bigger the size of k, the better the competitive ratio. Intuitively, as k approaches infinity, we have the optimal offline algorithm. When for some fixed constant c, we get

$$\begin{aligned} \frac{2}{(k+1)(k+2)}\cdot\left(\frac{n}{m}+\frac{1}{2}k(k+1)\right) &\leq \frac{2}{(k+1)(k+2)}\left(\frac{1}{2}(c-1)k^2+\frac{1}{2}k(k+1)\right) \\ &= \frac{k^2}{(k+1)(k+2)}(c-1)+\frac{k}{k+2} \\ &< (c-1)+1 \\ &= c. \end{aligned}$$

This indicates that the competitive ratio for LAk becomes bounded by the constant c for large k.

As a final point, we return to the discussion in Chapter 4 regarding the balance between execution time and amount of improvement. Figure 5.4 shows a plot of the competitive ratio of the LAk algorithm. In this graph, we vary k and hold both n and m fixed with $n \gg m$. A higher value of the competitive ratio means that the worst-case schedule is further from optimal. As expected, the quality of the schedule improves as the size of lookahead

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

increases. More interestingly, we note that there is a dramatic improvement from zero lookahead, the SJF algorithm, to a single lookahead job, LA1. Although a larger lookahead produce better schedules, the improvement is only marginal beyond Lookahead-1.

In Chapter 4, we showed that a LA1 algorithm can be implemented effectively in a real-time context, but that the running time of algorithms using more lookahead was prohibitive. Figure 5.4 supports these findings by demonstrating that the most improvement occurs between LA0 and LA1. Therefore, we conclude that a lookahead of 1 is the appropriate balance between efficiency and improvement.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 6

Simulation

6.1 Overview / Introduction

In addition to theoretical analysis, a second approach we take in evaluating lookahead algorithms is simulation experiments. There are a number of reasons why simulation is a very important tool for studying the behavior of algorithms. First, intuition tells us that lookahead algorithms may out-perform their online counterparts in situations when there are large variations in job lengths and small intervals between job arrivals. In addition, it seems logical that transient behavior, such as the queue size, may affect the relative performance of lookahead algorithms. Simulation can be used to verify these intuitions, whereas theoretical analysis cannot. Second, when theoretical analysis is difficult or impossible, simulation experiments provide insight. Third, although the competitive ratio provides an overall performance guarantee, it is sometimes too pessimistic to be informative in practice because it is a worst-case measure. In comparison, when a suitable probabilistic distribution is chosen, simulation gives average-case information on the performance of lookahead algorithms.

Overall, the goal of our simulation study is to quantify the performance of lookahead

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

algorithms on three job scheduling systems. We simulate a single machine system and a two identical machine system, the systems for which we developed lookahead algorithms in Chapter 4. The third is a two unrelated machine system. The difference between the dual machine systems concerns the service time of each job [1]. For identical machines, each job has the same service time on both machines. For the unrelated machines, each job has a potentially different service time on each machine. This configuration represents a system where each machine has different abilities. An example is the cross-docking model discussed in detail in Appendix A.

For all three systems, we assumed the system started empty and idle, there were no jobs in the single wait queue at time zero, and no jobs were executing. For each simulation, a fixed number of jobs arrived over time, and the simulations finished when all jobs had completed. These assumptions are consistent with the theoretical lookahead model described in Chapter 3.

We implemented the Shortest Job First (SJF) and Lookahead-1 (LA1) algorithms for all three systems. For the single machine system, we also implemented a Lookahead-2 (LA2) algorithm. We use the SJF algorithm as a point of comparison, justified because the LA1 and LA2 algorithms use the SJF rule when they decide to schedule. Therefore, SJF is also referred to as the Lookahead-0 (LA0) algorithm.

Regardless of the name, the SJF algorithm maintains the wait queue sorted by service time with the shortest job at the head of the queue. When a job arrives, it is inserted into the wait queue, and when a machine becomes idle, the first job in the queue is scheduled on that machine.

[1] Consistent with simulation terminology, in this chapter we use service time rather than processing time.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The LA1 algorithm for job scheduling maintains a lookahead queue in addition to the sorted wait queue. The lookahead queue contains a single job, the lookahead job, which is the next job that will arrive. Assuming the wait queue is not empty when a machine becomes idle, the algorithm must decide whether to wait for the lookahead job or to schedule the shortest job in the wait queue. From the work of Mao and Kincaid [79], we know that using the local optimal, the algorithm can make this decision in constant time for the single machine system [79], and from Chapter 4, we know the running time is near-constant for the two identical machine system. In both cases, if waiting produces a smaller total completion time of the jobs in the wait queue and lookahead queue, the algorithm waits for the lookahead job. Otherwise, the first and shortest job in the queue is executed.

Similar to the LA1 algorithm, the LA2 algorithm for job scheduling maintains both a wait queue and a lookahead queue. In this case, the lookahead queue always contains the next two jobs to arrive. The LA2 algorithm uses the information in both queues to calculate the local optimal schedule and then decides, consistent with this schedule, whether to wait or schedule. From Chapter 4, we know this decision can be made in linear time for the single machine system.

For all simulations, inter-arrival times were assumed to be Exponentially distributed. We studied both a stationary arrival process, where jobs arrive at a constant average rate throughout the simulation, and a Markovian Modulated Poisson Process (MMPP), where the arrival rate toggles between two rates. Results were gathered when the service times were taken from Uniform, Triangle, Erlang, Exponential, and Hyperexponential distributions. Table 6.1 lists the important information for each of these distributions.

The most important difference between these simulation results and the previous the-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table 6.1: The mean and standard deviation for each distribution used in the simulations.

Distribution	mean	standard deviation
Uniform(a, b)	$\frac{a+b}{2}$	$\frac{b-a}{\sqrt{12}}$
Triangle(a, b, c)	$\frac{a+b+c}{3}$	$\frac{\sqrt{(a-b)^2+(a-c)^2+(b-c)^2}}{6}$
Erlang(a, b)	ab	$\sqrt{a}b$
Exponential(μ)	μ	μ
Hyper(μ_1, μ_2, α)	$\alpha\mu_1 + (1-\alpha)\mu_2$	$\alpha\mu_1^2 + (1-\alpha)\mu_2^2$

oretical results is the algorithm with which we compare. In our theoretical study, we compared against the optimal algorithm to produce a worst-case measure. With our simulation experiments, we use the SJF as our point of comparison to generate average-case measures.

Because of this distinction, we use different criteria for each approach. In our theoretical results, we minimize the total completion time. In the simulations, we measure the average wait time. For job J_j, the completion time, C_j, is the arrival time, r_j, plus the time spent in the queue, or wait time, w_j, plus the service time, p_j. We see that the two criteria are equivalent since

$$\frac{1}{n}\sum C_j = \frac{1}{n}\sum (r_j + w_j + p_j) = \left[\frac{1}{n}\sum (r_j + p_j)\right] + \frac{1}{n}\sum w_j.$$

Notice that $\sum(r_j + p_j)$ is constant for a given instance of n jobs. Only a change in the wait time affects the sum of completion times and average wait time.

Using the average wait time is necessary with simulation because of the metric we use to compare two algorithms. The percent improvement of algorithm A_1 over algorithm A_2 for an instance is defined in terms of the cost of the two schedules, C_1 and C_2:

$$\frac{C_2 - C_1}{C_2}$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Let C represent the sum of arrival times plus the sum of service times for a set of jobs. Further, let W_1 be the sum of wait times incurred from A_1 and W_2 be the sum of wait times from A_2. Then the percent improvement using the sum of completion times is

$$\frac{(C + W_2) - (C + W_1)}{C + W_2} = \frac{W_2 - W_1}{C + W_2}$$

Using the average wait time, we have

$$\frac{\frac{1}{n}W_2 - \frac{1}{n}W_1}{\frac{1}{n}W_2} = \frac{W_2 - W_1}{W_2}$$

In both ratios, the numerator represents the difference in wait times between the two schedules. However, when the sum of completion times is used as the criteria, the denominator of the percent improvement contains C and is the sum of completion times of the second algorithm. Because each completion time is larger than the previous, this quantity dominates the ratio. Mathematically, if the number of jobs is large, then $C = \sum r_j + \sum p_j$ is also large. In the limit as n goes to infinity, the percent improvement using the sum of completion times goes to zero. In terms of our finite simulations, the measured percent improvement of the total completion time will always be less than the equivalent instance when calculating the average wait time.

6.2 Results

Our experiments were broken down into two main categories. First, we measured the improvement of lookahead algorithms with short bursts of 25 jobs, called the transient behavior. This type of study provides a very detailed look at how much improvement

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

is expected in the short term. The second approach is to execute the simulations with a very large number of jobs and measure the overall performance, called the steady-state behavior. Although more commonly used, this approach dilutes the improvement of the lookahead algorithm because the large majority of decisions do not result in a waiting decision, and improvement only occurs when the algorithm waits.

In Section 6.2.1, we consider the transient behavior of lookahead algorithms, and in Section 6.2.2, we consider the steady-state behavior.

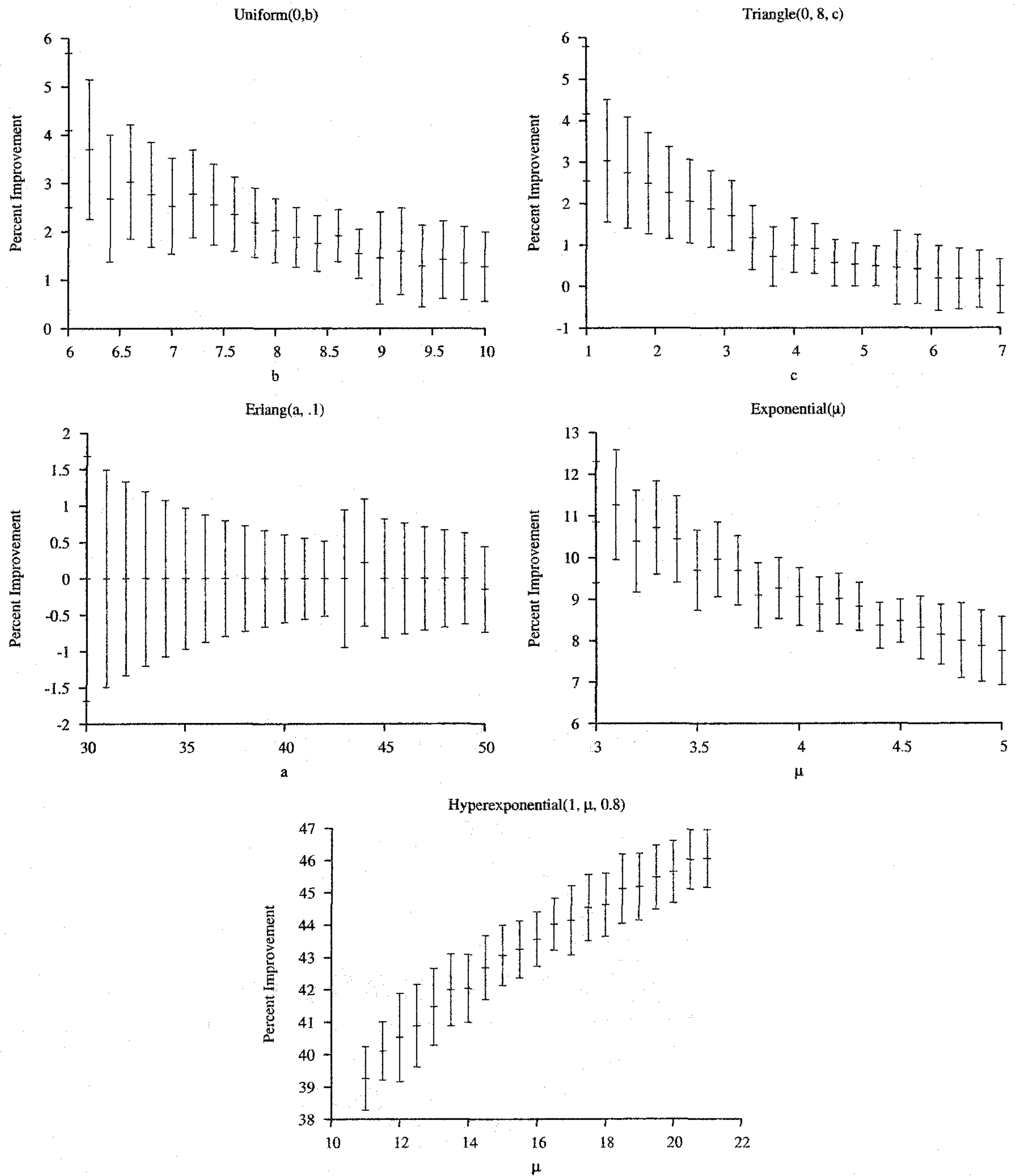
6.2.1 Transient Behavior

The goal of these simulations was to measure the improvement of lookahead algorithms over the Shortest Job First (SJF) algorithm. For the two identical machine system, we measured the average wait time for 100,000 executions of 25 jobs. Using these values, we computed the percent improvement of the LA1 algorithm over the SJF algorithm. Figure 6.1 contains graphs showing these results when the service times were drawn from Uniform, Triangle, Erlang, Exponential, and Hyperexponential distributions. For each graph, the range of average service times was between 3 and 5 time units. This means the only difference between the graphs is the service time distribution.

An obvious difference between the graphs is the significant improvement when service times are drawn from Exponential or Hyperexponential distributions and the marginal improvement of the Erlang, Triangle and Uniform distributions. In some cases, the SJF algorithm actually performs better than the LA1 algorithm, resulting in a negative percent improvement. In these cases, the sequences of jobs that arrived contained an instance where the LA1 algorithm incorrectly decided to wait for a job. These instances highlight

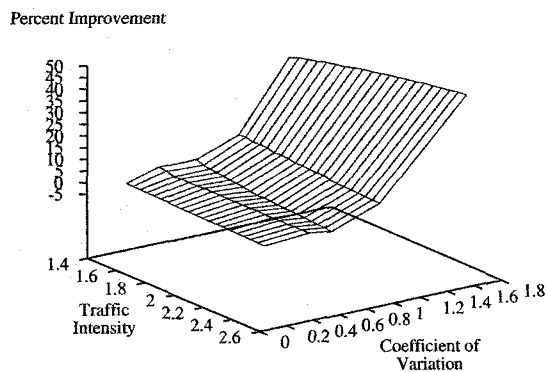
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 6.1: For two identical machines, we measured the average wait time for 25 jobs with service times drawn from various distributions. A single parameter was varied to produce average service times between 3 and 5 time units. Using 100,000 repetitions, each graph shows 95% confidence intervals on the percent improvement of the LA1 algorithm over the SJF algorithm. When the service times were drawn from distributions with small variance, little or no improvement is seen. However, when the variance is large, significant improvement occurs.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 6.2: Using the traffic intensity and coefficient of variation as unifying metrics, this graph is a composite of the five graphs in Figure 6.1. For a given traffic intensity, as the coefficient of variation increases the variance also increases. This graph shows the correlation between amount of improvement and variance.

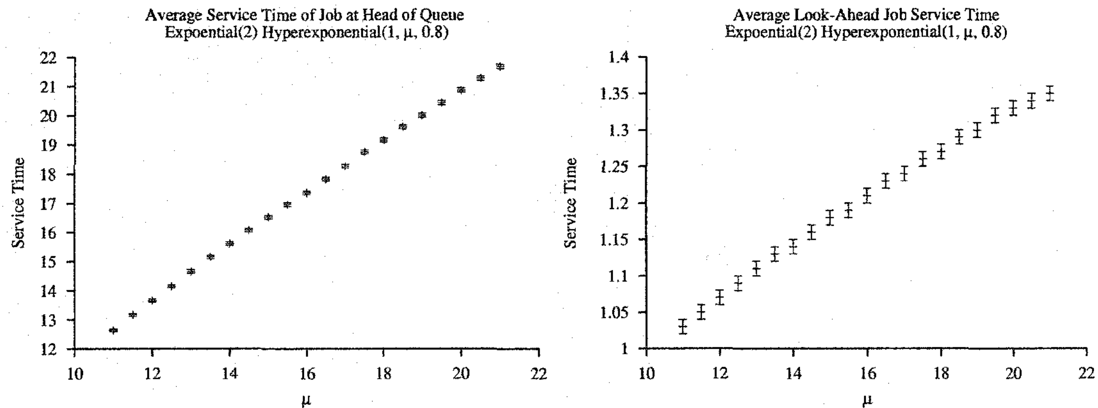


the fact that the LA1 algorithm is an approximation algorithm and sometimes makes bad scheduling decisions.

One important quality of the graphs in Figure 6.1 is that the performance is fairly consistent for each distribution. Drawing service times from the Hyperexponential distribution always produced significant improvement and using the Erlang distribution always produced schedules that were nearly identical to those of the SJF algorithm. Although each distribution uses the same range of means, the standard deviation varies. A unifying value across the distributions is the coefficient of variation, the standard deviation over the mean. Roughly speaking, this unit-less value represents the spread of a distribution. In the context of job lengths, larger values imply an increased likelihood that a long job was followed by a short job. Using the coefficient of variation, we can meaningfully compare the distributions at each traffic intensity. Figure 6.2 shows the percent improvement compared with both the traffic intensity and the coefficient of variation. The error intervals associ-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 6.3: In the two identical machine system, we measured the size of the lookahead job and the first job in the wait queue each time the algorithm decided to wait. These graphs show that long jobs were delayed to wait for jobs that were 12 to 16 times shorter.



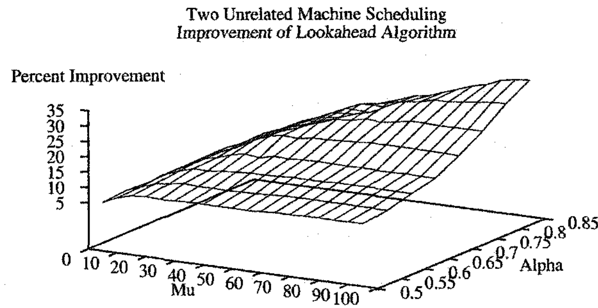
ated with each point are omitted for clarity because this graph is a composition of Figure 6.1. The graph shows that more improvement occurs when the coefficient of variation is higher. The constant mean for each set of data indicates that the standard deviation of the distribution, and in turn the variance, is an indicator of the amount of improvement.

Consequently, we see that improvement occurs when long jobs are delayed for shorts jobs. Recall that the lookahead algorithm either decides to schedule the first job in the queue or wait for the lookahead job. Using the two identical machine system, we collected job size information each time lookahead was utilized. In Figure 6.3, the size of the first job in the wait queue and the size of the lookahead job are both shown when the algorithm decides to wait. The waiting job is between 12 and 16 times larger than the lookahead job. Thus, lookahead is utilized to delay very long jobs to wait for short jobs.

These conclusions can be seen in the other systems as well. Figure 6.4 shows up to a 35 percent improvement of the LA1 algorithm over the SJF algorithm for two unrelated machines. In this case, only the Hyperexponential distribution was used for service times.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 6.4: In the two unrelated machine system, we measured the percent improvement for 25 jobs with service times drawn from a Hyperexponential($1, \mu, \alpha$) distribution. As μ increases the variance increases, and the LA1 algorithm produces more improvement over the SJF algorithm. As α varies, the ratio of long jobs and short jobs changes. Although the variance increases as α decreases, the percent improvement has the opposite trend.



The variance of a Hyperexponential(μ_1, μ_2, α) variable is defined to be

$$\alpha \cdot \mu_1^2 + (1 - \alpha) \cdot \mu_2^2.$$

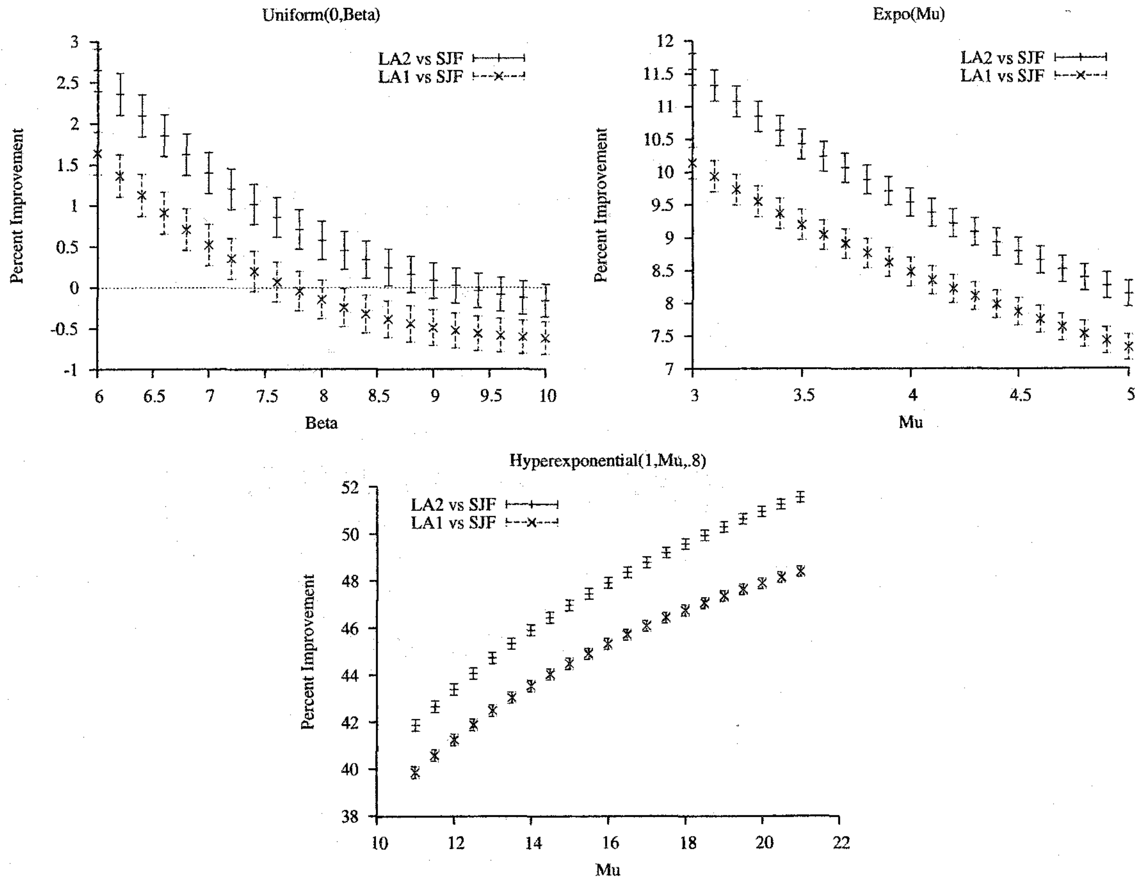
Therefore, as μ_2 increases, so does the variance. Figure 6.4 shows that as μ_2 increases, so does the improvement. Note from the above equation that as α increases, the variance decreases. However, Figure 6.4 shows a decrease of improvement as α increases. This demonstrates that the level of variance is not a perfect indicator of the amount of improvement.

For larger values of α, the proportion of short jobs to long jobs is large. Therefore, when a long job is in the queue, it is very likely that a short job is the lookahead job, and consequently, that the long job is delayed. On the other hand, when α is near .5, long jobs in the queue are equally likely to have long and short jobs as the lookahead job. Therefore, utilizing lookahead is less likely, and the overall improvement is decreased.

Finally, we consider the improvement in the single machine system. In this case, we

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

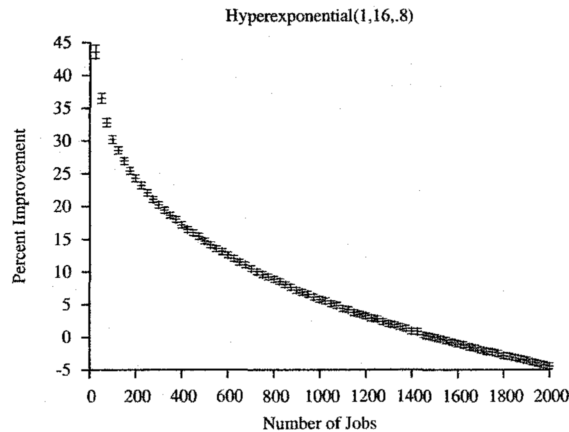
Figure 6.5: In the single machine system, we measured the percent improvement of the LA1 algorithm over the SJF algorithm and the LA2 algorithm over the SJF algorithm with service times drawn from Uniform, Exponential, and Hyperexponential distributions. Using 95% confidence intervals, the graphs demonstrate that higher variance corresponds to larger improvement.



are comparing both the LA1 and LA2 algorithm against the SJF algorithm. Similar to the graphs for the two identical machine system, Figure 6.5 shows the improvement of Uniform, Exponential, and Hyperexponential distributions for service times. As with the two identical machine system, higher variance results in more improvement.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 6.6: For two identical machines, we drew service times from a fixed Hyperexponential distribution and varied the number of jobs. As the number of jobs increases, the improvement of the LA1 algorithm over the SJF algorithm steadily decreases and eventually becomes negative.

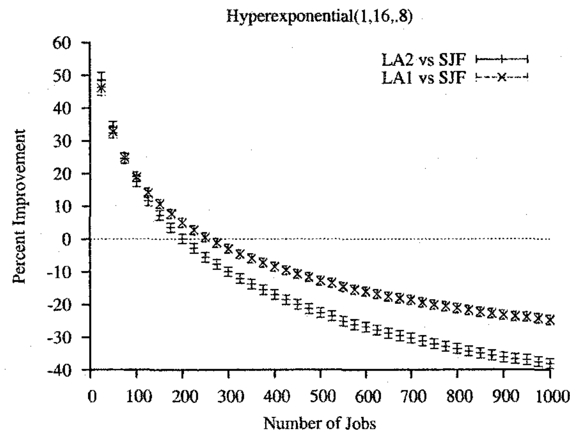


6.2.2 Steady-State Behavior

For the simulations studied so far, the number of jobs was only 25. Figures 6.6 and 6.7 demonstrate the effect of increasing the number of jobs. The distribution used for service times was Hyperexponential with parameters chosen so that, on average, jobs arrived at exactly the same rate they could be processed. For both scheduling systems, the percent improvement decreases dramatically as the number of jobs increases. For the single machine system, the LA1 algorithm produces worse schedules when the number of jobs is larger than approximately 250. To understand this, consider the complete schedule for a large number of jobs in the single machine system. Because we have a constant traffic intensity, jobs arrive such that the final schedule contains long blocks where jobs are scheduled without the machine becoming idle. Consider one such block containing n jobs. Assume that at time t somewhere toward the beginning of the block, the scheduling algorithm considers the jobs in the queue and the lookahead job. Based on the available information, waiting appears to be beneficial. In general, this means that the amount of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 6.7: In the single machine system, we drew service times from a Hyperexponential distribution and varied the number of jobs. Because the jobs arrive at a constant rate, the schedule has few idle period. Consequently, both the LA1 and LA2 algorithms perform poorly because waiting is rarely beneficial.

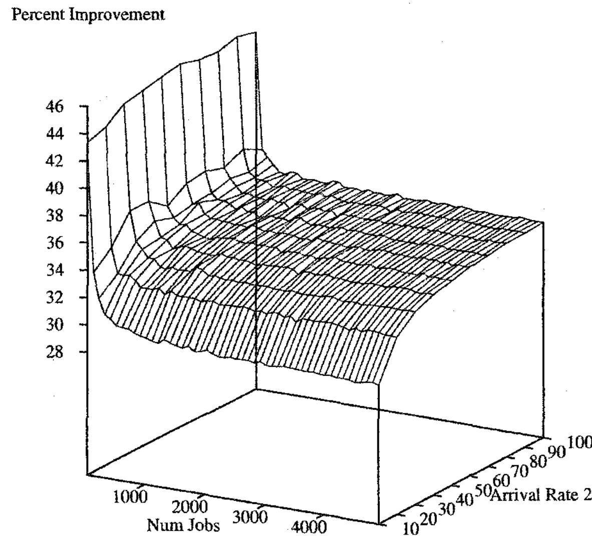


delay incurred by waiting is smaller than the gain of scheduling a lookahead job next. However, the incurred delay affects not only the current jobs in the wait queue, but also the remaining jobs in the block that have not arrived. Assuming the remaining jobs are scheduled without another wait, we have a long sequence of jobs scheduled one after another. With this long block of jobs, the decision to wait early in the block is actually a bad decision. The wait incurred affects not only the jobs currently in the queue, but also those that arrived later and were scheduled in the same block. Therefore, lookahead can ultimately create schedules with increased average wait time.

Because a constant arrival rate creates long blocks of jobs that cause problems for lookahead algorithms, we next considered a bursty arrival process. In a Markovian Modulated Poisson Process (MMPP), inter-arrival times are computed using three Exponentially distributed variables with averages μ_1, μ_2, and μ_3. Inter-arrival times are drawn from an Exponential(μ_1) distribution for some period and then switch to inter-arrivals with an

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 6.8: For two identical machines, we drew service times from a Markov-Modulated Poisson Process that caused jobs to arrive in bursts of approximately 25 jobs. Unlike Figure 6.6, the improvement stabilizes as the number of jobs increase.

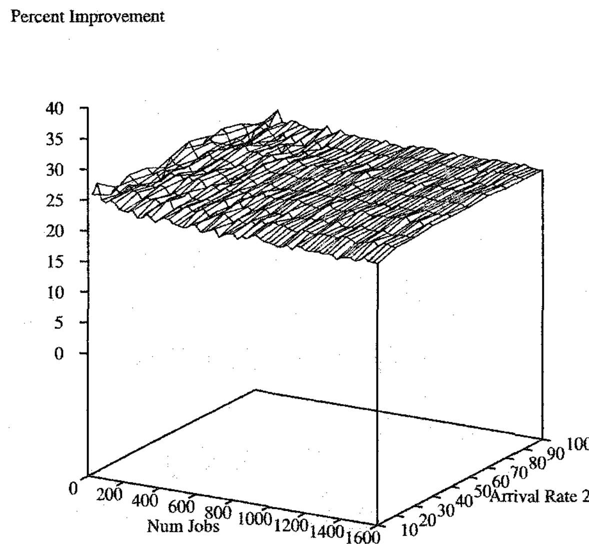


Exponential(μ_2) distribution. The length of each interval is drawn from an Exponential(μ_3) distribution. When we fix $\mu_1 = 4$ and $\mu_3 = 100$, on average 25 jobs arrive during half the bursts. When μ_2 is large, few jobs arrive during the other bursts. This creates blocks in the final schedule of approximately 25 jobs separated by idle periods. Figure 6.8 shows the effect of an MMPP arrival process on the two identical machine system using Hyper-exponential$(1, 16, 0.8)$ service times. Similar experiments were performed for the two unrelated machine system and the single machine system. These results are shown in Figures 6.9 and 6.10.

In all three graphs, we see that the percent improvement plateaus. For the two identical machine system, the improvement is around 35 percent, for the two unrelated machine

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 6.9: For two unrelated machines, service times were drawn from a Markov-Modulated Poisson Process that caused jobs to arrive in bursts of approximately 25 jobs. For this system, the improvement is consistent regardless of the parameter values.

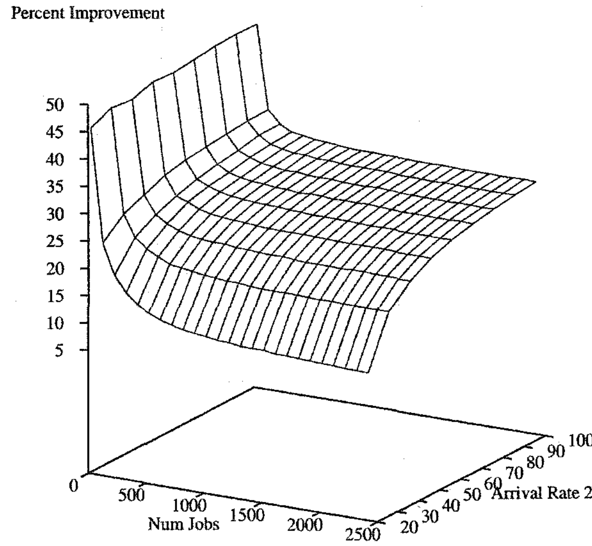


system 25 percent, and for the single machine system around 30 percent for both the LA1 and LA2 algorithms. This plateau is achieved both as the number of jobs increases and μ_2 increases. Intuitively, as μ_2 approaches μ_3, fewer jobs arrive during the "off" alternations. Thus, all four graphs demonstrate that lookahead algorithms produce significant improvement if small or medium-sized bursts of jobs occur.

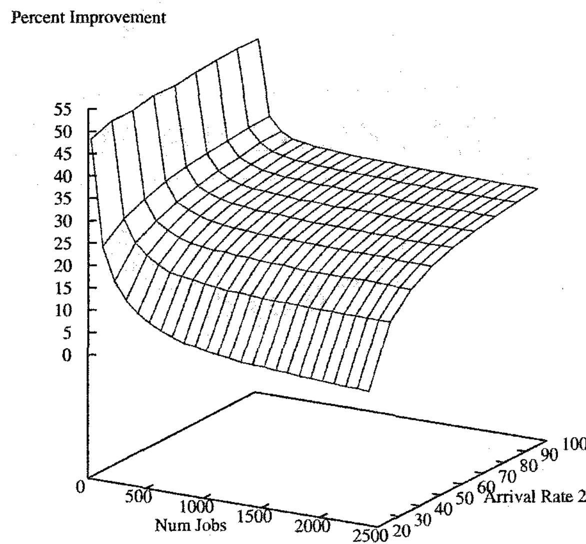
A final consideration is the relative performance of the LA1 and LA2 algorithms. As expected, for the single machine system in Figures 6.5 and 6.10 we see that the LA2 algorithm produces better schedules than the LA1 algorithm, and both lookahead algorithms perform better than the SJF algorithm. However, the amount of improvement of the LA2 algorithm over the LA1 algorithm is not as significant. That is, most of the improvement

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 6.10: For the single machine system, when service times were drawn from a Markov-Modulated Poisson Process, the improvement stabilizes for both that LA1 and LA2 algorithms.



Improvement of LA1 over SJF



Improvement of LA2 over SJF

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

comes from LA1 over SJF and the inclusion of a second lookahead job only adds a small amount of additional improvement. From the algorithm design perspective in Chapter 4, the complexity of the LA2 algorithm for the single machine system is significantly more involved than its Lookahead-1 counterpart [79]. In addition, the LA1 algorithm runs in constant time while the LA2 algorithm requires linear time, in the worst case. For these reasons, we conclude that LA1 algorithms are important to develop, but the added effort to design algorithms utilizing more lookahead may not be worth the resources.

6.3 Discussion

From these experiments, we gained significant insight into the behavior of lookahead algorithms. As expected, lookahead algorithms produce superior schedules in almost all situations. The most important requirement for improvement is a mixture of short and long jobs. In general, a high variance of the service times, and similarly a high coefficient of variance, indicate that this mixture is present. These values are not perfect indicators, however, as seen with the Hyperexponential distribution where we increased the variance but decreased the improvement.

A second requirement for improvement is small to medium-sized blocks of jobs. When a large number of jobs arrive such that they can be scheduled without any idle periods, using lookahead is detrimental. This is especially true when decisions to wait are made at the beginning of these long blocks. Because the lookahead algorithm makes a local decision by considering only the known jobs, it makes a poor decision that delays not only the known jobs, but also many jobs that arrive in the future. Therefore, what appears to be

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

a good decision immediately turns out to be a very poor decision in the future.

Even when these requirements for improvement are met, we were surprised to discover that while algorithms with more lookahead produce better results, the amount of improvement is not significantly larger than a Lookahead-1 algorithm. This is important when we consider the effort required to develop these algorithms. In addition, algorithms with more than a single lookahead job have linear running times, or worse, making them poor candidates for real-time systems. We conclude that there is incentive to develop a Lookahead-1 algorithm, but utilizing more lookahead is not cost-effective.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 7

Conclusion

7.1 Summary of Results and Conclusions

We have considered how lookahead can be applied to resource assignment problems, using job scheduling as a case study. Our first contribution is a formal model defining how to incorporate lookahead in a real-time context. This model realistically describes resource assignment problems because time is incorporated in a meaningful way.

In the area of algorithm design, we showed that lookahead algorithms can be implemented using rules that make decisions quickly because they typically do not have to consider every job in the wait queue. First, we developed a Lookahead-1 algorithm for a two machine system and showed that in most cases scheduling decisions can be made by considering only the first job in the wait queue, the length of the wait queue, and the lookahead job. On a single machine system, we demonstrated that a Lookahead-2 algorithm can be implemented using similar rules, however the development was significantly more complex. To make scheduling decisions, the algorithm frequently has to consider a larger number of jobs.

To study the performance of lookahead algorithms, we utilized both theoretical anal-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ysis and simulation studies. We showed lower bounds for the competitive ratio of the Shortest Job First (SJF) scheduling algorithm, also called the Lookahead-0 (LA0) algorithm, a Lookahead-1 algorithm, and a Lookahead-k algorithm for the identical parallel machine scheduling problem. Although these results suggest that knowledge of more future jobs will result in better schedules, they do not suggest how much improvement will occur in practice.

To answer this question, we compared lookahead algorithms with the SJF algorithm through simulation. On all three job scheduling systems considered, the lookahead algorithms produced schedules that were 25 to 40 percent better than the schedule created by the SJF algorithm. This improvement occurred when long jobs were delayed to wait for short jobs and when the jobs arrived in short bursts. The significant improvement implies that when any information about future jobs is available, it should be utilized to create better schedules.

Taking our design results together with our performance results, we conclude that lookahead is an important factor when trying to minimize the average wait time or, equivalently, the total completion time. Lookahead-1 algorithms produce a significant improvement over their non-lookahead counterparts. However, using information about jobs beyond Lookahead-1 provides only limited additional improvement. Because the amount of improvement is not proportional to the amount of additional effort necessary to design these algorithms and the running time of algorithms with more than a single lookahead is prohibitive in a real-time context, we conclude that Lookahead-1 algorithms should be utilized when possible, but Lookahead-k algorithms for $k > 1$ should not.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

7.2 Future Work

We have three areas of related future work. Two fit within our current model, and the third represents a minor deviation. First, we want to look for a way to mathematically predict when lookahead will be beneficial. In our simulation study in Chapter 6, we found that when service times were drawn from high-variance distributions, we typically had better improvement. However, there was one instance where we increased the variance of the Hyperexponential distribution (by decreasing α) and the improvement decreased. We are curious whether there is a distribution statistic that accurately predicts the amount of improvement. To study this idea, we will consider the skewness of each distribution, defined as

$$\frac{\mu_3}{\mu_2^{3/2}}.$$

In this expression μ_2 and μ_3 are the second and third moments of the distribution. If $\mathcal{X}$ is a random variable, then the ith moment is defined to be $E[\mathcal{X}^n]$. Using this definition, the skewness measures the degree of asymmetry of a distribution. Through simulation, we will study whether there is a direct correlation between skewness and the improvement of lookahead algorithms.

A second direction for future study is related to the metric used to construct a schedule. All of our results are based on the criteria of minimizing the average wait time or total completion time, but we are interested in considering other criteria. For some of these, we know that lookahead will not be effective. For example, when trying to minimize the makespan, or largest completion time, lookahead is less useful at the beginning of the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

scheduling process because, on average, the most important decisions are made at the end of the schedule. However, other criteria are more promising. Specifically, with the criteria of lateness, each job has a deadline and the goal is to minimize the number of jobs that miss their deadline. Using both theoretical analysis and simulation, we plan to study lookahead scheduling for models with lateness as a criteria.

In the final area for future research, we will change how the algorithm makes decisions. In our current model, the algorithm always makes the choice that appears to create the best solution, called the local-optimal. Hoogeveen and Vestjens [51] developed an online method for the single job scheduling problem that deviates from this approach. Their algorithm always waits until a job could have completed and then it schedules the job. At time t, if the algorithm believes it should schedule a job with processing time p_j, then it waits until time $t + p_j$ before actually executing the job. If another job with smaller processing time arrives during this idle period, it becomes the delayed job and the waiting period resets. Although this approach seems counter productive, it has the best-known competitive ratio.

We believe that adding lookahead to this technique will produce an even better competitive ratio, and we have started development of a Lookahead-1 algorithm based on this idea. Because this algorithm does not make local-optimal decisions, this is a departure from our original approach. Instead, the algorithm attempts to minimize the overall worst-case rather than minimizing individual instances. In effect, this is a defensive algorithm that does not allow any instance to stray too far from the optimal.

Both this defensive model and our more traditional model continue to be active areas of research within the resource assignment problem community. Our contributions in

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

this area will build on our successful development of lookahead algorithms in a real-time context.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Appendix A

Cross-docking Model

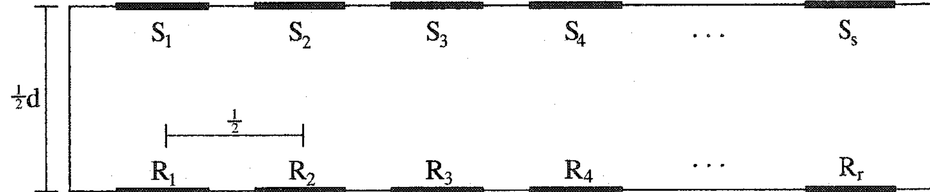
A.1 Introduction

A special case of unrelated machine scheduling is cross-docking. In this model, trucks must be assigned to doors at a shipping and receiving warehouse so the contents of the truck can be unloaded, sorted, and moved across the warehouse to be loaded on different trucks. Specifically, we have a set $R = \{R_1 \dots R_r\}$ of receiving doors where trucks are unloaded and a set $S = \{S_1 \dots S_s\}$ of shipping doors where cargo is loaded. Although these doors could be placed anywhere on an arbitrarily shaped building, we assume the doors are evenly spaced across two sides of a rectangular building with receiving doors on one side and shipping doors on the other. The round-trip distance between any two adjacent doors is one and the round-trip distance across the building is d. See Figure A.1.

We have a set $T = \{T_1 \dots T_n\}$ of trucks to be unloaded at the receiving doors, each with arrival time r_i. Each truck T_i contains a sequence of pallets $B_{i,1} \dots B_{i,n_i}$ and a capacity of c, implying that $1 \leq n_i \leq c$ for all i (a truck may not be empty). There is a linear precedence relation placed upon the pallets, $B_{i,1} \to B_{i,2} \to \dots \to B_{i,n_i}$ requiring that the pallets are unloaded in order. Associated with each pallet is a destination, $d_{i,j}$. To unload a truck, each

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure A.1: In the cross-docking model, the building is a long thin rectangle with receiving doors on one side and shipping doors on the other. The distance between two doors is one half and the distance across the facility is $\frac{1}{2}d$. Therefore the round-trip distances between doors is one and across the facility is d.



pallet must be moved to its assigned destination. If $d_{i,j} = d_{i,j+1}$, two trips are required.

The time to move one pallet from receiving door k to shipping door l is $|k - l| + d$, the round-trip rectilinear distance between the receiving door and the shipping door. Thus, the time to unload an entire truck T_i assigned to receiving door k is

$$\sum_{1 \leq j \leq n_i} |d_{i,j} - k| + d$$

We are interested in the completion time of each truck. Specifically, if truck T_i is assigned to receiving door k at time s_i, then the completion time is

$$c_i = s_i + \sum_{1 \leq j \leq n_i} |d_{i,j} - k| + d$$

The goal is to assign the trucks to the receiving doors so as to minimize $\sum c_i$, the sum of completion times for all trucks.

A.2 Relationship to Job Scheduling

This cross-docking model is a special case of the job scheduling problem $R|r_j|\sum C_j$. In this problem, we have n jobs to schedule on m machines. The machines are called unre-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

lated because each job has a potentially different processing time on each machine, and there is no relationship between the processing times of two jobs. This model accurately represents the situation where each machine has different abilities, although each machine can complete each job.

For each truck in the cross-docking model, the unloading time is determined by which door the truck is assigned to. The destinations of the pallets on the truck determine how long it will take the unloader to move the cargo from a particular receiving door. This value will be different for each truck and door combination.

An important fact about this model is that each truck has both an upper bound and a lower bound for the unloading time of any single truck. For a truck that contains c pallets, the smallest possible unloading time occurs when all c pallets have the same destination and the truck is assigned to the door directly across from this destination door. In this case, the unloading time is $c \cdot d$, that is, the unloader will have to move back and forth the width of the warehouse once for each pallet. Similarly, the largest possible unloading time occurs when the truck is full of pallets destined for door 1 and is assigned to door R_r (or destined for door S_s and assigned to door 1). In either case, the unloader will have to traverse the length and width of the warehouse for each pallet, with a total unloading time of $c \cdot (d + s)$.

These upper and lower bounds mean that we have a special-case of the unrelated machine scheduling problem. In the general problem, processing times can be arbitrarily large or small.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

A.3 Generating Solutions

In our model, discussed in Chapter 3, the scheduler makes decisions using a three-step process. First, it assumes that no jobs will arrive other than those in the lookahead queue. Second, it calculates the optimal schedules of the known jobs for the waiting and non-waiting schedules. Finally, the scheduler decides to wait or schedule based on which of the two schedules produces the smaller value of the metric.

We are interested in a technique to calculate the optimal waiting and non-waiting schedules for the unrelated machine scheduling problem with a single lookahead job. Because the cross-docking system is a special case of this problem, this approach can be used to solve our specific problem. In the next section, we discuss the calculation of the waiting schedule when the machines are all idle. The following section generalizes this approach for non-idle machines. Finally, Section A.3.3 discusses the technique to calculate the non-waiting schedule.

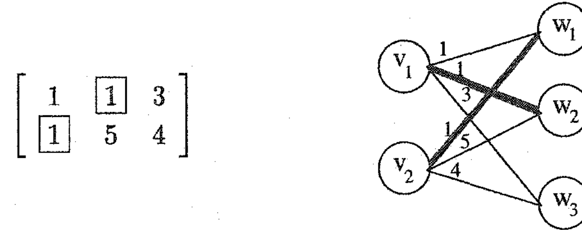
A.3.1 Calculating the Waiting Schedule

Because we assume that no other jobs will arrive, we have an instance of unrelated machine scheduling *without* arrival times when calculating the waiting schedule. In scheduling notation, this problem is $R||\sum c_j$. In the waiting schedule, we wait until the lookahead job arrives and is placed in the wait queue. After this arrival, we know all the jobs that must be scheduled.

This problem can be reduced to the assignment problem, also called the weighted matching problem. Here we have a complete bipartite graph $G = (V_1 \cup V_2, V_1 \times V_2)$ with

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure A.2: The assignment problem can be viewed as an m by n matrix or as a bipartite graph with m nodes on one side and n on the other.



$V_1 = \{v_1 \dots v_n\}$ and $V_2 = \{w_1 \dots w_m\}$. Without loss of generality, we assume $n < m$. Each edge (v_i, w_j) has an associated weight or cost $c_{i,j}$. The goal is to find $S \subset V_1 \times V_2$ with $(v_i, w_j), (v_k, w_l) \in S$, $v_i \neq v_k$ and $w_j \neq w_l$ such that the sum of the weights in S is minimized. A representative instance of the assignment problem is the task of assigning m people to n rooms. Each person ranks the rooms from 1 to n. The goal is to assign each person a room with as high a rank as possible.

The assignment problem can easily be represented as an $n \times m$ matrix. Intuitively, the goal is to choose n entries in the matrix such that no two entries are in the same row or column and the sum of the entries is minimized. Figure A.2 is an example of the assignment problem in both matrix and graph representation.

In the reduction from the scheduling problem to the assignment problem, each node in V_1 is a machine/position pair. $V_1 = \{v_{1,1}, v_{1,2}, \dots v_{1,m}, v_{2,1}, v_{2,2}, \dots v_{2,m} \dots v_{n,1} \dots v_{n,m}\}$. Node $v_{1,1}$ represents a job being scheduled last on machine 1. Node $v_{1,2}$ represents a job being scheduled last on machine 2. In general, $v_{i,j}$ represents a job being scheduled i from last on machine j. Because it is possible for all n jobs to be scheduled on any of the machines, there are nm nodes in V_1. $V_2 = \{w_1, w_2, \dots w_n\}$, and represent each job. Thus, $|V_2| = n$.

The edges in G are based on the contribution a job would make to the total completion

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

time if it were scheduled in a given position on a given machine. If jobs J_1, J_2, and J_3 were scheduled in order on machine 1 at time 0, then the total completion time for these jobs would be $p_{1,1} + (p_{1,1} + p_{2,1}) + (p_{1,1} + p_{2,1} + p_{3,1}) = 3p_{1,1} + 2p_{2,1} + 1p_{3,1}$. Therefore if job k is scheduled on machine j as the i from last job, it contributes $i \cdot p_{k,j}$. In our graph edge $(v_{i,j}, w_k)$ represents this occurrence with weight $i \cdot p_{k,j}$.

The overall graph can be represented as an mn by n matrix. Let $[\tau_{i,j}]$ be the m by n matrix representing processing times. Entry $\tau_{i,j}$ of the matrix contains the value $p_{i,j}$. Then the mn by n matrix representing the corresponding bipartite graph will have the form

$$\begin{bmatrix} [\tau_{i,j}] \\ 2\,[\tau_{i,j}] \\ \vdots \\ n\,[\tau_{i,j}] \end{bmatrix}$$

Figure A.3 represents an example of the scheduling problem reduced to the assignment problem and shows the solution in graph, matrix, and schedule form.

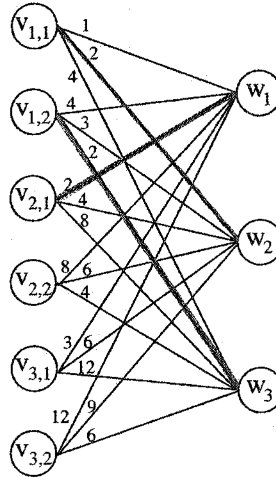
Given this complete weighted bipartite graph, the goal is to find an assignment of minimum weight. Bruno, Coffman, and Sethi [19] show that this assignment represents the optimal schedule.

To solve the assignment problem, we reduce it to a weighted flow problem. In a flow problem, we have a directed graph with two special nodes. The source, s, has only out-going edges and the sink, t, has only in-coming edges. Each edge has a maximum capacity of flow as well as a cost per unit flow. The goal is to find a maximum flow from s to t that produces the minimum cost. The reduction is accomplished by adding a source, s, and a sink, t, to the existing bipartite graph. We also add an edge (s, v_i) for all $v_i \in V_1$ and an edge (w_i, t) for all $w_i \in V_2$. Each of these new edges have cost (weight) 0. All edges in the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure A.3: When an instance of n jobs to be scheduled on m machines is reduced to an instance of the assignment problem, the result can be viewed as an mn by n matrix or a bipartite graph with mn nodes on one side and n nodes on the other.

$$\begin{bmatrix} 1 & \boxed{2} & 4 \\ 4 & 3 & \boxed{2} \\ \hline \boxed{2} & 4 & 8 \\ 8 & 6 & 4 \\ \hline 3 & 6 & 12 \\ 12 & 9 & 6 \end{bmatrix}$$



D_1	T_1	T_2	
D_2	T_3		

graph are given capacity 1.

Solving flow problems is based on the concept of a residual graph. A residual graph $G = (V, E')$ represents the edges in the original graph that can admit more flow. If $c(u, v)$ is the capacity of an edge and $f(u, v)$ is the flow on an edge, then $c_r(u, v) = c(u, v) - f(u, v)$ is the capacity of the edge in the residual graph. Note that E' will contain edges not in E and vice versa. For example, if $f(u, v) = c(u, v)$ then $c_r(u, v) = 0$ and the edge is not included in E'. In this case, (v, u) would be an edge in E' with $c(v, u) = f(u, v)$.

The method to solve a weighted flow problem is based on the ideas used to solve a simple flow problem. The Ford-Fulkerson method [29] states that we repeatedly search for a path from s to t in the residual network. This augmenting path can be used to move addition flow in the original graph. The max-flow, min-cut theorem states that when no augmenting paths exist, the current flow is maximized.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

When weighted flow graphs are used, it becomes possible to create negative cycles in the residual graph because $c_{u,v} = -c_{v,u}$. Klein showed that a flow has minimum cost if and only if its residual graph contains no negative cost cycles [68]. This theorem leads naturally to the cost reduction method. We begin with a maximum flow and then push as much flow as possible along a negative cost cycle. By repeating this process until no negative cycles exist, we will produce the maximum flow at minimum cost.

This algorithm was designed for general flow graphs and can be efficiently implemented to solve our scheduling problem by taking advantage of properties belonging to the underlying bipartite graph. First, since every edge in the graph has unit capacity, any path from s to t will have unit flow. As a result, pushing flow along a negative cost cycle reduces to inverting the direction of each edge in that cycle in the residual graph. Second, we need not consider any edge leading to or from the source or sink. Because they have zero cost, these edges do not contribute to negative cost cycles.

The graph can be stored efficiently as a mn by n matrix. A simple initial maximum flow is $(v_{i,i}, w_i)$ for $i = 1$ to n. The residual graph will contain all unused edges and $(w_i, v_{i,i})$ for $i = 1$ to n. Each negative cost cycle can be found using a modified Floyd-Warshall algorithm. When no negative cost cycles remain, the optimal schedule can be easily produced from the residual graph.

A.3.2 Non-Idle Machines

We have assumed that all jobs were available before scheduling began. Implicit in this assumption is that all machines were idle. By modifying the weights of the edges in the graph, we can use the same method to solve this scheduling problem when machines are

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

non-idle initially.

Consider jobs J_1, J_2, and J_3 that were scheduled in order on machine 1 that was busy from time 0 to time t_1. In this case, the total completion time for these jobs would be $t_1+p_{1,1}+(t_1+p_{1,1}+p_{2,1})+(t_1+p_{1,1}+p_{2,1}+p_{3,1}) = (t_1+3p_{1,1})+(t_1+2p_{2,1})+(t_1+1p_{3,1})$. Job j scheduled k from last on machine i contributes $t_i + k \cdot p_{i,j}$ to the total completion time. Solving a non-idle instance of $R||\sum c_j$ can be accomplished by adding t_i, the time machine P_i will be available, to each node representing that machine.

Let $[\tau^k_{i,j}]$ be the matrix representing the contribution of job j if scheduled k from last on machine i. Then $\tau^k_{i,j} = t_i + k \cdot \tau_{i,k}$. The matrix representing the corresponding bipartite graph has the form

$$\begin{bmatrix} \left[\tau^1_{i,j}\right] \\ \left[\tau^2_{i,j}\right] \\ \vdots \\ \left[\tau^n_{i,j}\right] \end{bmatrix}$$

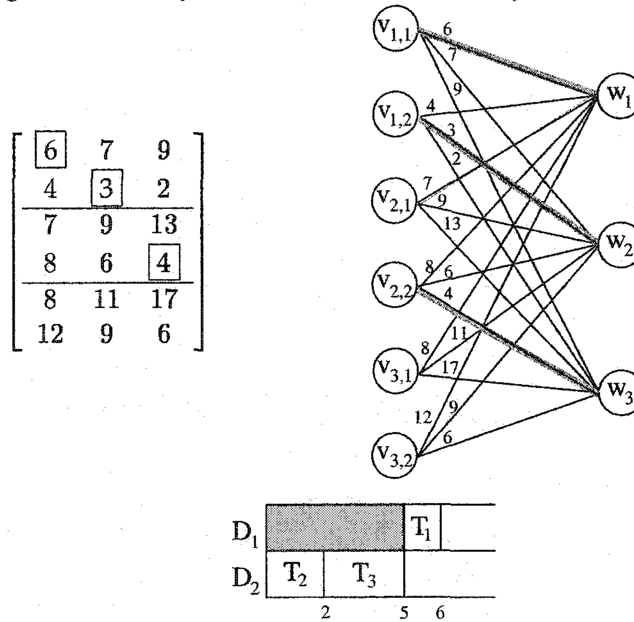
Figure A.4 shows the same three jobs in Figure A.3 scheduled on a system where the first machine does not become idle until time 5. Both the matrix and graph form of the assignment problem are shown as well as the final schedule.

A.3.3 Calculating the Non-Waiting Schedule

To calculate the non-waiting schedule, we must incorporate the lookahead job into the decision process *after* one or more of the waiting jobs are scheduled. However, the algorithm discussed in the previous two subsections calculates wait times from the end of the schedule.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure A.4: When a machine is not initially idle, the instance of the assignment problem is augmented by adding the availability time of the machine to each job.



Calculating the non-waiting schedule is done in two parts. First, we compute the non-waiting schedule of just the jobs in the wait queue, ignoring the lookahead job. Using this schedule, we can divide the waiting jobs into two sets: S_b, those that begin execution before r_{l+1} and S_a, those that begin execution after. We partially construct the non-waiting schedule by scheduling all jobs in S_b consistent with the non-lookahead schedule.

The second step is to calculate the schedule of the jobs in S_a plus the lookahead job using the partially constructed schedule as a base. Again, we use the algorithm for non-idle machines. In this case, each machine becomes available at the completion time of the latest job on that machine in the partial schedule.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

A.4 Discussion

The process of calculating the waiting and non-waiting schedules for the cross-docking model is significantly more complex than the algorithms discussed in Chapter 4. Although the algorithm described does not have to compute all possible schedules, its running time may be prohibitive in some applications. Fortunately, in the cross-docking model, these decisions can be made quickly relative to the time needed for trucks to move from the waiting area to the assigned door.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Bibliography

[1] D. AKSOY, M. ALTINEL, R. BOSE, U. CETINTEMEL, M. FRANKLIN, J. WANG, AND S. ZDONIK. Research in data broadcast and dissemination (invited paper). *Lecture Notes in Computer Science*, 1554:194–207, 1999.

[2] S. ALBERS. The influence of lookahead in competitive paging algorithms (extended abstract). In *Algorithms—ESA '93, First Annual European Symposium*, Thomas Lengauer, editor, volume 726 of *Lecture Notes in Computer Science*, pages 1–12, Bad Honnef, Germany, 30 September–2 October 1993. Springer.

[3] S. ALBERS. Better bounds for online scheduling. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 130–139, El Paso, Texas, 4–6 May 1997.

[4] S. ALBERS. On the influence of lookahead in competitive paging algorithms. *Algorithmica*, 18(3):283–305, July 1997.

[5] S. ALBERS. A competitive analysis of the list update problem with lookahead. *Theoretical Computer Science*, 197(1–2):95–109, 15 May 1998.

[6] E. J. ANDERSON AND C. N. POTTS. On-line scheduling of a single machine to minimize total weighted completion time. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 548–557. Society for Industrial and Applied Mathematics, 2002.

[7] N. AVRAHAMI AND Y. AZAR. Minimizing total flow time and total completion time with immediate dispatching. In *Proc. 15th Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 11–18. ACM, 2003.

[8] Y. AZAR AND O. REGEV. Online bin stretching. In *Proceedings of RANDOM*, pages 71–82, 1998.

[9] K.R. BAKER. *Introduction to Sequencing and Scheduling*. Wiley, New York, 1974.

[10] S. J. BEATY. Lookahead scheduling. In *Proceedings of the 25th Annual International Symposium on Microarchitecture*, Portland, Oregon, 1992.

[11] S. J. BEATY. List scheduling: Alone, with foresight, and with lookahead. In *Conference on Massively Parallel Computing Systems: the Challenges of General-Purpose and Special-Purpose Computing*, Ischia, Italy, 1994.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[12] S. BEN-DAVID AND A. BORODIN. A new measure for the study of on-line algorithms. *Algorithmica*, 11:73–91, 1994.

[13] A. BLUM. New approximation algorithms for graph coloring. *Journal of the ACM*, 41:470–516, 1994.

[14] A. BLUM AND A. KARGER. An $o(n^{3/14})$-coloring algorithm for 3-colorable graphs. *IPL: Information Processing Letters*, 61, 1997.

[15] A. BORODIN AND R. EL-YANIV. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[16] D. BRESLAUER. On competitive on-line paging with lookahead. *Theoretical Computer Science*, 209:365–375, 1998.

[17] P. BRUCKER. *Scheduling Algorithms*. Springer, Berlin, second edition, 1998.

[18] P. BRUCKER, A. DREXL, R. MÖHRING, K. NEUMANN, AND E. PESCH. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal Of Operational Research*, 112(1), 1999.

[19] J. BRUNO, JR. E. G. COFFMAN, AND R. SETHI. Scheduling independent tasks to reduce mean finishing time. *Communications of the ACM*, 17(7):382–387, 1974.

[20] S. CHAND, R. TRAUB, AND R. UZSOY. Single machine scheduling with dynamic arrivals: Decomposition results and a forward algorithm. Technical Report 93-10, School of Industrial Engineering, Purdue University, 1993.

[21] C. CHEKURI, R. MOTWANI, B. NATARAJAN, AND C. STEIN. Approximation techniques for average completion time scheduling. *SIAM Journal on Computing*, 31(1):146–166, February 2002.

[22] M. CHROBAK AND L. L. LARMORE. An optimal on-line algorithm for k servers on trees. *SIAM Journal on Computing*, 20(1):144–148, February 1991.

[23] C. CHU. Efficient heuristics to minimize total flow time with release dates. *Operations Research Letters*, 12:321–330, 1992.

[24] E. G. COFFMAN JR. *Computer and Job/Shop Scheduling Theory*. John Wiley & Sons, New York, 1976.

[25] E. G. COFFMAN JR., M. R. GAREY, AND D. S. JOHNSON. Approximation algorithms for bin packing: A survey. In *Approximation Algorithms for NP-Hard Problems*, D. S. Hochbaum, editor. PWS Publishing, Boston, 1997.

[26] B. COLEMAN AND W. MAO. Lookahead scheduling in a real-time context. In *Proceedings of the 6th International Conference on Computer Science and Informatics*, 2002.

[27] B. COLEMAN AND W. MAO. Lookahead scheduling of unrelated machines. In *Proceedings of the 7th International Conference On Computer Science and Informatics*, 2003.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[28] R. W. CONWAY, W. L. MAXWELL, AND L. W. MILLER. *Theory of Scheduling*. Addison-Wesley, Reading, MA, 1967.

[29] T.H. CORMEN, C.E LIESERSON, AND R.L. RIVEST. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, 1997.

[30] M. CRISTOFARI, M. TRONCI, F. CARON, AND E. MCDUFFIE. Interaction of control policies in a flexible manufacturing system. In *Simulation in Industry. 8th European Simulation Symposium. ESS'96. SCS*, volume 1, pages 298–302, Ghent, Belgium, 1996.

[31] J. CSIRIK AND D. S. JOHNSON. Bounded space on-line bin packing: best is better than first. In *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 309–319, 1991.

[32] J. S. DEOGUN. On scheduling with ready times to minimize mean flow time. *The Computer Journal*, 26(4):320–328, November 1983.

[33] M. I. DESSOUKY AND J. S. DEOGUN. Sequencing jobs with unequal ready times to minimize mean flow time. *SIAM Journal on Computing*, 10(1):192–202, February 1981.

[34] M. DROZDOWSKI. Scheduling multiprocessor tasks - an overview. *European Journal of Operational Research*, 94:215–230, 1996.

[35] L. EPSTEIN AND R. VAN STEE. Lower bounds for on-line singe-machine scheduling. Technical Report SEN-R0103, CWI, 2001.

[36] D. G. FEITELSON AND L. RUDOLPH. Parallel job scheduling: issues and approaches. In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph, editors, volume 949 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.

[37] A. FIAT, R. M. KARP, M. LUBY, L. A. MCGEOCH, D. D. SLEATOR, AND N. E. YOUNG. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, December 1991.

[38] G. GALAMBOS AND G. J. WOEGINGER. On-line bin packing – a restricted survey. Technical Report 280, Institut für Mathematik, Technische Universität Graz, 1993.

[39] M. R. GAREY AND D. S. JOHNSON. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.

[40] P. G. GAZMURI. Probabilistic analysis of machine scheduling. *Mathematics of Operations Research*, 10:328–339, 1985.

[41] M. X. GOEMANS. Improved approximation algorthims for scheduling with release dates. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1997.

[42] R. L. GRAHAM. Bounds on multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics*, 17(2):416–429, 1969.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[43] 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.

[44] E. F. GROVE. Online bin packing with lookahead. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 430–436, San Francisco, California, 22–24 January 1995.

[45] K. R. GUE. The effects of trailer scheduling on the layout of freight terminals. *Transportation Science*, 33(4):419–428, Nov 1999.

[46] L. A. HALL. Approximation algorithms for scheduling. In *Approximation Algorithms for NP-Hard Problems*, D. S. Hochbaum, editor. PWS Publishing, Boston, 1997.

[47] L. A. HALL, D. B. SCHULZ, AND J. WEIN. Scheduling to minimize average completion time: Off-line and on-line algorithms. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 142–151, New York/Philadelphia, January 28–30 1996. ACM/SIAM.

[48] L.A. HALL, A.S. SCHULZ, D.B. SHMOYS, AND J. WEIN. Scheduling to minimize average completion time: Off-line and on-line approximation problems. *MOR: Mathematics of Operations Research*, 22, 1997.

[49] Y. HE AND G. ZHANG. Semi on-line scheduling on two identical machines. *Computing*, 62:179–187, 1999.

[50] H. HOOGEVEEN, P. SCHUURMAN, AND G. J. WOEGINGER. Non-approximability results for scheduling problems with minsum criteria. *Lecture Notes in Computer Science*, 1412:353–362, 1998.

[51] J. A. HOOGEVEEN AND ARJEN P. A. VESTJENS. Optimal on-line algorithms for single-machine scheduling. In *Proc. 5th Conf. Integer Programming and Combinatorial Optimization*, pages 404–414, 1996.

[52] H. HYCKHOFF. A typology of cutting and packing problems. *European Journal of Operational Research*, 44:145–159, 1990.

[53] S. IRANI. Coloring inductive graphs on-line. In *Proceedings: 31st Annual Symposium on Foundations of Computer Science: October 22–24, 1990, St. Louis, Missouri*, IEEE, editor, volume 2, pages 470–479, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1990. IEEE Computer Society Press.

[54] S. IRANI. Competitive on-line algorithms for paging and graph coloring. Technical Report 92-013, International Computer Science Institute, Berkeley, CA, 1992.

[55] S. IRANI. Competitive analysis of paging. *Lecture Notes in Computer Science*, 1442:52–73, 1998.

[56] S. IRANI AND A. R. KARLIN. Online computation. In *Approximation Algorithms for NP-Hard Problems*, D. S. Hochbaum, editor. PWS Publishing, Boston, 1997.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[57] K. JANSEN AND L. PORKOLAB. Improved approximation schemes for scheduling unrelated parallel machines. In *Proceedings of the thirty-first annual ACM Symposium on Theory of Computing: Atlanta, Georgia, May 1–4, 1999*, ACM, editor, pages 408–417, New York, NY, USA, 1999. ACM Press.

[58] D.S. JOHNSON, A. DEMERS, J.D. ULLMAN, M.R. GAREY, AND R.L. GRAHAM. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3:299–325, 1974.

[59] M. Y. KAO AND S. R. TATE. Online matching with blocked input. *Information Processing Letters*, 38:113–116, 1991.

[60] A. R. KARLIN, M. S. MANASSE, L. RUDOLPH, AND D. D. SLEATOR. Competitive snoopy caching. *Algorithmica*, 3:79–119, 1988.

[61] R. M. KARP. On-line algorithms versus off-line algorithms: How much is it worth to know the future? Technical Report TR-92-044, International Computer Science Institute, July 1992.

[62] H. KELLERER, V. KOTOV, M. SPERANZA, AND Z. TUZA. Semi online algorithms for the partition problem. *Operations Research Letters*, 21:235–242, 1997.

[63] H. KELLERER, T. TAUTENHAHN, AND G. J. WOEGINGER. Approximability and non-approximability results for minimizing total flow time on a single machine. In *Proceedings of The Twenty-Eighth Annual ACM Symposium On The Theory Of Computing (STOC '96)*, pages 418–426, New York, USA, May 1996. ACM Press.

[64] P. KESKINOCAK. On-line algorithms: How much is it worth to know the future? Technical Report Research Report RC21340, IBM, 1998.

[65] P. KESKINOCAK AND W. MAO. Online algorithms: How much is it worth to know the futre? A survey in preparation.

[66] S. KHANNA, R. MOTWANI, AND R. WILSON. On certificates and lookahead in dynamic graph problems. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 222–231, 1996.

[67] J. KINIWA. Lookahead scheduling requests for multisize page caching. *IEEE Transactions on Computers*, 50(9), 2001.

[68] M. KLEIN. A primal method for minimal cost flows with applications to the assignment and transportation problems. *Management Science*, 14:205–220, 1967.

[69] C. P. KOULAMAS AND M. L. SMITH. Look-ahead scheduling for minimizing machine interference. *International Journal of Production Research*, 26(9):1523–33, Sept 1988.

[70] E. KOUTSOUPIAS AND C. H. PAPADIMITRIOU. Beyond competitive analysis. In *35th Annual Symposium on Foundations of Computer Science*, pages 394–400, Santa Fe, New Mexico, 20–22 November 1994.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[71] E.L. LAWLER, J.K. LENSTRA, A.H.G. RINNOOY KAN, AND D.B. SHMOYS. Sequencing and scheduling: Algorithms and complexity. In *Logistics of Production and Inventory*, S.C. Graves, A.H.G Rinnooy Kan, and P. Zipkin, editors, volume 4 of *Handbooks in Operations Research and Management Science*. Elsevier Science Pub. Co., North-Holland, 1990.

[72] J. K. LENSTRA, A. H. G. RINNOOY KAN, AND P. BRUCKER. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

[73] S. LEONARDI AND D. RAZ. Approximating total flow time on parallel machines. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 110–119, El Paso, Texas, 4–6 May 1997.

[74] D. LI, Y. IWAHORI, T. HAYASHI, AND N. ISHII. Enhanced look-ahead scheduling technique to overlap communication with computation. *IEICE Transactions on Information and Systems*, E81-D(11):1205–1212, November 1998.

[75] D. LI, A. MIZUNO, Y. IWAHORI, AND N. ISHII. A lookahead heuristic for heterogeneous multiprocessor scheduling with communication costs. *IEICE Transactions on Information and Systems*, E80-D(4):489–494, April 1997.

[76] W. P. LIU, J. B. SYDNEY, AND A. VAN VLIET. Ordinal algorithms for parallel machine scheduling. *Operations Research Letters*, 18:223–232, 1996.

[77] M. S. MANASSE, L. A. MCGEOCH, AND D. D. SLEATOR. Competitive algorithms for on-line problems. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 322–333, Chicago, Illinois, may 1988.

[78] W. MAO. Tight worst-case performance bounds for next-k-fit bin packing. *SIAM Journal on Computing*, 22:46–56, 1992.

[79] W. MAO AND R. K. KINCAID. A look-ahead heuristic for scheduling jobs with release dates on a single machine. *Computers Operations Research*, 21(10):1041–1050, 1994.

[80] W. MAO, R. K. KINCAID, AND A. RIFKIN. On-line single machine scheduling algorithms, chapter 8. In *The Impact of Emerging Technologies on Computer Science and Operations Research*, S. Nash and A. Sofer, editors, pages 157–173. Kluwer Academic Publisher, 1995.

[81] R. MOTWANI, V. SARASWAT, AND E. TORNG. Online scheduling with lookahead: Multipass assembly lines. *INFORMS Journal on Computing*, 10(3):331–40, 1998.

[82] C. PHILLIPS, C. STEIN, AND J. WEIN. Task scheduling in networks. *SIAM Journal on Discrete Mathematics*, 10(4):573–598, November 1997.

[83] C. PHILLIPS, C. STEIN, AND J. WEIN. Minimizing average completion time in the presence of release dates. *Mathematical Programming*, 82:199–224, 1998.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[84] S. PHILLIPS AND J. WESTBROOK. On-line algorithms: Competitive analysis and beyond. In *Algorithms and Theory of Computation Handbook*, M. J. Atallah, editor, chapter 10. CRC Press, Boca Raton, 1999.

[85] M. PINEDO. *Scheduling: Theory, Algorithms, and Systems.* Prentice Hall, New Jersey, 1995.

[86] M. E. POSNER. The deadline constrained weighted completion time problem: analysis of a heuristic. *Operations Research*, 36:742–746, 1988.

[87] P. RAMANAN, D. J. BROWN, C. C. LEE, AND D. T. LEE. On-line bin backing in linear time. *Journal of Algorithms*, 10:305–326, 1989.

[88] N. REINGOLD, J. WESTBROOK, AND D. D. SLEATOR. Randomized algorithms for the list update problem. *Algorithmica*, 11:15–32, 1994.

[89] U. K. SARKAR, P. P. CHAKRABARTI, S. GHOSE, AND S. C. DESARKAR. Improving greedy algorithms by lookahead-search. *Journal of Algorithms*, 16:1–23, January 1994.

[90] A. SCHAERF. Combining local search and look-ahead for scheduling and constraint satisfaction problems. In *Proc. of the 15th International Joint Conf. on Artificial Intelligence (IJCAI-96)*, pages 1254–1259, Nagoya, Japan, 1997. Morgan Kaufmann.

[91] R. SCHULZ AND M. SKUTELLA. Random-based scheduling: New approximations and LP lower bounds. In *RANDOM: International Workshop on Randomization and Approximation Techniques in Computer Science.* LNCS, 1997.

[92] R. SCHULZ AND M. SKUTELLA. Scheduling-LPs bear probabilities: Randomized approximations for min-sum criteria. In *ESA: Annual European Symposium on Algorithms*, 1997.

[93] S. SEIDEN, J. SGALL, AND G. WOEGINGER. Semi-online scheduling with decreasing job sizes. Technical Report KAM-DIMATIA Series 98-410, Charles University, Prague, 1998.

[94] J. SGALL. *Online Algorithms: The State of the Art*, volume 1442 of *Lecture Notes in Comput. Sci.*, chapter 9: On-line Scheduling, pages 196–231. Springer-Verlag, 1998.

[95] R. SIMHA AND A. MAJUMDAR. On lookahead in the list update problem. *Information Processing Letters*, 50:105–110, 1994.

[96] D. D. SLEATOR AND R. E. TARJAN. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[97] E. TORNG. A unified analysis of paging and caching. In *36th Annual Symposium on Foundations of Computer Science: October 23–25, 1995, Milwaukee, Wisconsin*, IEEE, editor, pages 194–203, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1995. IEEE Computer Society Press.

[98] R. VAN STEE AND H. LA POUTRÉ. Minimizing the total completion time on-line on a single machine, using restarts. *Lecture Notes in Computer Science*, 2461:872–883, 2002.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[99] B. VELTMAN, B.J. LAGEWEG, AND J.K. LENSTRA. Multiprocessor scheduling with communication delays. *Parallel Computing*, 16:173–182, 1990.

[100] T. VREDEVELD AND C. HURKENS. Experimental comparison of approximation algorithms for scheduling unrelated parallel machines. *INFORMS Journal of Computing*, 14(2):175–189, 2002.

[101] A. WINCKLER. A distributed look-ahead algorithm for scheduling interdependent tasks. In *Proceedings ISADS 93. International Symposium on Autonomous Decentralized Systems*, pages 190–197, Los Alamitos, CA, USA, 1993. IEEE Comput. Soc. Press.

[102] N. YOUNG. Competitive paging and dual-guided on-line weighted caching and matching algorithms. Technical Report 348-91, Ph. D. Thesis, Department of Computer Science, Princeton University, 1991.

[103] N E. YOUNG. Bounding the diffuse adversary. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 420–425, San Francisco, California, 25–27 January 1998.

[104] NEAL YOUNG. *Competitive paging and dual-guided on-line weighted caching and matching algorithms*. PhD thesis, Department of Computer Science, Princeton University, 1991.

[105] P. S. YU, J. L. WOLF, AND H. SHACHNAI. Design and analysis of a look-ahead scheduling scheme to support pause-resume for video-on-demand applications. *Multimedia Systems*, 3:137–149, 1995.

[106] G. ZHANG. A simple semi on-line algorithm for $P2//C_{max}$ with a buffer. *Information Processing Letters*, 61:145–148, 1997.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

VITA

Benjamin J. Coleman

Benjamin John Coleman was born in Toledo, Ohio on March 15, 1975 and grew up in Maumee, Ohio. After graduating from Maumee High School in 1993, Ben attended Ithaca College where he earned his B.S. in Computer Science in 1993. He received his M.S. degree at the College of William of Mary in 1997 and defended his dissertation in May 2004, also at the College of William and Mary.

Along with his wife and one year old son, Ben currently lives in Bethlehem, Pennsylvania. He teaches at Moravian College, also in Bethlehem, as an Assistant Professor of Computer Science.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.