ABSTRACT

Title of Document:	DEVELOPMENT AND EVALUATION OF ALGORITHMS FOR SCHEDULING TWO UNRELATED PARALLEL PROCESSORS
	Dennis D. Leber, Master of Science, 2007
Directed By:	Associate Professor, Jeffrey W. Herrmann, Department of Mechanical Engineering

Given a group of tasks and two non-identical processors with the ability to complete each task, how should the tasks be assigned to complete the group of tasks as quickly as possible? This thesis considers this unrelated parallel machine scheduling problem with the objective of minimizing the completion time of a group of tasks (the makespan) from the perspective of a local printed circuit board manufacturer. An analytical model representing the job dependent processing time for each manufacturing line is developed and actual job data supplied by the manufacturer is used for analysis. Two versions of a complete enumeration algorithm which identify the optimal assignment schedule are presented. Several classic assignment heuristics are considered with several additional heuristics developed as part of this work. The algorithms are evaluated and their performance compared for jobs built at the local manufacturing site. Finally, a cost-benefit tradeoff for the algorithms considered is presented.

DEVELOPMENT AND EVALUATION OF ALGORITHMS FOR SCHEDULING TWO UNRELATED PARALLEL PROCESSORS

By

Dennis D. Leber

Thesis submitted to the Faculty of the Graduate School of the University of Maryland, College Park, in partial fulfillment of the requirements for the degree of
Master of Science
2007

Advisory Committee:
Associate Professor, Jeffrey W. Herrmann, Chair
Associate Professor, David Bigio
Associate Professor, Peter Sandborn

© Copyright by
Dennis D. Leber
2007

Acknowledgements

I would like to thank my advisor, Dr. Jeffery Herrmann, for the ongoing support and guidance in completing this work. At times little progress seemed to be made with seemingly unobtainable and unfocused goals; however, with Dr. Herrmann's continued encouragement and advisement I was able to complete what at one time seemed like an endless venture. Others at the University of Maryland that I would like to acknowledge include Dr. Linda Schmidt who provided enthusiasm through her teaching and substantial guidance early in my graduate work, ultimately recognizing and pairing my interests with that of Dr. Herrmann. I would also like to thank Dr. Peter Sandborn and Dr. David Bigio for their participation on my advisory committee.

I would like to acknowledge my colleagues in the Statistical Engineering Division at NIST, in particular Dr. James Filliben, Dr. William Strawderman, and Mr. William Guthrie who are always willing to entertain a thought or question which often lead to time consuming, yet thought provoking discussions. Many of these thoughts made their way into this work.

I would like to thank my family and friends, in particular my Mom and Dad for instilling the invaluable morals, ethics and drive necessary to respectfully succeed in the endeavors that I embark; for providing me with the foundation to figure things out for myself, but always being there with a loving hand when needed.

And above all, I would like to thank my beautiful and loving wife, Maria, whose unparalleled love, encouragement, and laughter provide the undoubting strength to overcome life's challenges.

Table of Contents

Acknowledgements ii
Table of Contents iii
List of Tables v
List of Figures vi
Chapter 1: Introduction 1
1.1 Problem Setting 1
1.2 Description of Problem 2
1.3 Overview of Thesis 2
Chapter 2: Related Work 4
2.1 Scheduling 4
2.2 Unrelated Parallel Machines 6
2.3 Heuristics 7
2.4 Chapter Summary 8
Chapter 3: Problem Formulation and Analysis 9
3.1 Problem Description 9
3.2 Facility Description 10
3.2.1 Manufacturing Lines 10
3.2.2 Component Placement 12
3.2.3 Line Setup 13
3.3 Job and Manufacturing Processes Data 14
3.3.1 Overview 14
3.3.2 Job Build Data 16
3.3.3 Line Performance Data 16
3.4 Total Line Processing Time Model 17
3.4.1 Setup Time Model 18
3.4.2 First Board Processing Time Model 19
3.4.3 Remaining Boards Processing Time Model 20
3.4.4 Total Line Processing Time Model 20
3.4.5 Makespan 21
3.4.6 Total Line Processing Time Example 21
3.5 Scheduling Heuristic Evaluation 24
3.5.1 Problem Instance Data 24
3.5.2 Scheduling Heuristic Performance 25
3.5.3 Algorithm Performance 25
3.6 Chapter Summary 26
Chapter 4: Scheduling Algorithms 28
4.1 Formulation of Scheduling Algorithms 28
4.2 Complete Enumeration Algorithm 29
4.2.1 Overview 29
4.2.2 Complete Enumeration – Classical Approach 29
4.2.3 Complete Enumeration – Matrix Multiplication Approach 32
4.3 LPT Algorithm 37
4.4 Delta Algorithm 42

4.5 Initial Assign Algorithm 45
4.6 Ibarra-Kim Algorithm F 50
4.7 Large k Algorithm 56
4.8 Chapter Summary 61
Chapter 5: Experimental Results 63
5.1 Experimental Conditions 63
5.2 Heuristic Parameter Selection 63
5.2.1 Delta Heuristic – Choosing Δ 63
5.2.2 Initial Assign Heuristic – Choosing Φ 70
5.3 Scheduling Heuristic Performance Results 75
5.4 Algorithm Performance Results 82
5.4.1 Algorithm Results 84
5.4.2 CPU Time as a Function of Number of Jobs 91
5.5 Cost – Benefit Tradeoff 93
5.6 Chapter Summary 97
Chapter 6: Summary and Conclusions 99
6.1 Work Performed 99
6.2 Results and Conclusions 103
6.3 Future Work 107
Appendix A: Algorithm S Codes 110
References 155

.

List of Tables

Table 1: Job build variables 16
Table 2: Manufacturing process step variables 17
Table 3: Example job build data 21
Table 4: Setup time 22
Table 5: First board production time 22
Table 6: Remaining boards production time 23
Table 7: Total processing time 23
Table 8: Evaluation of one schedule permutation 30
Table 9: Evaluation of second schedule permutation 30
Table 10: Complete enumeration of all possible assignments for the five-job example 31
Table 11: Design matrix for 5 jobs, 32 assignment permutations 34
Table 12: Resulting processing times and makespans 36
Table 13: Initial assignment, LPT Sum algorithm 38
Table 14: Delta – LPT Max, Δ = 300, initial assignments 42
Table 15: LPT Max assignments 43
Table 16: Initial Assign, Φ = 2, initial assignments 47
Table 17: LPT Sum assignments 47
Table 18: Initial assignment, Ibbara-Kim Algorithm F algorithm 52
Table 19: Large *k* initial assignments 57
Table 20: Summary of five job assignment example 62
Table 21: Selected Δ parameter values 69
Table 22: Selected Φ parameter values 74
Table 23: Algorithm summary 76
Table 24: Scheduling heuristic performance (makespan ratio) statistics 78
Table 25: Algorithm performance (CPU time) statistics 86
Table 26: Cost-benefit values 95
Table 27: Dominating algorithm alternatives, 20 job instance 97

List of Figures

Figure 1: Manufacturing line layouts........ 10
Figure 2: Representation of total line processing time 18
Figure 3: Gantt chart for optimal schedule 31
Figure 4: Gantt charts for five job assignment example, LPT Sum algorithm 40
Figure 5: Gantt charts for five job assignment example, Delta – LPT Max, Δ = 300 algorithm 44
Figure 6: Gantt charts for five job assignment example, Initial Assign, Φ = 2 algorithm 49
Figure 7: Gantt charts for five job assignment example, Ibbara-Kim Alg. F 54
Figure 8: Gantt charts, displaying number of boards, for five job assignment example, Large *k* Algorithm 59
Figure 9: Standard Gantt charts for five job assignment example, Large *k* Algorithm 60
Figure 10: Histogram of the absolute differences in processing times for the entire job dataset 64
Figure 11: Typical makespan ratio response curve, Delta algorithm 65
Figure 12: Makespan ratio response curve with multiple local minima 66
Figure 13: Histogram of "best" Δ parameter values for the 10 job instance dataset, Delta LPT-Sum algorithm 67
Figure 14: Boxplots of makespan ratios for the 10 job instance dataset, Delta LPT-Sum algorithm 69
Figure 15: Typical makespan ratio response curve, Initial Assign algorithm 71
Figure 16: Makespan ratio response curve with multiple local minima 72
Figure 17: Histogram of "best" Φ parameter values for the 10 job instance dataset, Initial Assign algorithm 73
Figure 18: Boxplots of makespan ratios for the 10 job instance dataset, Initial Assign algorithm 74
Figure 19: Boxplots of makespan ratios 77
Figure 20: Empirical cumulative distribution functions, 10 jobs per instance 81
Figure 21: Empirical cumulative distribution functions, 20 jobs per instance 81
Figure 22: CPU processing time for $\sum_{i=1}^{1,000,000} i$ summation 84
Figure 23: Boxplots of CPU times 85
Figure 24: Empirical cumulative distribution functions, 10 jobs per instance 89
Figure 25: Empirical cumulative distribution functions, 20 jobs per instance 89
Figure 26: Empirical cumulative distribution functions, 10 jobs per instance, adjusted CPU Time axis 90
Figure 27: Empirical cumulative distribution functions, 20 jobs per instance, adjusted CPU Time axis 90
Figure 28: CPU time as a function of number of jobs assigned 92
Figure 29: Cost-benefit tradeoff, 10 jobs per instance 94
Figure 30: Cost-benefit tradeoff, 20 jobs per instance 95

Figure 31: Cost-benefit tradeoff curve, dominating algorithms, 20 jobs per instance 97
Figure 32: CPU time as a function of number of jobs assigned 105
Figure 33: Tradeoff curves for dominating algorithms.. 106

Chapter 1: Introduction

1.1 Problem Setting

One task of a local manufacturer is to fabricate printed circuit boards. These circuit boards consist of the board itself, functioning components attached to the board by solder, and a network of "wiring" contained either within the board or atop the board establishing communication between the components.

The circuit board assembly process involves beginning with a bare board specific to the product being manufactured, applying a coating of solder paste to precise locations on the board, connecting the required components to the board, and finally heating the entire board to set the connections. These processes are fully automated and typically only require human interaction during setup and for any components that require manual placement due to their size, packaging, or sensitivity.

The manufacturer has two production lines for assembling circuit boards. The two lines are similar but not identical. The time needed to complete a particular job on each line may differ. Given the difference in manufacturing lines and an upcoming group of jobs to be produced, one is left with the question of which line should process each job in order to complete the group of jobs as soon as possible. This question is addressed in this thesis.

1.2 Description of Problem

With minimizing the total time necessary to process a given group of jobs (the makespan) as a primary objective, this thesis considers the problem of scheduling a number of independent jobs on two unrelated parallel machines without preemption denoted by the standard three-field classification presented by Graham, Lawler, Lenstra and Rinnooy Kan [1] as $R_2| |C_{max}$. When considering the objective of a minimum makespan, it is important to note that the order the jobs are to be processed on a machine is not important, only the line to which a job is assigned. This problem is known to be NP-hard in the strong sense as even the simplest case of two identical parallel machines has been shown to be NP-hard [2].

1.3 Overview of Thesis

The goal of this thesis is to present a comparison of the performance tradeoffs (costs and benefits) for several heuristics that address the two unrelated parallel machine scheduling problem. In addition to heuristics found in the scheduling literature, several heuristics are developed as part of this work based on the actual manufacturing line characteristics and job data supplied by the circuit board manufacturer.

This thesis studies a production scheduling problem motivated by work with a specific manufacturer. However, the work has wider applicability because this type of problem can occur in a wide range of settings. The results presented here contribute to the general body of knowledge about this production scheduling problem.

In the chapters to follow, Chapter 2 discusses related work on the two unrelated parallel machine scheduling problem. Chapter 3 describes the details of the specific manufacturing setting and circuit boards used as the problem basis. Chapter 4 provides a detailed description of the development of the heuristic algorithms and their performance measures. Chapter 5 presents the algorithm simulation results and a comparison of the cost and benefits associated with the heuristics. And finally, a summary of the work performed, results obtained, and future work is presented in Chapter 6.

Chapter 2: Related Work

2.1 Scheduling

Scheduling refers to the problem of assigning limited resources to tasks that require the use of these resources over a period of time and determining when each task should be done. The resources and tasks can take on a variety of forms. Resources may be airport runways, shipping trucks, manufacturing lines, computer processors, or surgeons. The tasks may be airplane take-offs and landings, packages to be shipped, products to be built, computer codes to be run, or patients to be operated on. The goals of a schedule may also take on many forms. For example, one goal may be to minimize the completion time of the final task. Another goal may be to maximize the utilization of the available resources. The schedule chosen may have a great impact on the performance of the system and the ability of the system to meet its goals.

One may consider Euler's 1736 solution of the Seven Bridges of Königsberg [3] and the subsequent birth of mathematical graph theory to be an early formalization of scheduling problems. In the more recent history, much theoretical work has been done on the application of scheduling to the production and service problems of the times. An extensive library of notation has evolved and has been adopted by the community that captures the structure involved in scheduling models. An example of this notation is the standard three-field scheduling problems classification scheme introduced by Graham et al. in 1979 [1]. The form of this notation is $\alpha \mid \beta \mid \gamma$ where

the first field describes the processor environment; the second, the detailed task characteristics, and the third refers to the solution objective.

The processor environment can consist of, amongst others, a single processor represented by the notation 1, m identical processors in parallel (P_m), m unrelated processors in parallel (R_m), or a m processor flow shop (F_m). Task characteristics may include information on due dates (d_j) and processing times (p_j) or task restrictions and constraints such as release dates (r_j), ability for preemptions (*prmp*), and precedence constraints (*prec*). Examples of objective functions include the minimization of the makespan (C_{max}), the total lateness (L_{max}), or the total weighted completion time ($\Sigma w_j C_j$). See Pinedo [4] for an extensive list of scheduling settings and approaches.

A characteristic of the complexity of a scheduling problem is the time required to optimally solve the problem. Because the size of the problem (as given by the number of inputs) has a direct impact on the time required to solve the problem, the measure of complexity is given as a function of problem size. Algorithms that can solve problems in polynomial time are more efficient than their exponential time counterparts. A problem that cannot be solved by an algorithm of polynomial time is referred to as intractable. The class of intractable problems is referred to as NP-complete or NP-hard. See Garey and Johnson [5] for a complete discussion of the theory of NP-completeness.

2.2 Unrelated Parallel Machines

The problem to be studied in this thesis is the two unrelated parallel machine scheduling problem. This problem requires identifying the assignment that minimizes the maximum task completion time (makespan). It is known as $R_2 \mid \mid C_{max}$. In this problem there are n independent jobs, $j = 1 \ldots n$, to be processed on either one of two machines, $i = 1, 2$. The machines are in parallel, that is, the machines are independent and both machines are capable of completing each and every job.

The time that machine i requires to process job j, denoted by t_{ij}, is different for each machine. That is, t_{1j} does not necessarily equal t_{2j}. Moreover, there is no simple relationship between the two times (as there is in other parallel machine scheduling problems). Preemptions, the interrupting of a processing job to begin another job instead, are not allowed. No other restrictions, such as staggered releases and due dates apply. It is desired to assign the jobs to the machines to minimize the makespan, the completion time of the last job to leave the system.

The realization of this problem is common in the real world as minimizing the makespan usually implies high utilization of both machines and ensures the balancing of the load across machines. When considering the objective of minimizing the makespan, the order in which each machine processes the jobs assigned to it is not important. The assignment is the critical decision.

2.3 Heuristics

The unrelated parallel machine problem is NP-hard as it is a generalization of the simpler case of the two identical parallel machine problem that Karp [2] has shown to be NP-hard. Lenstra et al. [6] have shown that no polynomial time algorithm exists for the general unrelated parallel machine, minimal makespan problem that can achieve a worst-case makespan performance ratio better than $\frac{3}{2}$ unless P = NP. Due to the complexity of the unrelated parallel machine problem researchers have set out to develop heuristics, which try to find a near-optimal solution in a reasonable amount of time.

Though the problem is known to be NP-hard, search methods have been developed to obtain the optimal solution. A beam search method is presented by Ghirardi and Potts [7] that has been shown to produce good results on large instances (up to 50 machines and 1000 jobs) within a reasonable time limit. Martello et al. [8] and Stern [9] have developed branch-and-bound procedures to find the optimal assignment.

A complicated heuristic that utilizes branch-and-bound methods in addition to mixed integer linear modeling to obtain near optimal assignments is presented by Mokotoff and Jimeno [10].

Heuristics based on list scheduling rules are presented by Davis and Jaffe [11] and by Ibarra and Kim [12]. Ibarra and Kim present five heuristic algorithms for the unrelated parallel machine, minimal makespan problem. The heuristic presented that

pertains specifically to the two processor case, Algorithm F, is shown to have time complexity O(n log n) and produces a schedule with a worst-case makespan performance ratio of $\frac{(\sqrt{5}+1)}{2}$.

The Ibarra-Kim Algorithm F and the Longest Processing Time first (LPT) [4] heuristics are considered in this thesis for comparative purposes. The LPT rule is also incorporated into several developed heuristics.

2.4 Chapter Summary

This chapter provided a brief introduction to the body of research on scheduling problems. It described the problem that is considered in this thesis and reviewed the key results. Because the problem is NP-hard, we will develop new heuristics for finding near-optimal solutions and will compare them to existing heuristics.

Chapter 3: Problem Formulation and Analysis

3.1 Problem Description

The general problem being considered is as follows: given a group of not necessarily equivalent tasks and several not necessarily equivalent processors able to complete the tasks, which tasks should be assigned to which processor in order to complete the given group of tasks in as short amount of time as possible? The tasks can be thought of as shipments to be shipped, products to be built, computer codes to be run, or even patients to be operated on. The corresponding processors could be shipping trucks, manufacturing lines, computer processors, or surgeons. This problem has been formalized in the scheduling literature as the minimization of the makespan (time required to complete all tasks) for unrelated parallel processors [4].

The specific problem being considered in this thesis is a problem that stems from a local printed circuit board fabricator: given a group of jobs each consisting of one or more electronic circuit boards to be produced and two non-identical manufacturing lines, which jobs should be assigned to which manufacturing line in order to minimize the makespan? This problem requires minimizing the makespan on two unrelated parallel processors or machines.

The manufacturing process of the local circuit board fabricator has been studied and modeled. Actual historical job build data has been supplied by the manufacturer.

This model and job data is described below and is the cornerstone for the development and evaluation of the heuristics presented in this thesis.

3.2 Facility Description

3.2.1 Manufacturing Lines

As described, the manufacturing setting of the local printed circuit board fabricator consists of two non-identical parallel manufacturing lines (Figure 1). Each line is fully capable of producing each and every job manufactured, though a given line may be more efficient at producing a given job due to the attributes of the job and the functionality of the manufacturing line.

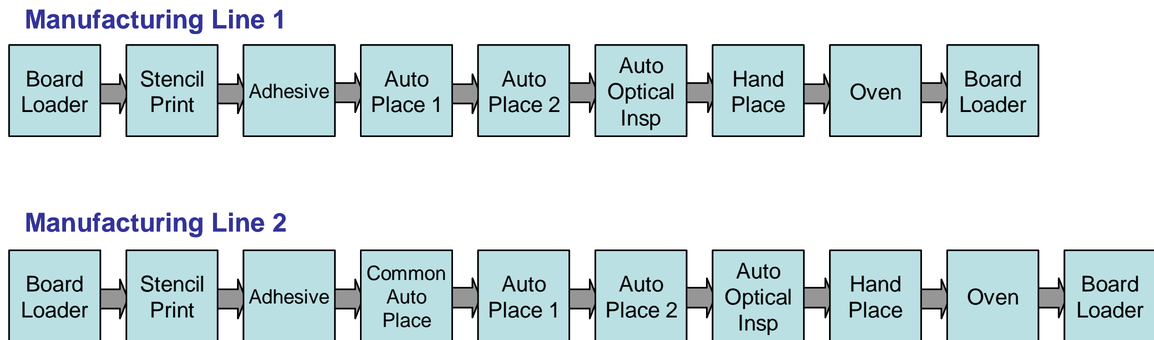

Figure 1: Manufacturing line layouts

When production runs ("jobs") consist of low quantities of boards to be produced, a majority of the total production time is spent setting up the manufacturing line for the specific product; particularly the component application portion of the manufacturing line. The manufacturer's engineers have recognized that there are approximately 200 components that are used to some degree on a majority of the products produced. These components have been permanently placed on the second of the two

manufacturing lines ("Line 2"), greatly minimizing the time required to set up the component portion of the line for products containing a large portion of these common components.

As shown in Figure 1, the first manufacturing line, called "Line 1", consists of the following automated stations connected by automatic conveyer transfers: Board Loader, Stencil Print, Loctite Adhesive Dispense, Component Auto Place 1, Component Auto Place 2, Automated Optical Inspection, Component Hand Placement, Oven, and a final Board Loader. The second manufacturing line, called "Line 2", is identical to Line 1 with the exception of the additional auto placement machine devoted to placing the circuit board components that are common to most jobs manufactured.

The boards for a job are fed into the line by the board loader that leads to the stencil print. The stencil print contains a job-specific stencil that aids in the application of solder paste which is applied using a squeegee to wipe the paste across the board. The solder paste is applied to the board precisely where the components are to be attached. A variety of components require additional adhesive which is applied to the board at the Loctite adhesive dispense station. Once the solder paste and additional adhesive have been applied to the appropriate location of the board, the component auto placement machines place the components onto the board in their correct specified locations. After the components are placed, the partially completed boards move through an automated optical inspection station. Any error identified by the

automated inspection, as well as any parts necessary for hand placement are handled by an operator at the hand placement station. The boards are then fed into the oven where the solder is heated and allowed to cure. The final board loader removes the board from the line and stacks them for customer delivery.

3.2.2 Component Placement

Most of the components used in fabricating circuit boards at this manufacturing site are placed automatically by one of the auto placement machines with relatively few being placed by hand. Of the components placed by the auto place machines, the way in which the components are packaged and delivered into the auto placement machines include:

- Tape and Reel – a delivery system that closely resembles a movie reel in which a component is held at each “frame” of the tape. Once the component is removed from the tape by the auto placement machine, the tape is advanced.
- Tube – a delivery system which resembles a large, somewhat flattened straw. The components are lined up within this tube that is inserted into the machine at an incline. When a component is removed from the end of the tube by the auto placement machine, gravity, due to the incline, advances all components within the tube.
- Tray – components in this delivery system, often larger components, are laid flat side-by-side on a tray. The auto placement machine is trained as to the location of each and every one of these components and removes them as necessary.

The approximately 200 common parts, which are permanently installed on the first auto place machine on Line 2, are all of delivery type tape and reel. Beyond this, the components required for each job are divided amongst the remaining auto place machines as follows: both the tape and reel and tube delivery systems are divided in half, with half installed on the first auto place machine (the second machine in Line 2) and half on the second auto place machine (the third in Line 2). The final auto place machine in each line places all tray delivery systems.

3.2.3 Line Setup

A number of things occur when a line is setup to run a particular job. The stencil is removed from the previous job, the lead paste applicator is cleaned, and the new stencil is inserted. The conveyers between each station are adjusted for the new board size. The oven is calibrated to the new baking temperature. Documentation of the previous job is completed while documentation of the new job is begun.

In addition to the aforementioned overall line processes involved with setting up the line for a new job, there are several job specific tasks that must be performed. Each job requires some number of components from some number of delivery systems to be added to each of the circuit boards. For example, a job may require a total of 10 components with 5 of these components coming from one tape and reel delivery system and the other 5 coming from a second tape and reel delivery system for a total of 2 component delivery systems. As each component delivery system is installed into an auto place machine, the placement head within the machine must be manually

calibrated as to the exact location of the delivery system. The time it takes to complete all setup calibrations obviously varies depending upon how many component delivery systems are necessary for each particular job.

As one can observe, a significant time effort in the set up of the job is invested and therefore preemptions are not practical and are not considered in this thesis.

3.3 Job and Manufacturing Processes Data

3.3.1 Overview

Understanding the detailed process that an entity undergoes is the first step to understanding and creating a beneficial schedule. In the case being considered, the process that each circuit board undergoes in its assembly on the different manufacturing assembly lines is of great importance in developing a schedule for distributing a group of jobs across the available manufacturing lines that will minimize the time necessary to manufacture the given set of jobs. With the process knowledge an analytical model describing the time necessary to process each entity, in this case the time necessary to process each board and ultimately each job on each manufacturing line can be created. The processing time for each entity or group of entities allows one to formulate schedules based on processing times, but more importantly allows one to calculate the response variable of direct interest, the makespan, for any given schedule.

In order to develop the analytical model describing the processing time for each entity, several sources of data are necessary. These data sources include each entity

characteristic that directly affects the processing time as well as the data describing the necessary timing for every aspect of the process of interest. In this problem setting the entity characteristics include the number of boards within a job, the number of common components on a board, and the number of different tube delivery systems needed. Data describing the manufacturing process for the circuit board manufacturing problem include the time it takes to set up the various delivery systems, the time an auto-place machine requires for the placement of an individual component, and the time it takes for the board to pass through the oven.

The entity characteristic data is well defined and forms part of the product specifications. The process timing data, on the other hand, did not exist and had to be defined and collected. In the circuit board manufacturing problem, the process of building a circuit board on each manufacturing line that a job may enter was broken down into its distinct steps, from the setup of the machines to the placement of the components on each board, to the baking and exit of the circuit boards from the manufacturing line. The time required to perform each of these distinct tasks was determined. Many of the distinct process steps are directly related to the identified board characteristics. For instance, the time it takes to place a tape and reel component is directly related to the number of tape and reel components contained on the board.

Unlike the individual entity characteristics, the process timing steps may not be absolute, but may contain variability. Including the variability in the process timing

steps in the analytical model may produce a model that is more representative of reality. However, in this circuit board problem, variability in the individual manufacture step times was observed to be small relative to the overall job processing time. Therefore, we treat and model the problem as a deterministic one.

3.3.2 Job Build Data

For the purposes of this thesis, the circuit board manufacturer supplied data for more than 425 actual jobs. These data were well defined, absolute, discrete variables. The job build variables, parameterized in Table 1, have a direct impact on the time required to complete a job and hence impact the assignment schedule to be created.

Table 1: Job build variables

Job Specific Parameters	
k	Total Number of Boards within job j
X_1	Number of Common Tape and Reel Delivery Systems
X_2	Number of Non-Common Tape and Reel Delivery Systems
X_3	Number of Tube Delivery Systems
X_4	Number of Tray Delivery Systems
X_5	Number of Hand-Place Delivery Systems
Z_1	Number of Common Tape and Reel Components
Z_2	Number of Non-Common Tape and Reel Components
Z_3	Number of Tube Components
Z_4	Number of Tray Components
Z_5	Number of Hand-Place Components

3.3.3 Line Performance Data

The distinct steps involved in the process of building a circuit board on each manufacturing line that a job may enter were observed. The parameterization and time required to perform each step is displayed in Table 2.

Table 2: Manufacturing process step variables

Line Performance Parameters		
t_1	Time to perform general setup tasks	600 sec.
t_2	Time to setup and calibrate Tape and Reel Delivery Systems	60 sec.
t_3	Time to setup and calibrate Tube Delivery Systems	60 sec.
t_4	Time to setup and calibrate Tray Delivery Systems	60 sec.
t_5	Time to setup and calibrate Hand-Place Delivery Systems	45 sec.
t_6	Time to place Tape and Reel components	3 sec.
t_7	Time to place Tube components	3 sec.
t_8	Time to place Tray components	5 sec.
t_9	Time to place Hand-Place components	10 sec.
t_{10}	Time to transfer from common autoplace machine	8 sec.
t_{11}	Time to transfer from autoplace machine 1	8 sec.
t_{12}	Time to transfer from autoplace machine 2	8 sec.
t_{13}	Time to transfer from Hand-Place	8 sec.
t_{14}	Time for Board Load (entry) and transfer	20 sec.
t_{15}	Time for Stencil Print and transfer	30 sec.
t_{16}	Time for Adhesive and transfer	30 sec.
t_{17}	Time for Auto Optical Inspect and transfer	20 sec.
t_{18}	Time for Oven and transfer	120 sec.
t_{19}	Time for Board Loader (exit)	15 sec.

3.4 Total Line Processing Time Model

Let j represent the identifier of the job considered where j can range from 1 to the total number of jobs considered, n. Let i represent the identifier of the manufacturing line considered where i can take on the values of 1 or 2.

The total processing time for any given job j on line i, can be described as the time required for the manufacturing line setup plus the time required to produce the first board plus the rate at which the remaining boards exit the line times the number of remaining boards as illustrated in Figure 2.

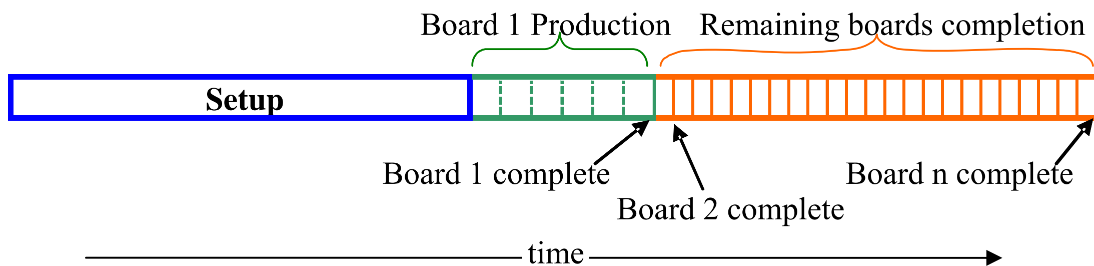

Figure 2: Representation of total line processing time

3.4.1 Setup Time Model

As previously described there are setup tasks that are required of all jobs and the time required to perform these tasks is fairly consistent from one job to another such as the cleaning of the lead paste applicator or the completion of standard job documentation. From the modeling perspective, these tasks have been combined and are represented as "general setup tasks". The fixed time required to perform the general setup tasks for any given job is $t_1 = 600$ seconds.

In addition to the general setup tasks, there are setup tasks that are job specific with the time required to perform the task dependent on the job characteristics. For example, the time required to complete the setup and calibration for the tape and reel delivery systems is dependent on the number of tape and reel delivery systems required for the given job.

With the data given in Section 3.3, the time necessary for setup for job j on line i is modeled as:

$$S_{ij} = t_1 + (2 - i)X_1 t_2 + X_2 t_2 + X_3 t_3 + X_4 t_4 + X_5 t_5$$

Note that the $(2 - i)X_1t_2$ term in the above equation adds setup time for the setup and calibration of common tape and reel delivery systems only when the job is processed on Line 1 since there is no setup time required when processed on Line 2 as these delivery systems are permanently installed on Line 2.

3.4.2 First Board Processing Time Model

The time to process an entire board, i.e. the first board, will be the processing and transfer time for the non-placement machines (board loaders, stencil print, adhesive, auto optical inspection, and oven) plus the processing time for the auto placement machines and hand placements, which are a function of the job characteristics.

With the data given in Section 3.3, the time necessary to produce the first board for job j on line i is modeled as:

$$B_{ij} = t_{14} + t_{15} + t_{16} + (i-1)(Z_1 t_6 + t_{10}) + [(2-i)Z_1 + Z_2]t_6 + t_{11} + Z_3 t_7 + Z_4 t_8 + t_{12} + t_{17} + Z_5 t_9 + t_{13} + t_{18} + t_{19}$$

Again note that the $(i - 1)$ and $(2 - i)$ terms in the Board 1 Production equation are included to control for the proper accumulation of time for the placement of common tape and reel components. If the board is processed on Line 1, there is no accumulation of time for components placed by the common auto place machine or a transfer from this machine. And likewise, if the board is processed on Line 2, an

accumulation of time for components placed by the common auto place machine and a transfer from this machine is included, but potentially less time is spent on the subsequent auto placement machines for tape and reel component placement.

3.4.3 Remaining Boards Processing Time Model

The interarrival time for the second and subsequently remaining boards to complete production is solely dependent upon the slowest task, or bottleneck, in the manufacturing process. The process bottleneck is the process which takes the maximum amount of time to complete and is equivalent to the rate at which the remaining boards are completed.

With the data given in Section 3.3, the time necessary to produce the remaining (k – 1) boards for job *j* on line i is modeled as:

$$R_{ij} = (k-1)\max\left[t_{14}, t_{15}, t_{16}, (i-1)(Z_1 t_6 + t_{10}), \frac{1}{2}[(2-i)Z_1 + Z_2]t_6 + \frac{1}{2}Z_3 t_7 + t_{11}, \frac{1}{2}[(2-i)Z_1 + Z_2]t_6 + \frac{1}{2}Z_3 t_7 + Z_4 t_8 + t_{12}, t_{17}, Z_5 t_9 + t_{13}, t_{18}, t_{19}\right]$$

3.4.4 Total Line Processing Time Model

With an understanding and model of the setup time, the time to process the first board and the time to process the remaining boards the total line processing time is modeled as illustrated in Figure 2.

The time necessary to complete any given job *j*, on line *i*, is modeled as:

$$T_{ij} = S_{ij} + B_{ij} + R_{ij}$$

The Total Line Processing Time model is used to evaluate line assignment schedules produced by the considered heuristics by calculating the required Line 1 and Line 2 processing times and the makespan of the assignment schedule.

3.4.5 Makespan

A feasible solution to this problem assigns each job to one of the two production lines. Let A_1 be the set of jobs assigned to Line 1, and let A_2 be the set of jobs assigned to Line 2. Then, the total processing time on Line i can be calculated as follows:

$$L_i = \sum_{j \in A_i} T_{ij}$$

The makespan $MS = \max \{L_1, L_2\}$.

3.4.6 Total Line Processing Time Example

Five jobs were randomly selected with replacement from the 425+ jobs supplied by the manufacturer. The job build data for the $n = 5$ jobs are displayed in Table 3.

Table 3: Example job build data

Job	k	X_1	X_2	Z_1	Z_2	X_5	Z_5	X_4	Z_4	X_3	Z_3
1	17	2	2	7	7	1	4	0	0	0	0
2	2	10	24	55	132	0	0	1	1	1	1
3	14	13	40	92	286	0	0	0	0	2	5
4	5	4	17	16	68	0	0	3	7	2	21
5	7	2	72	7	265	4	6	18	30	2	20

To calculate the total time necessary to complete a job on each manufacturing line, the setup time, the first board production time, and the remaining boards production time for each job is calculated as displayed in Table 4 through Table 6.

Table 4: Setup time

Job	Line	Setup Time Calculation	S_{ij} (sec)
1	1	$600 + 2 \cdot 60 + 2 \cdot 60 + 45 =$	885
	2	$600 + 2 \cdot 60 + 45 =$	765
2	1	$600 + 10 \cdot 60 + 24 \cdot 60 + 60 + 60 =$	2760
	2	$600 + 24 \cdot 60 + 60 + 60 =$	1320
3	1	$600 + 13 \cdot 60 + 40 \cdot 60 + 2 \cdot 60 =$	3900
	2	$600 + 40 \cdot 60 + 2 \cdot 60 =$	3120
4	1	$600 + 4 \cdot 60 + 17 \cdot 60 + 2 \cdot 60 + 3 \cdot 60 =$	2160
	2	$600 + 17 \cdot 60 + 2 \cdot 60 + 3 \cdot 60 =$	1140
5	1	$600 + 2 \cdot 60 + 72 \cdot 60 + 2 \cdot 60 + 18 \cdot 60 + 4 \cdot 45 =$	6420
	2	$600 + 72 \cdot 60 + 2 \cdot 60 + 18 \cdot 60 + 4 \cdot 45 =$	2100

Table 5: First board production time

Job	Line	First Board Production Time Calculation	B_{ij} (sec)
1	1	$20 + 30 + 30 + (7 + 7) \cdot 3 + 8 + 8 + 20 + 4 \cdot 10 + 8 + 120 + 15 =$	341
	2	$20 + 30 + 30 + 7 \cdot 3 + 8 + 7 \cdot 3 + 8 + 8 + 20 + 4 \cdot 10 + 8 + 120 + 15 =$	349
2	1	$20 + 30 + 30 + (55 + 132) \cdot 3 + 8 + 3 + 5 + 8 + 20 + 8 + 120 + 15 =$	828
	2	$20 + 30 + 30 + 55 \cdot 3 + 8 + 132 \cdot 3 + 8 + 3 + 5 + 8 + 20 + 8 + 120 + 15 =$	836
3	1	$20 + 30 + 30 + (92 + 286) \cdot 3 + 8 + 5 \cdot 3 + 8 + 20 + 8 + 120 + 15 =$	1408
	2	$20 + 30 + 30 + 92 \cdot 3 + 8 + 286 \cdot 3 + 8 + 5 \cdot 3 + 8 + 20 + 8 + 120 + 15 =$	1416
4	1	$20 + 30 + 30 + (16 + 68) \cdot 3 + 8 + 21 \cdot 3 + 7 \cdot 5 + 8 + 20 + 8 + 120 + 15 =$	609
	2	$20 + 30 + 30 + 16 \cdot 3 + 8 + 68 \cdot 3 + 8 + 21 \cdot 3 + 7 \cdot 5 + 8 + 20 + 8 + 120 + 15 =$	617
5	1	$20 + 30 + 30 + (7 + 265) \cdot 3 + 8 + 20 \cdot 3 + 30 \cdot 5 + 8 + 20 + 6 \cdot 10 + 8 + 120 + 15 =$	1345
	2	$20 + 30 + 30 + 7 \cdot 3 + 8 + 265 \cdot 3 + 8 + 20 \cdot 3 + 30 \cdot 5 + 8 + 20 + 6 \cdot 10 + 8 + 120 + 15 =$	1353

Table 6: Remaining boards production time

Job	Line	Remaining Boards Production Time Calculation	R_{ij} (sec)
1	1	$(17-1)\cdot \max\left[20,30,30,\frac{1}{2}\cdot(7+7)\cdot 3+8,\frac{1}{2}\cdot(7+7)\cdot 3+8,20,4\cdot 10+8,120,15\right]=$	1920
	2	$(17-1)\cdot \max\left[20,30,30,7\cdot 3+8,\frac{1}{2}\cdot 7\cdot 3+8,\frac{1}{2}\cdot 7\cdot 3+8,20,4\cdot 10+8,120,15\right]=$	1920
2	1	$(2-1)\cdot \max\left[20,30,30,\frac{1}{2}\cdot(55+132)\cdot 3+\frac{1}{2}\cdot 3+8,\frac{1}{2}\cdot(55+132)\cdot 3+\frac{1}{2}\cdot 3+5+8,20,8,120,15\right]=$	295
	2	$(2-1)\cdot \max\left[20,30,30,55\cdot 3+8,\frac{1}{2}\cdot 132\cdot 3+\frac{1}{2}\cdot 3+8,\frac{1}{2}\cdot 132\cdot 3+\frac{1}{2}\cdot 3+5+8,20,8,120,15\right]=$	213
3	1	$(14-1)\cdot \max\left[20,30,30,\frac{1}{2}\cdot(92+286)\cdot 3+\frac{1}{2}\cdot 5\cdot 3+8,\frac{1}{2}\cdot(92+286)\cdot 3+\frac{1}{2}\cdot 5\cdot 3+8,20,8,120,15\right]=$	7572.5
	2	$(14-1)\cdot \max\left[20,30,30,92\cdot 3+8,\frac{1}{2}\cdot 286\cdot 3+\frac{1}{2}\cdot 5\cdot 3+8,\frac{1}{2}\cdot 286\cdot 3+\frac{1}{2}\cdot 5\cdot 3+8,20,8,120,15\right]=$	5778.5
4	1	$(5-1)\cdot \max\left[20,30,30,\frac{1}{2}\cdot(16+68)\cdot 3+\frac{1}{2}\cdot 21\cdot 3+8,\frac{1}{2}\cdot(16+68)\cdot 3+\frac{1}{2}\cdot 21\cdot 3+7\cdot 5+8,20,8,120,15\right]=$	802
	2	$(5-1)\cdot \max\left[20,30,30,16\cdot 3+8,\frac{1}{2}\cdot 68\cdot 3+\frac{1}{2}\cdot 21\cdot 3+8,\frac{1}{2}\cdot 68\cdot 3+\frac{1}{2}\cdot 21\cdot 3+7\cdot 5+8,20,8,120,15\right]=$	706
5	1	$(7-1)\cdot \max\left[20,30,30,\frac{1}{2}\cdot(7+265)\cdot 3+\frac{1}{2}\cdot 20\cdot 3+8,\frac{1}{2}\cdot(7+265)\cdot 3+\frac{1}{2}\cdot 20\cdot 3+30\cdot 5+8,20,6\cdot 10+8,120,15\right]=$	3576
	2	$(7-1)\cdot \max\left[20,30,30,7\cdot 3+8,\frac{1}{2}\cdot 265\cdot 3+\frac{1}{2}\cdot 20\cdot 3+8,\frac{1}{2}\cdot 265\cdot 3+\frac{1}{2}\cdot 20\cdot 3+30\cdot 5+8,20,6\cdot 10+8,120,15\right]=$	3513

From the setup time, the first board production time, and the remaining boards production time, the total processing time for each job on each line is calculated as the sum of the components. Table 7 displays the total processing time for the five job example.

Table 7: Total processing time

Job	Line	S_{ij} (sec)	B_{ij} (sec)	R_{ij} (sec)	T_{ij} (sec)
1	1	885	341	1920	3146
	2	765	349	1920	3034
2	1	2760	828	295	3883
	2	1320	836	212.5	3208.5
3	1	3900	1408	7572.5	12880.5
	2	3120	1416	5778.5	10314.5
4	1	2160	609	802	3571
	2	1140	617	706	3243
5	1	6420	1345	3576	11341
	2	2100	1353	3513	11166

3.5 Scheduling Heuristic Evaluation

Because the problem is known to be NP-complete, we will consider a variety of scheduling heuristics to solve this parallel machine scheduling problem. These will be presented in Chapter 4. This section describes how we will evaluate the scheduling heuristics.

3.5.1 Problem Instance Data

Each of the scheduling heuristics considered in this thesis has been programmed as a computer algorithm (as described in Section 3.5.3). The input required by each algorithm is the job build data for one or more jobs as described in Section 3.3.2. The job build data for the jobs input into the algorithm is processed and a line assignment for each job based on the underlying heuristic is produced.

A typical production week for the circuit board fabricator consists of the production of 10 to 20 jobs. From the more than 425 actual job data supplied by the circuit board manufacturer two datasets of problem instances have been created and are used to evaluate the scheduling heuristics. The first dataset was created by selecting 1,000 samples, each consisting of 10 jobs randomly selected with replacement, from the supplied jobs. The second dataset also contains 1,000 samples from the jobs supplied; however each sample in this dataset consists of 20 jobs randomly selected with replacement.

3.5.2 Scheduling Heuristic Performance

The output from each scheduling heuristic is a line assignment (schedule) for the given group of jobs. The makespan of this schedule is determined using the total line processing time model developed in Section 3.4.

An attribute of this problem that provides a solid foundation for the analysis is that for any given set of jobs there is an optimal solution that minimizes the makespan. By considering all possible assignment schedules – ranging from all jobs assigned to Line 1 to some jobs assigned to each line to all jobs assigned to Line 2 – we can find a schedule that minimizes the makespan and determine the optimal makespan. We use a complete enumeration algorithm to do this.

An ideal heuristic will produce schedules whose makespan is equal to or slightly greater than the optimal makespan. For each schedule produced, the resulting makespan is divided by the optimal makespan. This ratio we call the makespan ratio.

For each of the 1,000 samples of job instances in each of the two job instance datasets, the makespan ratio is calculated for each algorithm considered. An empirical cumulative distribution function of the makespan ratios for each algorithm for each of the two job instance datasets is created and comparisons are made.

3.5.3 Algorithm Performance

The heuristic algorithms have been coded in a computer language named S. S is a non-commercial statistical programming language developed primarily by John

Chambers, Rick Becker, and Allan Wilks of Bell Laboratories [13]. The S language is a scripting language currently implemented in the commercial software package S-Plus along with the open source package R. The algorithm results presented in this thesis are a result of calculations performed in the S-Plus software package. The codes for the heuristic algorithms are presented in Appendix A.

Although the development of computer processors continues to follow Moore's Law, and hence the price and required time for computational processing continues to rapidly decrease, quickness and simplicity still rule on the manufacturing floor when schedules are being considered. The effort required to produce the line assignment schedules associated with the algorithms is measured by the required CPU processing time in S-Plus. This measure is captured within the algorithms using an internal S function.

Like the schedule performance data, the algorithm performance data is captured for each of the 1,000 samples of job instances in each of the two job instance datasets. The dependency of the algorithm's performance on the number of jobs to be assigned is explored as well as empirical cumulative distribution functions for each algorithm for each of the two job instance datasets are created and compared.

3.6 Chapter Summary

The problem considered in this thesis requires minimizing the makespan on two unrelated parallel processors. This chapter described the manufacturing setting of a local printed circuit board fabricator that has two non-identical manufacturing lines.

It presented the variables that make up the actual job build data and the manufacturing line performance parameters. It derived the total line processing model.

This chapter discussed the performance measures by which the schedules produced by the heuristics and the algorithms are to be evaluated. The scheduling heuristic performance is calculated as a ratio of the schedule makespan to the optimal makespan. The algorithm performance is captured by the S-Plus CPU processing time.

Chapter 4: Scheduling Algorithms

4.1 Formulation of Scheduling Algorithms

Scheduling algorithms can vary over a vast range of complexity, necessary data inputs, and computational efforts. A minimalist algorithm could simply assign the entities to the processors at random. This approach would be very simple, use no data inputs, and require essentially no computational efforts. On the other hand, one could develop an algorithm that incorporates all of the known characteristics of the entities and investigates all possible entity and processor combinations resulting in an optimal solution. This algorithm would be fairly complex, require a large amount of data inputs, and be computationally intensive.

Various scheduling algorithms pertaining to the unrelated parallel machine process have been developed to span this space of complexity, inputs, and computational effort. Of the algorithms considered in this thesis, the complete enumeration algorithm is the easiest to define yet the most complex and computationally intensive, but it identifies an optimal solution. Other scheduling heuristics use the processing times of the jobs on each manufacturing line. One scheduling heuristic considered uses only one of the job characteristics.

The five randomly selected jobs displayed in Table 3 of Section 3.4.6 are used throughout this chapter to illustrate the scheduling algorithms.

4.2 Complete Enumeration Algorithm

4.2.1 Overview

Let n equal the number of tasks being considered for assignment on the two processors. The complete enumeration algorithm considers all 2^n possible task-processor assignments. The task characteristics are utilized to calculate processing times on each processor and the makespan that results from each task-processor assignment. A task-processor assignment that results in the smallest possible makespan is an optimal assignment.

Two programming approaches to the complete enumeration algorithm are considered. The first considers one assignment at a time and stores the best solution and makespan found as it goes along. The second approach uses a two-level, full factorial statistical experimental design matrix and matrix multiplication to consider all scheduling assignments simultaneously.

4.2.2 Complete Enumeration – Classical Approach

The classical approach to finding an optimal assignment through complete enumeration begins with the evaluation of one possible assignment and calculation of the resulting makespan. This first assignment and makespan is set to be the optimal assignment and makespan. Table 8 displays the evaluation of one possible assignment for the five job example. In this case, the optimal assignment is set to be all jobs assigned to line 2 with an optimal makespan of 30,966 seconds.

Table 8: Evaluation of one schedule permutation

Job 1 Assignment	Job 2 Assignment	Job 3 Assignment	Job 4 Assignment	Job 5 Assignment	L_1 (sec)	L_2 (sec)	Makespan (sec)
Line 2	Line 2	Line 2	Line 2	Line 2	0.0	30966.0	30966.0

A second assignment is then evaluated and the resulting makespan is compared to the optimal makespan. If the resulting makespan is smaller than the current optimal makespan, the associated assignment and makespan replace the optimal assignment and makespan. Table 9 displays the evaluation and results of a second assignment. Since the resulting makespan of the second assignment, 27,932 seconds, is less than current optimal makespan of 30,966 seconds, the second assignment and associated makespan are set to the optimal assignment and makespan.

Table 9: Evaluation of second schedule permutation

Job 1 Assignment	Job 2 Assignment	Job 3 Assignment	Job 4 Assignment	Job 5 Assignment	L_1 (sec)	L_2 (sec)	Makespan (sec)
~~Line 2~~	~~Line 2~~	~~Line 2~~	~~Line 2~~	~~Line 2~~	~~0.0~~	~~30966.0~~	~~**30966.0**~~
Line 1	Line 2	Line 2	Line 2	Line 2	3146.0	27932.0	**27932.0**

This evaluate, compare and replace process is repeated until all 2^n possible assignments have been considered. Table 10 displays all $2^5 = 32$ schedule permutations and evaluations for the $n = 5$ job example in the order the permutations were considered. A bold entry in Table 10 indicates that the makespan and associated assignment is or once was set to the optimal assignment and makespan. Entries that are crossed out were dominated by another considered assignment. It is seen that the assignment that assigns jobs 4 and 5 to line 1 and jobs 1, 2, and 3 to line 2 is the optimal assignment with a minimum makespan of 16,557 seconds. A Gantt chart for the optimal assignment is given in Figure 3.

Table 10: Complete enumeration of all possible assignments for the five-job example

Job 1 Assignment	Job 2 Assignment	Job 3 Assignment	Job 4 Assignment	Job 5 Assignment	L_1 (sec)	L_2 (sec)	Makespan (sec)
Line 2	Line 2	Line 2	Line 2	Line 2	0.0	30966.0	**30966.0**
Line 1	Line 2	Line 2	Line 2	Line 2	3146.0	27932.0	**27932.0**
Line 2	Line 1	Line 2	Line 2	Line 2	3883.0	27757.5	**27757.5**
Line 1	Line 1	Line 2	Line 2	Line 2	7029.0	24723.5	**24723.5**
Line 2	Line 2	Line 1	Line 2	Line 2	12880.5	20651.5	**20651.5**
Line 1	Line 2	Line 1	Line 2	Line 2	16026.5	17617.5	**17617.5**
Line 2	Line 1	Line 1	Line 2	Line 2	16763.5	17443.0	**17443.0**
Line 1	Line 1	Line 1	Line 2	Line 2	19909.5	14409.0	19909.5
Line 2	Line 2	Line 2	Line 1	Line 2	3571.0	27723.0	27723.0
Line 1	Line 2	Line 2	Line 1	Line 2	6717.0	24689.0	24689.0
Line 2	Line 1	Line 2	Line 1	Line 2	7454.0	24514.5	24514.5
Line 1	Line 1	Line 2	Line 1	Line 2	10600.0	21480.5	21480.5
Line 2	Line 2	Line 1	Line 1	Line 2	16451.5	17408.5	**17408.5**
Line 1	Line 2	Line 1	Line 1	Line 2	19597.5	14374.5	19597.5
Line 2	Line 1	Line 1	Line 1	Line 2	20334.5	14200.0	20334.5
Line 1	Line 1	Line 1	Line 1	Line 2	23480.5	11166.0	23480.5
Line 2	Line 2	Line 2	Line 2	Line 1	11341.0	19800.0	19800.0
Line 1	Line 2	Line 2	Line 2	Line 1	14487.0	16766.0	**16766.0**
Line 2	Line 1	Line 2	Line 2	Line 1	15224.0	16591.5	**16591.5**
Line 1	Line 1	Line 2	Line 2	Line 1	18370.0	13557.5	18370.0
Line 2	Line 2	Line 1	Line 2	Line 1	24221.5	9485.5	24221.5
Line 1	Line 2	Line 1	Line 2	Line 1	27367.5	6451.5	27367.5
Line 2	Line 1	Line 1	Line 2	Line 1	28104.5	6277.0	28104.5
Line 1	Line 1	Line 1	Line 2	Line 1	31250.5	3243.0	31250.5
Line 2	Line 2	Line 2	Line 1	Line 1	14912.0	16557.0	**16557.0**
Line 1	Line 2	Line 2	Line 1	Line 1	18058.0	13523.0	18058.0
Line 2	Line 1	Line 2	Line 1	Line 1	18795.0	13348.5	18795.0
Line 1	Line 1	Line 2	Line 1	Line 1	21941.0	10314.5	21941.0
Line 2	Line 2	Line 1	Line 1	Line 1	27792.5	6242.5	27792.5
Line 1	Line 2	Line 1	Line 1	Line 1	30938.5	3208.5	30938.5
Line 2	Line 1	Line 1	Line 1	Line 1	31675.5	3034.0	31675.5
Line 1	Line 1	Line 1	Line 1	Line 1	34821.5	0.0	34821.5

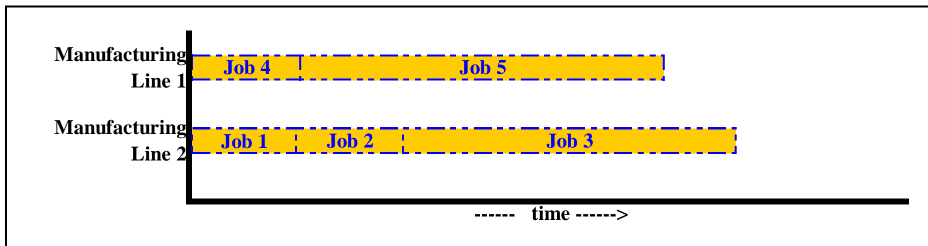

Figure 3: Gantt chart for optimal schedule

To summarize, the input to the Complete Enumeration – Classical Approach algorithm is the matrix of job build data values, ***JB***, as displayed in the 5 job example in Table 3. The algorithm considers one assignment permutation evaluation at a time

comparing and updating the optimal schedule and makespans as detailed below. The output of the algorithm is the line assignment schedule with the minimum makespan.

Algorithm CE-CA(***JB***)

1. Let $x = (x_1, \ldots, x_n)$, the line assignment for jobs 1, 2 … n

   Let A_i be the set of jobs assigned to Line i
2. Let $x = (2, 2\ldots 2)$
3. Calculate the total processing time (T_{ij}) for each job j for each line i
4. For $i = 1, 2$, calculate Line i processing time; $L_i = \sum_{j \in A_i} T_{ij}$
5. Calculate the assignment makespan $MS(x) = \max(L_1(x), L_2(x))$
6. Set optimal schedule $x^* = x$ and $MS^* = MS(x)$
7. Do $2^n - 1$ times:

   7.1. Get next assignment x = (job$_1$ assign, job$_2$ assign… job$_n$ assign)

   7.2. For $i = 1, 2$, calculate Line i processing time; $L_i = \sum_{j \in A_i} T_{ij}$

   7.3. Calculate the assignment makespan $MS(x) = \max(L_1(x), L_2(x))$

   7.4. If $MS(x) < MS^*$ then set $x^* = x$ and $MS^* = MS(x)$
8. Report x^* and MS^*

4.2.3 Complete Enumeration – Matrix Multiplication Approach

A commonly used statistical experimental design is the two-level, full factorial design. In this design two levels for each experimental factor are considered. The levels are viewed as low and high, often portrayed by "-" and "+" or "0" and "1". An experimental observation is made for all 2^n possible combinations of the two levels

for all experimental factors. Recognizing that only two processors are being considered for n tasks, the idea of the two-level, full factorial design is applied to the complete enumeration algorithm.

The complete enumeration matrix multiplication algorithm uses a linear algebra approach to consider all possible task assignments and associated makespans. A design matrix $\boldsymbol{D}$ is constructed consisting of 2^n rows representing all possible assignments and n columns representing the individual task assignments for tasks 1 through n. The entries within the design matrix will take on the values 0 and 1, with a 1 indicating the task assigned to processor 1 and a 0 indicating the task assigned to processor 2. The design matrix $\boldsymbol{D}$ for the 5 job example is displayed in Table 11. As an illustration, assignment 14 assigns jobs 1, 3, and 4 to manufacturing line 1 and jobs 2 and 5 to manufacturing line 2.

Table 11: Design matrix for 5 jobs, 32 assignment permutations

Assignment Permutation	Job 1 Assignment	Job 2 Assignment	Job 3 Assignment	Job 4 Assignment	Job 5 Assignment
1	0	0	0	0	0
2	1	0	0	0	0
3	0	1	0	0	0
4	1	1	0	0	0
5	0	0	1	0	0
6	1	0	1	0	0
7	0	1	1	0	0
8	1	1	1	0	0
9	0	0	0	1	0
10	1	0	0	1	0
11	0	1	0	1	0
12	1	1	0	1	0
13	0	0	1	1	0
14	1	0	1	1	0
15	0	1	1	1	0
16	1	1	1	1	0
17	0	0	0	0	1
18	1	0	0	0	1
19	0	1	0	0	1
20	1	1	0	0	1
21	0	0	1	0	1
22	1	0	1	0	1
23	0	1	1	0	1
24	1	1	1	0	1
25	0	0	0	1	1
26	1	0	0	1	1
27	0	1	0	1	1
28	1	1	0	1	1
29	0	0	1	1	1
30	1	0	1	1	1
31	0	1	1	1	1
32	1	1	1	1	1

Multiplying the design matrix $\boldsymbol{D}$ by the vector of task processing times on line 1, T_{1j}, results in the total processing time on processor 1, L_1, for each task assignment permutation. Similarly, the product of $\mathbf{1} - \boldsymbol{D}$ and the vector of task processing times

on line 2, T_{2j}, results in the total processing time on processor 2, L_2, for each assignment.

The maximum of the total processing time on processors 1 and 2 is the makespan for each assignment. The assignment found to have the minimum makespan is the optimal assignment. Assignment number 25 in the example considered is an optimal assignment with a makespan of 16,557 seconds as displayed in Table 12. The resulting assignment is to assign jobs 4 and 5 to line 1 and jobs 1,2, and 3 to line 2.

Table 12: Resulting processing times and makespans

Assignment Permutation	L_1 (sec)	L_2 (sec)	Makespan (sec)
1	0.0	30966.0	30966.0
2	3146.0	27932.0	27932.0
3	3883.0	27757.5	27757.5
4	7029.0	24723.5	24723.5
5	12880.5	20651.5	20651.5
6	16026.5	17617.5	17617.5
7	16763.5	17443.0	17443.0
8	19909.5	14409.0	19909.5
9	3571.0	27723.0	27723.0
10	6717.0	24689.0	24689.0
11	7454.0	24514.5	24514.5
12	10600.0	21480.5	21480.5
13	16451.5	17408.5	17408.5
14	19597.5	14374.5	19597.5
15	20334.5	14200.0	20334.5
16	23480.5	11166.0	23480.5
17	11341.0	19800.0	19800.0
18	14487.0	16766.0	16766.0
19	15224.0	16591.5	16591.5
20	18370.0	13557.5	18370.0
21	24221.5	9485.5	24221.5
22	27367.5	6451.5	27367.5
23	28104.5	6277.0	28104.5
24	31250.5	3243.0	31250.5
25	14912.0	16557.0	16557.0
26	18058.0	13523.0	18058.0
27	18795.0	13348.5	18795.0
28	21941.0	10314.5	21941.0
29	27792.5	6242.5	27792.5
30	30938.5	3208.5	30938.5
31	31675.5	3034.0	31675.5

In summary, the input to the Complete Enumeration – Matrix Multiplication Approach algorithm is the matrix of job build data values, ***JB***, as displayed in the 5 job example in Table 3. The algorithm simultaneously considers all possible assignment permutations and resulting makespans as detailed below with the output being the line assignment schedule with the minimum makespan.

Algorithm CE-MM(***JB***)

1. Calculate line processing time vectors T_1 and T_2 where $T_i = [T_{i1}, T_{i2} \dots T_{in}]$
2. Develop 2^n x n design matrix $\boldsymbol{D}$ with rows indexed by $l = 1, 2 \dots 2^n$
3. Using matrix multiplication, calculate vectors $L_1 = \boldsymbol{D}T_1$ and $L_2 = (\mathbf{1} - \boldsymbol{D})T_2$
4. Calculate vector MS with $MS_l = \max(L_{1l}, L_{2l})$
5. Report optimal assignment where $MS_{opt} = \min(MS_l)$

4.3 LPT Algorithm

The LPT algorithm is named such because it employs a Longest Processing Time first (LPT) rule [4] to assign tasks to processors. Because task processing times are needed, the LPT algorithm must utilize the detailed task characteristics and embark the total line processing time model to calculate the required processing time on processor 1 (T_{1j}) and the required processing time on processor 2 (T_{2j}). This algorithm is a scheduling heuristic because there is no guarantee that it will generate an optimal assignment.

The LPT rule considers the cumulative processing times on processors 1 and 2 for tasks currently assigned. The unassigned task with the "largest processing time" is assigned to the processor with the current minimum cumulative processing time. Since for each task there are two processing times to consider, T_{1j} and T_{2j}, three variations of "largest processing time" have been defined. The first is the sum of T_{1j} and T_{2j} which is analogous to a definition considering the average of T_{1j} and T_{2j}. This algorithm is identified as the "LPT Sum" algorithm. The second definition of "largest

processing time" used in the "LPT Max" algorithm considers the maximum of T_{1j} and T_{2j}. And the final variation and definition of "largest processing time" is the minimum of T_{1j} and T_{2j} used in the "LPT Min" algorithm.

Initially there are no tasks assigned to either processor and hence no unique minimum cumulative processing time exists as the processing times for both processors is zero. In this situation the task identified to have the "largest processing time" is assigned to the processor that processes it the quickest. Considering the 5 job example and the LPT Sum algorithm, Table 13 displays that job 3 has the "largest processing time" ($T_{1j} + T_{2j}$) with the quicker processing time occurring on Line 2, therefore job 3 is the first job assigned to Line 2.

Table 13: Initial assignment, LPT Sum algorithm

Job	T_{1j} (sec)	T_{2j} (sec)	$T_{1j} + T_{2j}$ (sec)	Line Assign
1	3146.0	3034.0	6180.0	-
2	3883.0	3208.5	7091.5	-
3	12880.5	10314.5	23195.0	2
4	3571.0	3243.0	6814.0	-
5	11341.0	11166.0	22507.0	-

Assignment step 2: With job 3 assigned to Line 2, Line 2 has a cumulative processing time of 10,314.5 seconds, and no jobs are yet assigned to Line 1. The next job will be assigned to Line 1. Of the remaining unassigned jobs, the job with "largest processing time" is job 5 with $T_{1j} + T_{2j}$ = 22,507 seconds. Job 5 is assigned to Line 1, bringing the cumulative processing time for Line 1 to 11,341 seconds.

Assignment step 3: Line 2 now has the minimum cumulative processing time of 10,314.5 seconds (versus 11,341 seconds on Line 1), hence the next job to be assigned will be assigned to Line 2. Of the now remaining 3 unassigned jobs (jobs 1, 2, and 4) job 2 has the “largest processing time” of $T_{1j} + T_{2j} = 7{,}091.5$ seconds and is assigned to Line 2. The new Line 2 cumulative processing time is 13,523 seconds.

Assignment step 4: The processing line with the minimum cumulative processing time has again flip-flopped to Line 1. Of the remaining jobs, job 4 has the “largest processing time” and will be assigned to Line 1.

Final assignment step: The final unassigned job, job 1, is assigned to the manufacturing line with the smaller cumulative processing time; in this case, Line 2. The result is an assignment that assigns jobs 4 and 5 to manufacturing line 1 for a total Line 1 processing time $L_1 = 14{,}912$ seconds and assigns jobs 1, 2, and 3 to manufacturing line 2 for a total Line 2 processing $L_2 = 16{,}557$ seconds. The makespan for this assignment is given by $MS = \max(L_1, L_2) = 16{,}557$ seconds.

Figure 4 illustrates, as Gantt charts, the assignment process of the five job example using the LPT Sum assignment algorithm. The LPT Max and LPT Min assignment algorithms follow the same logic, however the “largest processing time” definition is properly adjusted.

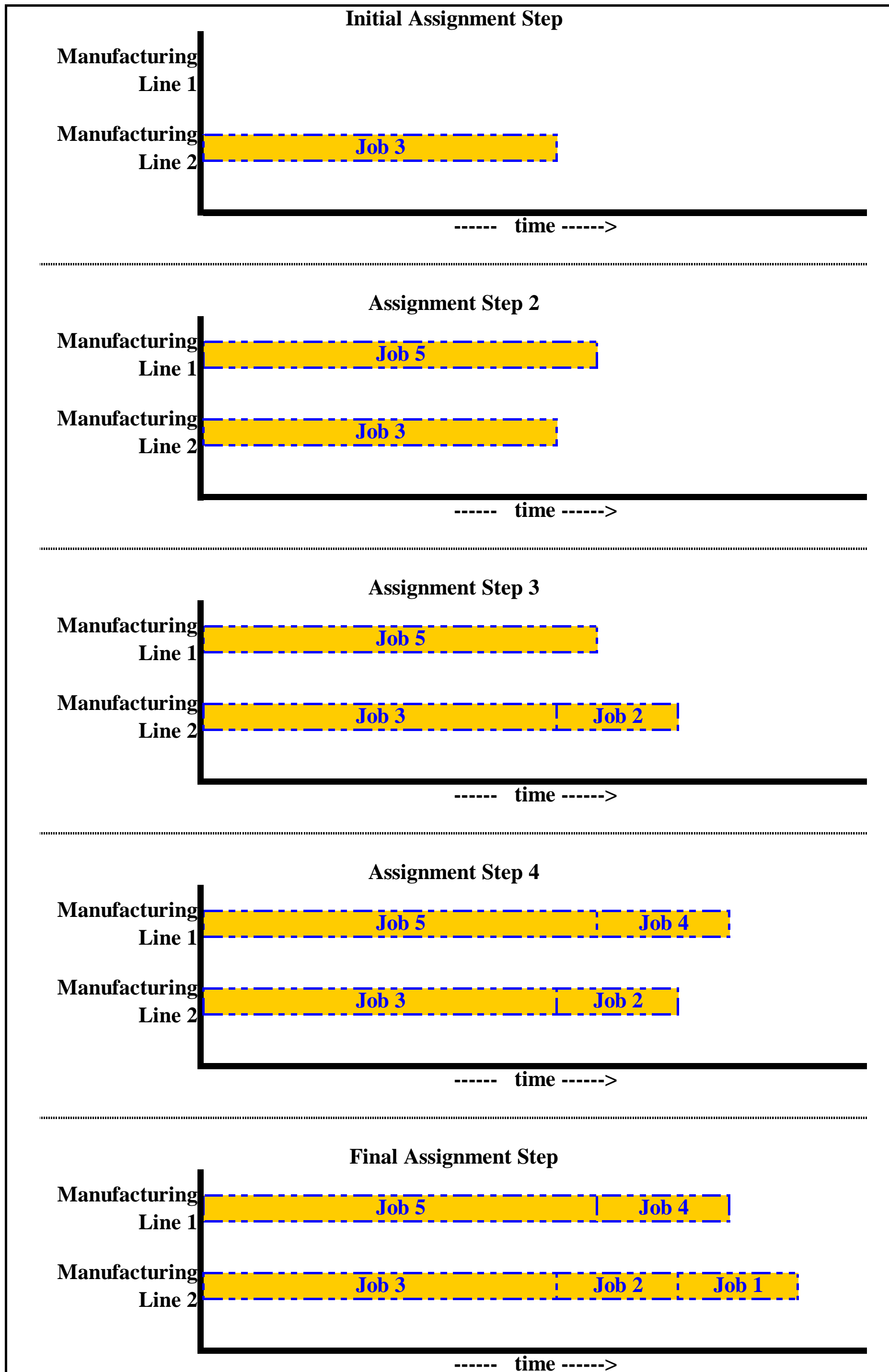

Figure 4: Gantt charts for five job assignment example, LPT Sum algorithm

In summary, the input to the LPT algorithms is the matrix of job build data values, ***JB***, as displayed in the 5 job example in Table 3. The algorithms utilize the detailed

job characteristics and the total line processing time model to calculate T_{1j} and T_{2j}. With the appropriate definition of "largest processing time", the jobs are assigned to the manufacturing lines based on the LPT rule as detailed below. The output of the algorithm is the single assignment schedule produced and its associated makespan.

Algorithm LPT Sum/Max/Min(***JB***)

1. Let A_i be the set of jobs assigned to Line i; $A_1 = A_2 = \{\}$

   Let U be the set of unassigned jobs; $U = \{1, 2 \ldots n\}$
2. Calculate line processing time vectors T_1 and T_2 where $T_i = [T_{i1}, T_{i2} \ldots T_{in}]$
3. Calculate appropriate "largest processing time" vector, $LPT_j = \{T_{1j} + T_{2j}$ or $\max(T_{1j}, T_{2j})$ or $\min(T_{1j}, T_{2j})\}$
4. Let $k = \text{argmax}(LPT_j)$ and i = $\text{argmin}(T_{1k}, T_{2k})$. Assign Job k to Line i. Add k to A_i and remove k from U.
5. Do $n - 1$ times:

   5.1. For $i = 1, 2$, calculate Line i cumulative processing time; $L_i = \sum_{j \in A_i} T_{ij}$

   5.2. Let $k = \text{argmax}(LPT_j)$ and $i = \text{argmin}(L_1, L_2)$. Assign Job k to Line i. Add k to A_i and remove k from U
6. For $i = 1, 2$, calculate Line i processing time; $L_i = \sum_{j \in A_i} T_{ij}$
7. Output A_1, A_2, and $MS = \max(L_1, L_2)$

4.4 Delta Algorithm

The delta algorithm makes use of the fact that for any task there is often a processor that processes the task more efficiently, and potentially much more efficiently. By utilizing the detailed task characteristics and total line processing time model to calculate the required processing times on processor 1 (T_{1j}) and processor 2 (T_{2j}) the delta method assigns tasks to the processor that processes the task the quickest. Those tasks with processing times on processor 1 and processor 2 that are relatively close to one another, i.e. where $|T_{1j} - T_{2j}| \leq \Delta$ seconds, are not initially assigned. For those tasks not assigned due to falling within this delta range, one of the three LPT assignment rules is employed as described in Section 4.3, thus three delta algorithms exist: Delta – LPT Sum, Delta – LPT Max, and Delta – LPT Min. The benefit provided by the initial assignment is that the number of jobs that must be considered for the more burdensome LPT assignment is reduced.

Considering the 5 job example and the Delta – LPT Max algorithm with $\Delta = 300$ seconds, Table 14 displays the initial assignments made. Jobs 2, 3, and 4 have been initially assigned to manufacturing line 2 as $|T_{1j} - T_{2j}| > 300$ and $T_{2j} < T_{1j}$.

Table 14: Delta – LPT Max, Δ = 300, initial assignments

Job	T_{1j} (sec)	T_{2j} (sec)	$\lvert T_{1j} - T_{2j}\rvert$ (sec)	Initial Line Assign
1	3146.0	3034.0	112	-
2	3883.0	3208.5	674.5	2
3	12880.5	10314.5	2566	2
4	3571.0	3243.0	328	2
5	11341.0	11166.0	175	-

The remaining unassigned jobs, jobs 1 and 5, where $|T_{1j} - T_{2j}| \leq 300$ seconds are assigned using the LPT Max assignment rule. With jobs 2, 3, and 4 already assigned to Line 2 and no jobs assigned to Line 1, the next job to be assigned will be assigned to Line 1 as it has the minimum cumulative processing time of 0 seconds. From Table 15, it is seen that job 5 will be the next job to be assigned with the "largest processing time" ($\max(T_{1j}, T_{2j})$) of the unassigned jobs. Job 5 is assigned to Line 1.

Table 15: LPT Max assignments

Job	T_{1j} (sec)	T_{2j} (sec)	$\text{Max}(T_{1j} + T_{2j})$	Line Assign
1	3146.0	3034.0	3146.0	-
2	3883.0	3208.5	3883.0	2
3	12880.5	10314.5	12880.5	2
4	3571.0	3243.0	3571.0	2
5	11341.0	11166.0	11341.0	1

The cumulative processing time on Line 1 of 11,341 seconds continues to be smaller than that of Line 2 (16,766 seconds) and hence the final job to be assigned, job 1, will again be assigned to Line 1. The result is an assignment schedule that assigns jobs 1 and 5 to manufacturing line 1 for a total Line 1 processing time $L_1 = 14{,}487$ seconds and assigns jobs 2, 3, and 3 to manufacturing line 2 for a total Line 2 processing $L_2 = 16{,}766$ seconds. The makespan for this schedule is given by $MS = \max(L_1, L_2) = 16{,}766$ seconds.

Figure 5 illustrates, as Gantt charts, the assignment process of the five job example using the Delta – LPT Max assignment algorithm with $\Delta = 300$ seconds. The Delta – LPT Sum and Delta – LPT Min assignment algorithms follow the same logic, however the "largest processing time" definition is appropriately defined.

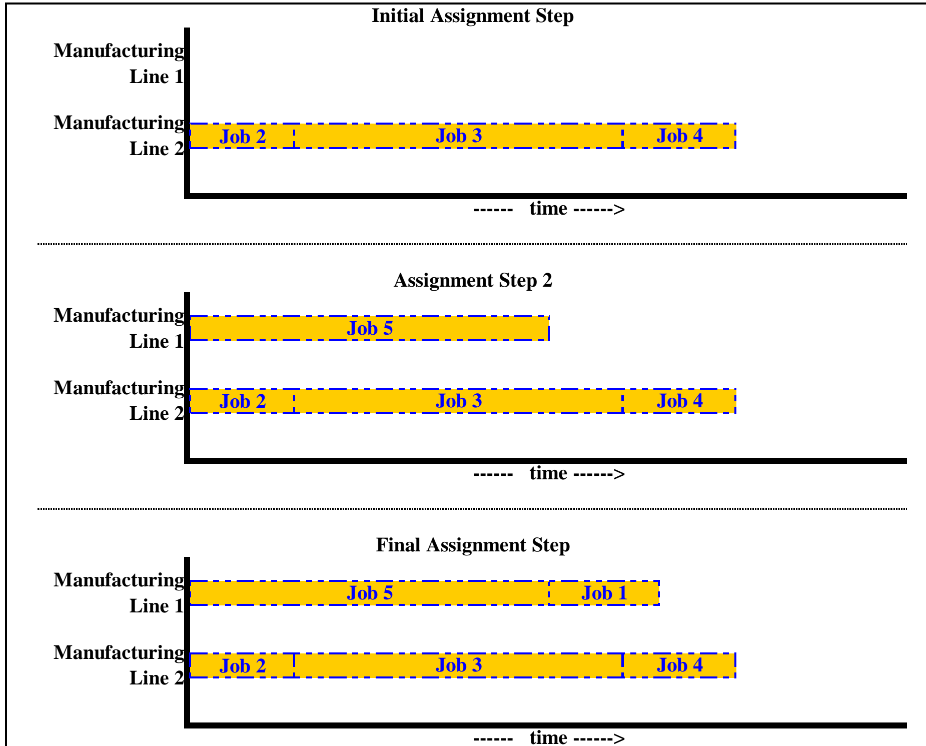

Figure 5: Gantt charts for five job assignment example, Delta – LPT Max, Δ = 300 algorithm

In summary, the input to the Delta – LPT algorithms is the matrix of job build data values, ***JB***, as displayed in the 5 job example in Table 3. The algorithms utilize the detailed job characteristics and the total line processing time model to calculate T_{1j} and T_{2j}. Jobs where $|T_{1j} - T_{2j}| > \Delta$ are initially assigned to the manufacturing line that processes them the quickest. The remaining unassigned jobs are assigned to the manufacturing lines based on the LPT rule with the appropriate definition of "largest processing time" as detailed below. The output of the algorithm is the single assignment schedule produced and its associated makespan.

Algorithm Delta – LPT Sum/Max/Min(***JB***)

1. Let A_i be the set of jobs assigned to Line i; $A_1 = A_2 = \{\}$

   Let U be the set of unassigned jobs; $U = \{1, 2 \dots n\}$

2. Calculate line processing time vectors T_1 and T_2 where $T_i = [T_{i1}, T_{i2} \dots T_{in}]$

3. For job $j = 1\dots n$, if $|T_{1j} - T_{2j}| > \Delta$, then let $i = \text{argmin}(T_{1j}, T_{2j})$, assign j to Line i, add j to A_i, and remove j from U.

4. Calculate appropriate "largest processing time" vector, $LPT_j = \{T_{1j} + T_{2j}$ or $\max(T_{1j}, T_{2j})$ or $\min(T_{1j}, T_{2j})\}$

5. If no jobs were assigned in Step 3, i.e., $A_1 = A_2 = \{\}$, then let $k = \text{argmax}(LPT_j)$ and $i = \text{argmin}(T_{1k}, T_{2k})$, assign k to Line i, add k to A_i, and remove k from U.

6. Do until U is empty:

   6.1. For $i = 1, 2$, calculate Line i cumulative processing time; $L_i = \sum_{j \in A_i} T_{ij}$

   6.2. Let $k = \text{argmax}(LPT_j : j \text{ in } U)$ and $i = \text{argmin}(L_1, L_2)$. Assign Job k to Line i. Add k to A_i and remove k from U

7. For $i = 1, 2$, calculate Line i processing time; $L_i = \sum_{j \in A_i} T_{ij}$

8. Output A_1, A_2, and $MS = \max(L_1, L_2)$

4.5 Initial Assign Algorithm

Recognizing that adjusting the continuous delta parameter in the delta algorithm leads to the adjustment in the number of jobs being initially assigned, the Initial Assign algorithm was developed. The Initial Assign algorithm takes the same approach as the delta algorithm of pre-assigning some number of jobs based on relatively simple

processing time comparisons prior to initiating a more burdensome scheduling routine. However, the Initial Assign approach simplifies the continuous and positively boundless parameter Δ to a discrete initial number of jobs parameter, Φ, bounded by zero and the number of jobs being considered, n.

Upon exploring the LPT and Delta – LPT assignment algorithms, it was also observed that the LPT Sum and Delta – LPT Sum algorithms performed better than their Max and Min counterparts on a consistent basis. Therefore, for those tasks not initially assigned by the Initial Assign algorithm, the LPT Sum assignment rule is utilized. Again, the benefit provided by the initial assignment is that the number of jobs that must be considered for the more laborious LPT assignment is reduced. An additional benefit over the similar Delta algorithm is the direct control of the number jobs to be subjected to the more burdensome LPT scheduling routine as a result of the discrete Φ parameter.

Like the LPT and Delta – LPT algorithms, the Initial Assign algorithm utilizes the detailed task characteristics and total line processing time model to calculate the required processing times on processor 1 (T_{1j}) and processor 2 (T_{2j}). The Initial Assign algorithm then considers the absolute difference in processing time on the two manufacturing lines, $|T_{1j} - T_{2j}|$, for each job. The Φ jobs with the largest absolute difference in processing time are assigned to the line that processes the job the quickest. The remaining $n - \Phi$ jobs are assigned using the LPT Sum rule as described in Section 4.3.

Considering the 5 job example and the Initial Assign algorithm with $\Phi = 2$ jobs, Table 16 displays the initial assignments made. Jobs 2 and 3 are initially assigned to manufacturing line 2 as they present the 2 largest $|T_{1j} - T_{2j}|$ and $T_{2j} < T_{1j}$ for both jobs.

Table 16: Initial Assign, Φ = 2, initial assignments

Job	T_{1j} (sec)	T_{2j} (sec)	$\|T_{1j} - T_{2j}\|$ (sec)	Initial Line Assign
1	3146.0	3034.0	112	-
2	3883.0	3208.5	674.5	2
3	12880.5	10314.5	2566	2
4	3571.0	3243.0	328	-
5	11341.0	11166.0	175	-

Assignment Step 2: The LPT Sum assignment rule is applied to the remaining unassigned jobs. With jobs 2 and 3 assigned to Line 2 and no jobs assigned to Line 1, the next job to be assigned will be assigned to Line 1 as it has the minimum cumulative processing time of 0 seconds. From Table 17, it is seen that of the unassigned jobs, job 5 will be the next job to be assigned with the "largest processing time" ($T_{1j} + T_{2j}$). Job 5 is assigned to Line 1.

Table 17: LPT Sum assignments

Job	T_{1j} (sec)	T_{2j} (sec)	$T_{1j} + T_{2j}$ (sec)	Line Assign
1	3146.0	3034.0	6180.0	-
2	3883.0	3208.5	7091.5	2
3	12880.5	10314.5	23195.0	2
4	3571.0	3243.0	6814.0	-
5	11341.0	11166.0	22507.0	1

Assignment Step 3: The cumulative processing time of Line 1 (11,341 seconds) is compared to that of Line 2 (13,523 seconds) and it is determined that the next job to be assigned will be assigned to Line 1 as it continues to have the minimum cumulative processing time. Of the remaining unassigned jobs, job 4 has the "largest processing time" and is assigned to Line 1.

Final Assignment Step: The final remaining unassigned job, job 1, is assigned to the manufacturing line with the minimum cumulative processing time, Line 2.

The result is an assignment schedule that assigns jobs 4 and 5 to manufacturing line 1 for a total Line 1 processing time L_1 = 14,912 seconds and assigns jobs 1, 2, and 3 to manufacturing line 2 for a total Line 2 processing L_2 = 16,557 seconds. The makespan for this schedule is $MS = \max(L_1, L_2)$ = 16,557 seconds. Figure 6 illustrates, as Gantt charts, the assignment process of the five job example using the Initial Assign algorithm with Φ = 2 jobs.

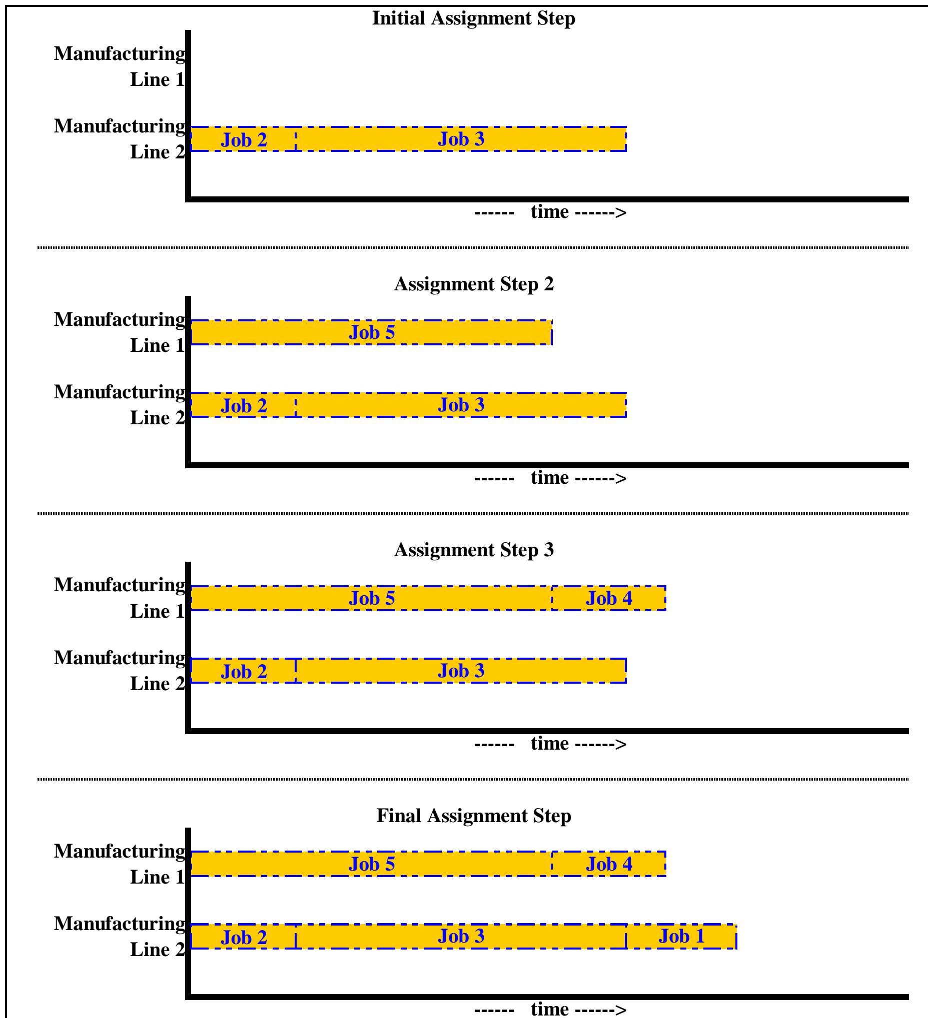

Figure 6: Gantt charts for five job assignment example, Initial Assign, Φ = 2 algorithm

In summary, the input to the Initial Assign algorithm is the matrix of job build data values, ***JB***, as displayed in the 5 job example in Table 3. The algorithm utilizes the detailed job characteristics and the total line processing time model to calculate T_{1j} and T_{2j}. A predetermined number of jobs, Φ, with the largest $|T_{1j} - T_{2j}|$ are initially assigned to the manufacturing line that processes them the quickest. The remaining unassigned jobs are assigned to the manufacturing lines based on the LPT Sum rule as detailed below. The output of the algorithm is the single assignment schedule produced and its associated makespan.

Algorithm IA(***JB***)

1. Let A_i be the set of jobs assigned to Line i; $A_1 = A_2 = \{\}$

   Let U be the set of unassigned jobs; $U = \{1, 2 \ldots n\}$
2. Calculate line processing time vectors T_1 and T_2 where $T_i = [T_{i1}, T_{i2} \ldots T_{in}]$
3. Let B be the set of Φ jobs with the largest $|T_{1j} - T_{2j}|$. Remove these from U. For each job j in B, let $i = \text{argmin}(T_{1j}, T_{2j})$, assign j to Line i, and add j to A_i.
4. For all jobs j in U, calculate "largest processing time" vector, $LPT_j = T_{1j} + T_{2j}$
5. Do until U is empty:

   5.1. For i = 1, 2, calculate Line i cumulative processing time; $L_i = \sum_{j \in A_i} T_{ij}$

   5.2. Let $k = \text{argmax}(LPT_j : j \text{ in } U)$ and $i = \text{argmin}(L_1, L_2)$. Assign Job k to Line i. Add k to A_i and remove k from U
6. For i = 1, 2, calculate Line i processing time; $L_i = \sum_{j \in A_i} T_{ij}$
7. Output A_1, A_2, and $MS = \max(L_1, L_2)$

4.6 Ibarra-Kim Algorithm F

Ibarra and Kim [12] present several heuristics for scheduling independent tasks on non-identical processors with a performance goal of minimizing the makespan. A portion of Ibarra and Kim's work focuses on the system with exactly two non-identical processors for which they present an algorithm, algorithm F, shown to perform better than the other general n-processor heuristics presented.

The Ibarra-Kim Algorithm F utilizes the detailed task characteristics and total line processing time model to calculate the required processing times on processor 1 (T_{1j}) and processor 2 (T_{2j}). The algorithm then temporarily assigns each task to the processor for which the task has a smaller processing time. If, after the initial assignment is complete, the total processing times on the two processors are equal, the assignment schedule is optimal and the algorithm is complete. If however, one processor is to remain idle while the other completes its assigned tasks, the assignment schedule may not be optimal. In this latter scenario, the Ibarra-Kim Algorithm F considers reassigning jobs from the processor with the longer total processing time (”longer processor”) to the processor with the shorter processing time (“shorter processor”) to reduce the idle period and hence reducing the makespan.

The algorithm considers reassigning one task at a time from the longer processor to the shorter processor. If the reassignment decreases the schedule’s makespan the reassignment is accepted and the assignment schedule is updated, otherwise the reassignment is rejected and the assignment schedule remains as was prior to the move. This process is repeated until all tasks on the longer processor have been considered. The order in which the tasks on the longer processor are considered for reassignment is in decreasing order of the ratio: required processing time on assigned processor divided by required processing time on non-assigned processor. The basis for this order is to decrease the total processing time on the longer processor by as much as possible while minimizing the increase in the total processing time on the shorter processor.

Table 18 displays the initial Ibarra-Kim assignment for the five job example. It is seen that all 5 jobs are assigned to line 2 since their processing times on line 2 are faster than on line 1 ($T_{2j} < T_{1j}$). With all jobs assigned to line 2, it is obvious line 2 is the longer line with a total processing time equal to the current assignment schedule makespan of 30,966 seconds. Therefore, all jobs initially assigned to line 2 will be considered for reassignment to line 1. The order in which the jobs will be considered for reassignment is dictated by the ratio of $\frac{\min(T_{1j}, T_{2j})}{\max(T_{1j}, T_{2j})}$ for each job assigned to line 2, in decreasing order as displayed in Table 18.

Table 18: Initial assignment, Ibbara-Kim Algorithm F algorithm

Job	T_{1j} (sec)	T_{2j} (sec)	Initial Assignment	$\frac{\min(T_{1j}, T_{2j})}{\max(T_{1j}, T_{2j})}$	Reassignment Order
1	3146.0	3034.0	2	0.964	2
2	3883.0	3208.5	2	0.826	4
3	12880.5	10314.5	2	0.801	5
4	3571.0	3243.0	2	0.908	3
5	11341.0	11166.0	2	0.985	1

By examining the consequence of reassigning the first job for consideration, job 5, to line 1 it is seen that the overall makespan is reduced to 19,800 seconds and hence the reassignment is accepted. Consider the second job for reassignment to line 1, job 1, it is again seen that the makespan is reduced to 16,766 seconds with the reassignment again accepted.

When the consequence for reassigning the third job to line 1, job 4, is examined it is seen that the resulting makespan of 18,058 seconds is an increase over the makespan of the previous assignment schedule (16,766 seconds); therefore the reassignment is rejected and job 4 remains assigned to line 2. When considering each of the remaining two potential reassignments, it is again seen that the makespan increases with each of these reassignments and hence the remaining jobs remain assigned to line 2.

The resulting assignment schedule for the given example by way of the Ibarra-Kim Algorithm F assigns jobs 1 and 5 to Line 1 for a total Line 1 processing time L_1 = 14,487 seconds and assigns jobs 2, 3, and 4 to Line 2 for a total Line 2 processing time L_2 = 16,766 seconds. The resulting makespan of this schedule is 16,766 seconds. Figure 7 illustrates, as Gantt charts, the assignment process of the five job example using Ibarra-Kim Algorithm F.

Figure 7: Gantt charts for five job assignment example, Ibbara-Kim Alg. F

In summary, the input to the Ibarra-Kim Algorithm F is the matrix of job build data values, ***JB***, as displayed in the 5 job example in Table 3. The algorithm utilizes the detailed job characteristics and the total line processing time model to calculate T_{1j} and T_{2j}. All jobs are assigned to the manufacturing line that processes them the quickest. All jobs on the manufacturing line with the longer resulting processing time

are considered to be reassigned to the manufacturing line with the shorter processing time. The jobs are considered for reassignment in decreasing order of the ratio $\frac{\min(T_{1j}, T_{2j})}{\max(T_{1j}, T_{2j})}$ for each job on the longer line. The reassignments are made one at a time. If the reassignment reduces the schedule makespan, then the reassignment is accepted; otherwise, the reassignment is rejected and the job remains with the original line assignment. The output of the algorithm is the single assignment schedule produced and its associated makespan.

Algorithm I-K(***JB***)

1. Let A_i be the set of jobs assigned to Line i; $A_1 = A_2 = \{\}$
2. Calculate line processing time vectors T_1 and T_2 where $T_i = [T_{i1}, T_{i2} \ldots T_{in}]$
3. For each job $j = 1\ldots n$, let $i = \text{argmin}(T_{1j}, T_{2j})$, assign j to Line i, and add j to A_i
4. For $i = 1, 2$, calculate Line i processing time; $L_i = \sum_{j \in A_i} T_{ij}$
5. Let $MS = \max(L_1, L_2)$
6. Let $MS^* = MS$
7. If $L_1 = L_2$ go to step 10. Else, let $l = \text{argmax}(L_1, L_2)$
8. For all j in A_l, calculate reassignment ratio $r_j = \frac{\min(T_{1j}, T_{2j})}{\max(T_{1j}, T_{2j})}$. Let $U = A_l$.
9. Do until U is empty:

   9.1. Let $k = \text{argmax}\ \{r_j : j \text{ in } U\}$. Move job k from A_l to A_{3-l}, and remove k from U.

9.2. For $i = 1, 2$, calculate Line i processing time; $L_i = \sum_{j \in A_i} T_{ij}$

9.3. Calculate $MS = \max(L_1, L_2)$.

9.4. If $MS < MS^*$ then set $MS^* = MS$. Otherwise, return job k from A_{3-l} to A_l (return to the previous assignment).

10. Report A_1 and A_2 and MS^*.

4.7 Large *k* Algorithm

While the algorithms presented thus far are fairly different in their scheduling approach, one thing they have in common is that they all require the knowledge of a task's processing time on each processor in order to make their assignment decisions. To obtain a task's processing time on each processor, specific characteristics of the task must be known. Additionally, an analytical model must be built to encompass the effect of the task characteristics on the processing time and interaction of the two. Aside from the variability and errors that may be introduced by this modeling process, each time one of these scheduling algorithms is to be run, the analytical model of the processor times must also be run.

In contrast, the Large *k* algorithm does not require knowledge of a task's processing time on each processor to make assignment decisions, but rather only requires one job characteristic attribute, the number of boards, k. The Large *k* algorithm assigns the job with the largest k to one of the manufacturing lines, in our case Line 2. The remaining unassigned jobs are assigned with an *LPT-type* rule. That is, the cumulative number of boards on processors 1 and 2 for jobs currently assigned are

considered. The unassigned job with the largest k is assigned to the processor with the current minimum cumulative number of boards.

Considering the 5 job example and the Large k algorithm, Table 19 displays that job 1 with the largest $k = 17$ is assigned to Line 2. Job 3 with the second largest $k = 14$ is then assigned to Line 1.

Table 19: Large *k* initial assignments

Job	*k*	Line Assign
1	17	2
2	2	-
3	14	1
4	5	-
5	7	-

Assignment Step 2: With Line 1 having the minimum cumulative number of boards (14), the next job to be assigned will be assigned to Line 1. Of the remaining unassigned jobs, job 5 has the largest $k = 7$ and is assigned to Line 1.

Assignment Step 3: Line 2 now has the minimum cumulative number of boards (17), therefore the next job to be assigned will be assigned to Line 2. Of the remaining unassigned jobs, job 4 has the largest $k = 5$ and is assigned to Line 2.

Final Assignment Step: The final remaining unassigned job, job 2 with $k = 2$ is assigned to the manufacturing line with the minimum cumulative number of boards, Line 1 with 21 boards assigned thus far.

The result is an assignment schedule that assigns jobs 2, 3, and 5 to manufacturing line 1 and jobs 1 and 4 to manufacturing line 2. Note that for evaluation purposes only, the processing time model is used to calculate the required processing times and resulting makespan. The makespan for this assignment schedule is MS = 28,104.5 seconds.

Figure 8 illustrates, as Gantt charts with number of boards as the x-axis, the assignment process of the five job example using the Large k algorithm. Figure 9 shows, with typical Gantt charts, how the processing times and makespan progressed for the assignment process.

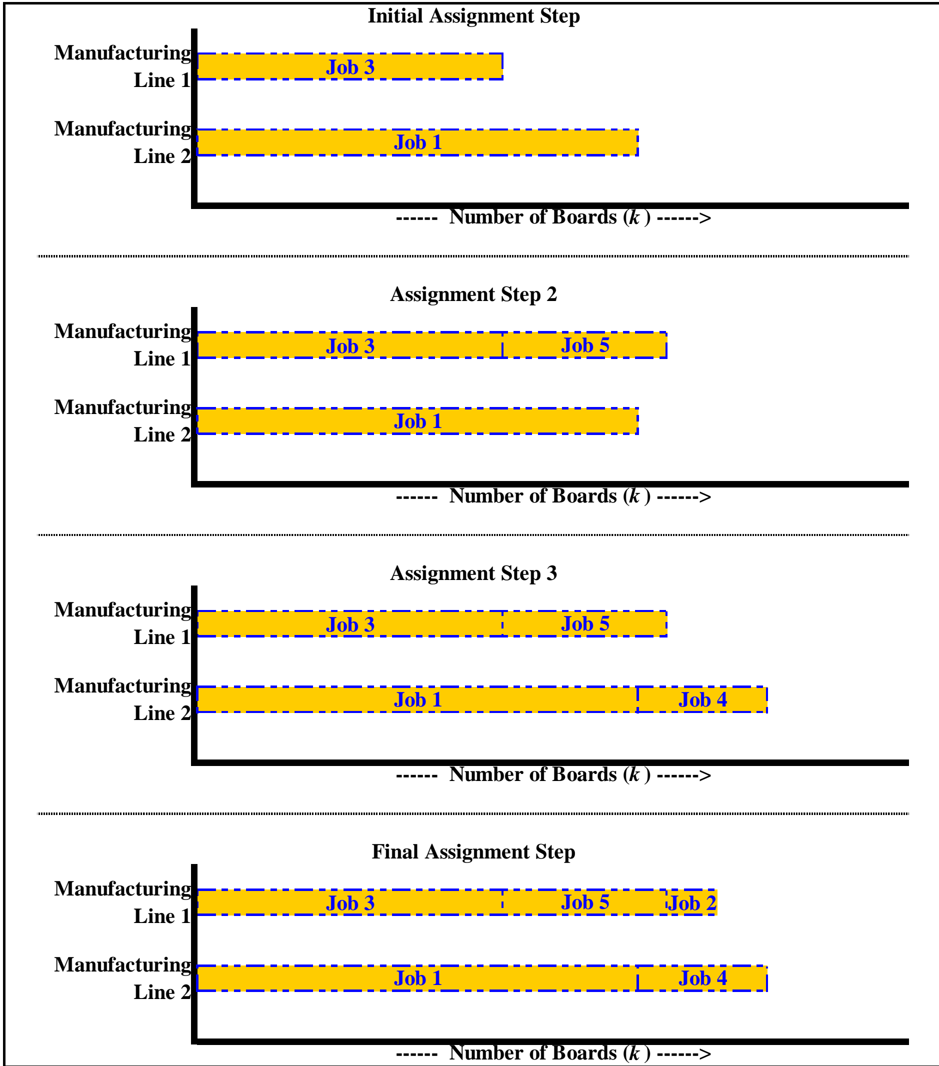

Figure 8: Gantt charts, displaying number of boards, for five job assignment example, Large *k* Algorithm

Figure 9: Standard Gantt charts for five job assignment example, Large *k* Algorithm

In summary, the Large k algorithm only requires one job attribute, the number of boards k as input. The algorithm assigns the job with the largest k to manufacturing line 2. The job with the second largest k is assigned to Line 1. The remaining unassigned jobs are assigned one-at-a-time in decreasing order of k to the manufacturing line with the minimum number of boards currently assigned as detailed below. The output of the algorithm is the single assignment schedule produced.

Algorithm L(***k***)

1. Let A_i be the set of jobs assigned to Line i; $A_1 = A_2 = \{\}$
2. Order jobs in decreasing order of k: $\{\text{Job}_{(n)}, \text{Job}_{(n\text{-}1)} \ldots \text{Job}_{(2)}, \text{Job}_{(1)}\}$
3. Assign $\text{Job}_{(n)}$ to Line 2: Add $\text{Job}_{(n)}$ to A_2
4. Assign $\text{Job}_{(n\text{-}1)}$ to Line 1: Add $\text{Job}_{(n\text{-}1)}$ to A_1
5. Let $t = 2$
6. Repeat $n - 2$ times:

   6.1. For $i = 1, 2$, calculate Line i cumulative number of boards: $N_i = \sum_{j \in A_i} k_{ij}$

   6.2. Assign $\text{Job}_{(n\text{-}t)}$ to Line i where $i = \text{argmin}(N_1, N_2)$. Add $\text{Job}_{(n\text{-}t)}$ to A_i.

   6.3. Let $t = t + 1$

7. Report A_1 and A_2

4.8 Chapter Summary

This chapter presented a number of scheduling heuristics of varying degrees of complexity to solve the unrelated parallel machine scheduling problem. With the exception of the Large k algorithm, all of the algorithms presented rely on the detailed task characteristics and total line processing time model to calculate the required processing times on processor 1 (T_{1j}) and processor 2 (T_{2j}).

Two approaches to the complete enumeration algorithm were presented. Three variations of the application of the LPT (longest processing time) rule to solve the unrelated parallel machine scheduling problem were discussed. A heuristic taken from the scheduling literature, the Ibarra-Kim Algorithm F was presented. Several

heuristics including the Delta Algorithms, Initial Assign Algorithm and the Large *k* Algorithm that were developed part of this thesis were explained in detail.

The five job example introduced in Chapter 3 was applied to each scheduling heuristic presented. Gants charts illustrating the assignment process of the algorithm were displayed. The resulting schedules and makespans for the example are shown in Table 20. Further discussion and evaluation of the scheduling heuristics pertaining to the results of the job instance datasets are presented in Chapter 5.

Table 20: Summary of five job assignment example

Heuristic	Line Assignment					Makespan (sec)
	Job 1	Job 2	Job 3	Job 4	Job 5	
Complete Enumeration	2	2	2	1	1	16,557.0
LPT Sum	2	2	2	1	1	16,557.0
Delta – LPT Max, $\Delta = 300$	1	2	2	2	1	16,766.0
Initial Assign	2	2	2	1	1	16,557.0
Ibbara-Kim, $\Phi = 2$	1	2	2	2	1	16,766.0
Large *k*	1	2	2	1	2	28,104.5

Chapter 5: Experimental Results

5.1 Experimental Conditions

Each of the 1,000 samples of job instances from both the 10 job instance dataset and the 20 job instance dataset were used as input for the algorithms described in Chapter 4. The algorithms, coded in the S scripting computer language, were run on the same laptop PC with a 2.00 GHz Intel Pentium M processor, 1.00 GB of RAM at 533 MHz, running Windows XP Professional operating system. The performance of the heuristic is captured in the makespan ratio. The cost of running the heuristic is captured in the CPU processing time. The results of these runs are presented below.

5.2 Heuristic Parameter Selection

Two of the heuristics developed as part of this work, the Delta heuristic and the Initial Assign heuristic, are parameterized heuristics. The choice in the parameter value must be made prior to running the algorithm and is seen to have a large impact on the performance of the heuristic. The following sections will discuss the approach taken in choosing the parameter value.

5.2.1 Delta Heuristic – Choosing Δ

As described in Chapter 4, the Delta heuristic makes an initial comparison of the required processing time for each task on each processor. If the absolute difference exceeds the predefined parameter Δ, then the task is assigned to the processor that processes the task the quickest. The remaining unassigned tasks are assigned using the more burdensome LPT assignment rule. The ideal Δ parameter value would be

such that the resulting makespan ratio is the minimum over all possible Δ parameter values.

The distribution of absolute differences in processing times for the entire job dataset supplied by the circuit board manufacturer was explored. It was found that this difference ranged from a minimum of 33.5 seconds to a maximum of 12,862 seconds with a mean difference of 1,444.93 seconds and a median difference of 933 seconds. A histogram of the absolute differences in processing times is displayed in Figure 10.

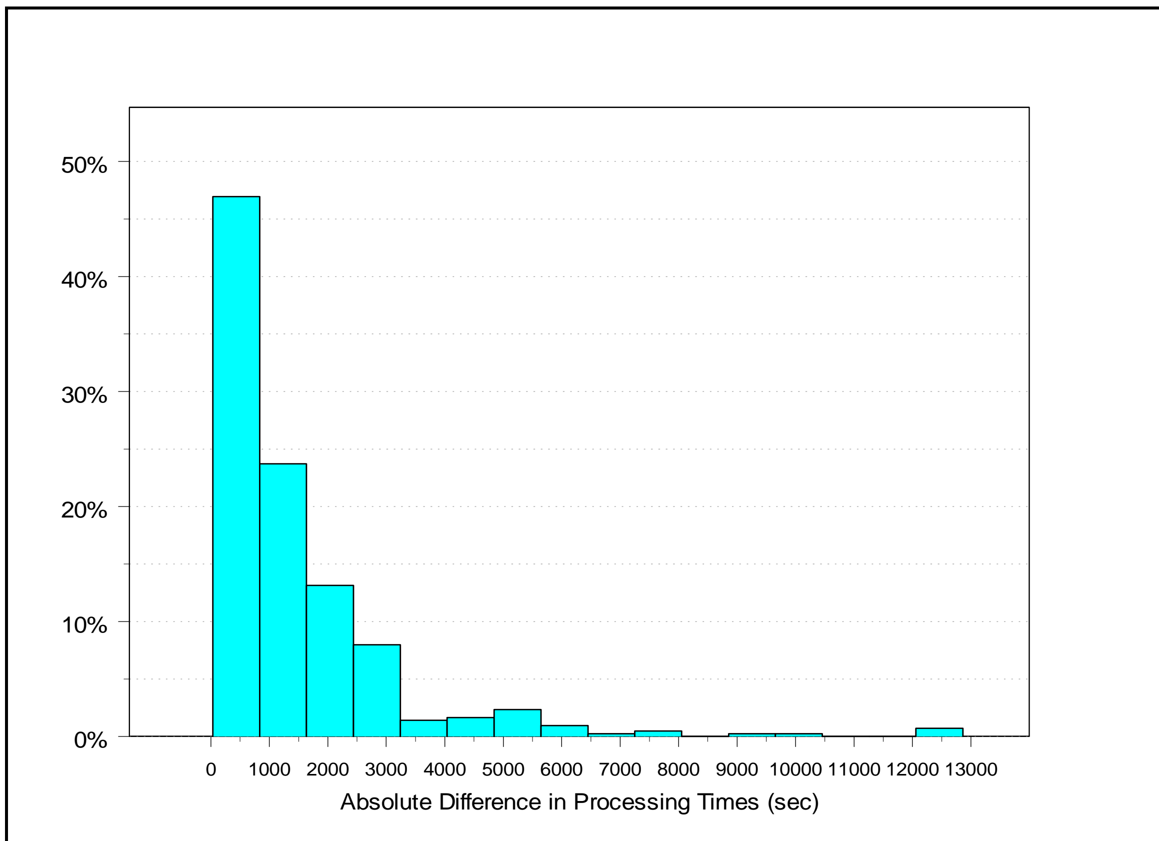

Figure 10: Histogram of the absolute differences in processing times for the entire job dataset

To assess the impact the Δ parameter has on the scheduling heuristic performance, an algorithm was developed to calculate the makespan ratios that result from a number of Δ parameter values. One hundred and thirty one Δ parameter values ranging from 0 to 13,000 seconds in 100 second intervals were explored for each of the 1,000 samples of job instances from both job instance datasets. A typical example of how the makespan ratio responded as a function of Δ is displayed in Figure 11.

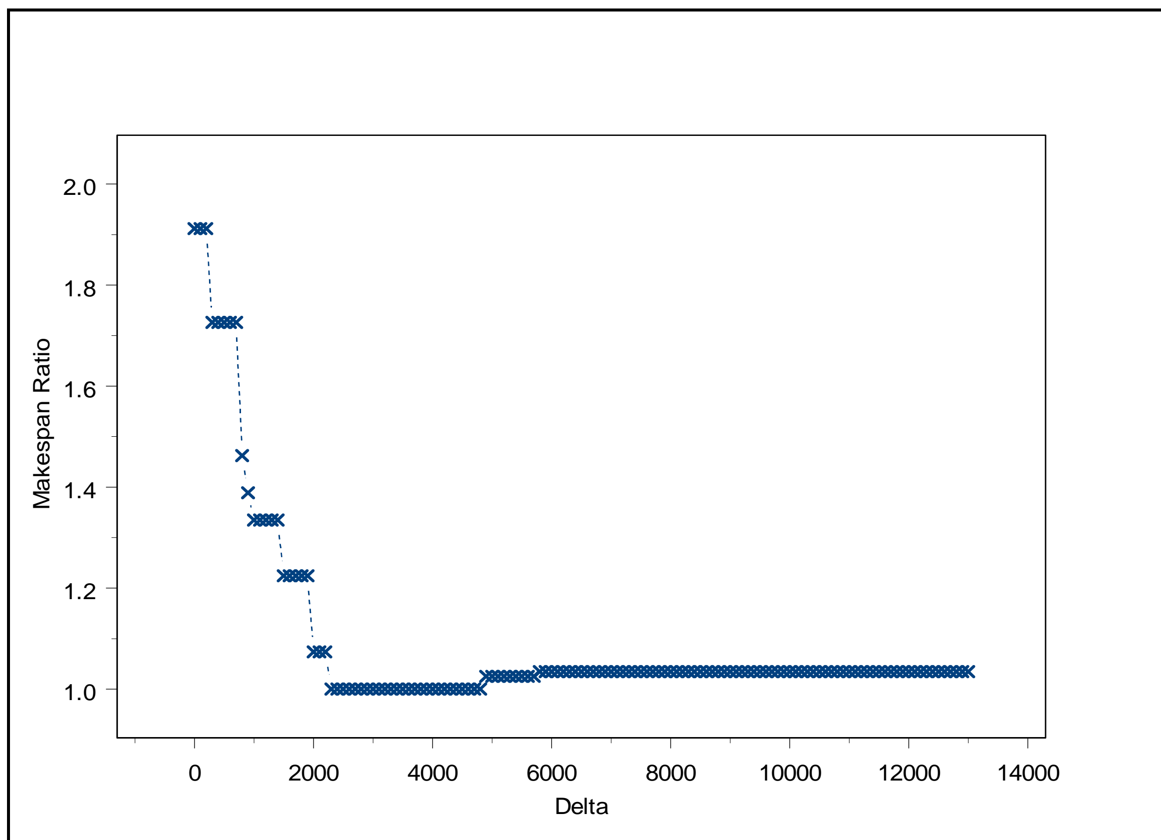

Figure 11: Typical makespan ratio response curve, Delta algorithm

It was thought that if it could be proven that the responses for all job instances resembled the shape displayed in Figure 11, in that the response function is quasi-convex and hence only has one global and one local minimum that an algorithm could

be developed that only searched the Δ parameter values until this minimum was found. It was observed that this minimum makespan ratio often occurred much closer to the minimum of the Δ parameter range than to its maximum. Therefore such a search algorithm would quickly locate the ideal delta parameter value. Unfortunately, this hypothesis of a quasi-convex response function was quickly proven false with several counter examples. Figure 12 displays one such counterexample where multiple local minima exist.

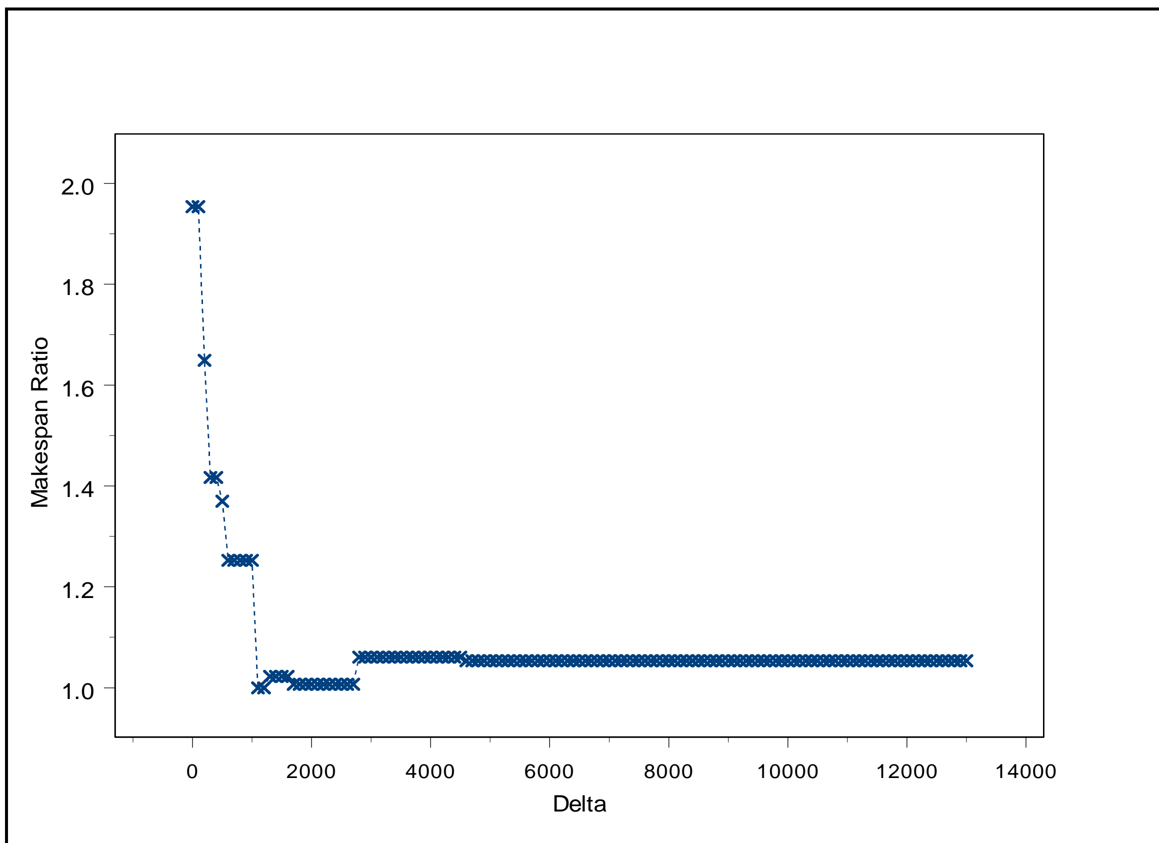

Figure 12: Makespan ratio response curve with multiple local minima

The distribution of the Δ values for which the minimum makespan ratio was observed for each of the 1,000 samples of the 10 job instance when subjected to the Delta LPT-

Sum heuristic algorithm was explored. It was found that these "best" Δ parameter values ranged from a minimum of 200 seconds to a maximum of 12,900 seconds with a mean and median of 1,816.4 seconds and 1,500 seconds respectively. A histogram of these best Δ parameter values is displayed in Figure 13.

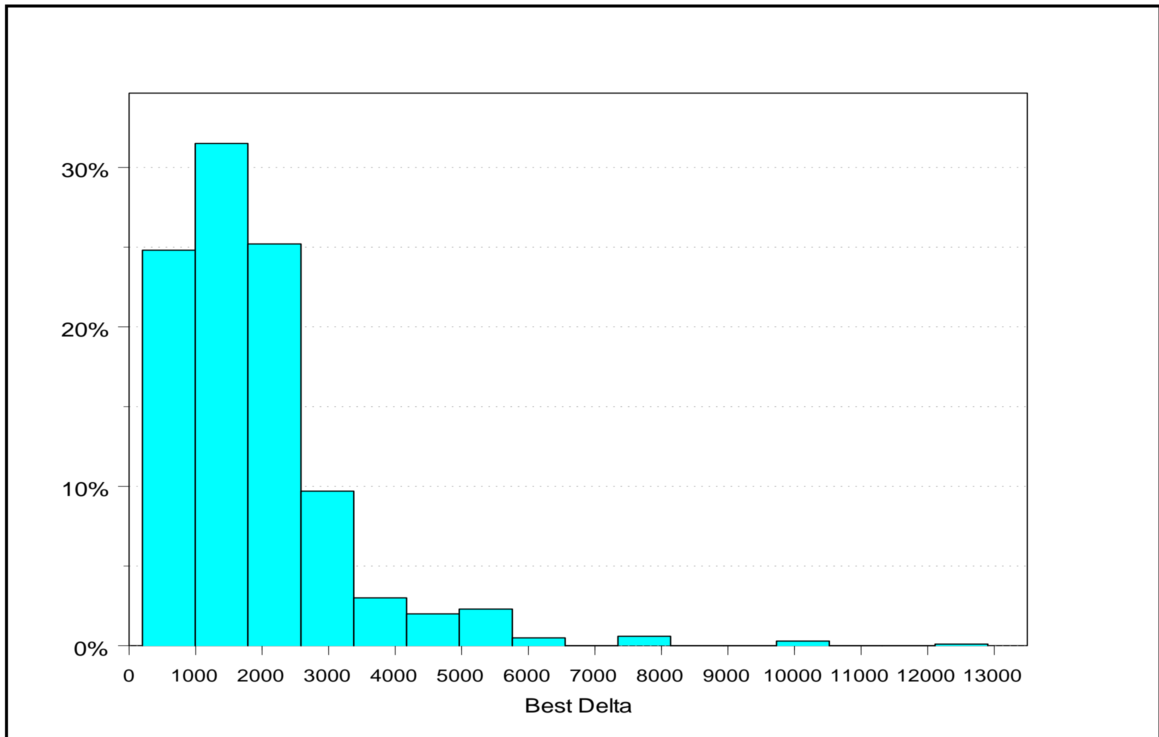

Figure 13: Histogram of "best" Δ parameter values for the 10 job instance dataset, Delta LPT-Sum algorithm

Though the mean or median of this distribution may be a natural choice for the Δ parameter value, from the shape of the makespan ratio response curves as previously displayed in Figure 11 and Figure 12, it is seen that a choice in the Δ parameter value that underestimates the best Δ parameter value has a much larger negative impact on the resulting makespan ratio than a Δ parameter value that overestimates the best Δ parameter value. Therefore, if the mean or median of the distribution is chosen as the

Δ parameter value there would be at least 50% of the job instances for which the best Δ parameter value has been underestimated resulting in a much poorer heuristc performance than necessary.

The distributions of the makespan ratio across the 1,000 sample job instances for each of the 131 Δ parameter values considered were explored. Box plots displaying the minimum, maximum, median, interquartile range, and any outliers [14] for the 10 job instance dataset using the Delta LPT-Sum algorithm are displayed in Figure 14. The Δ parameter value that produced the smallest 90th-percentile point for the makespan ratio across the 1,000 job instances was chosen as the Δ parameter value to be used in the heuristic comparisons in this thesis. By choosing this Δ parameter value for each Delta LPT algorithm, the heuristic assignment performance will be at least as good as this 90th-percentile point for 90% of the job instances considered. Table 21 displays the Δ parameter values chosen and associated 90th-percentile makespan ratios for each of the Delta LPT heuristics.

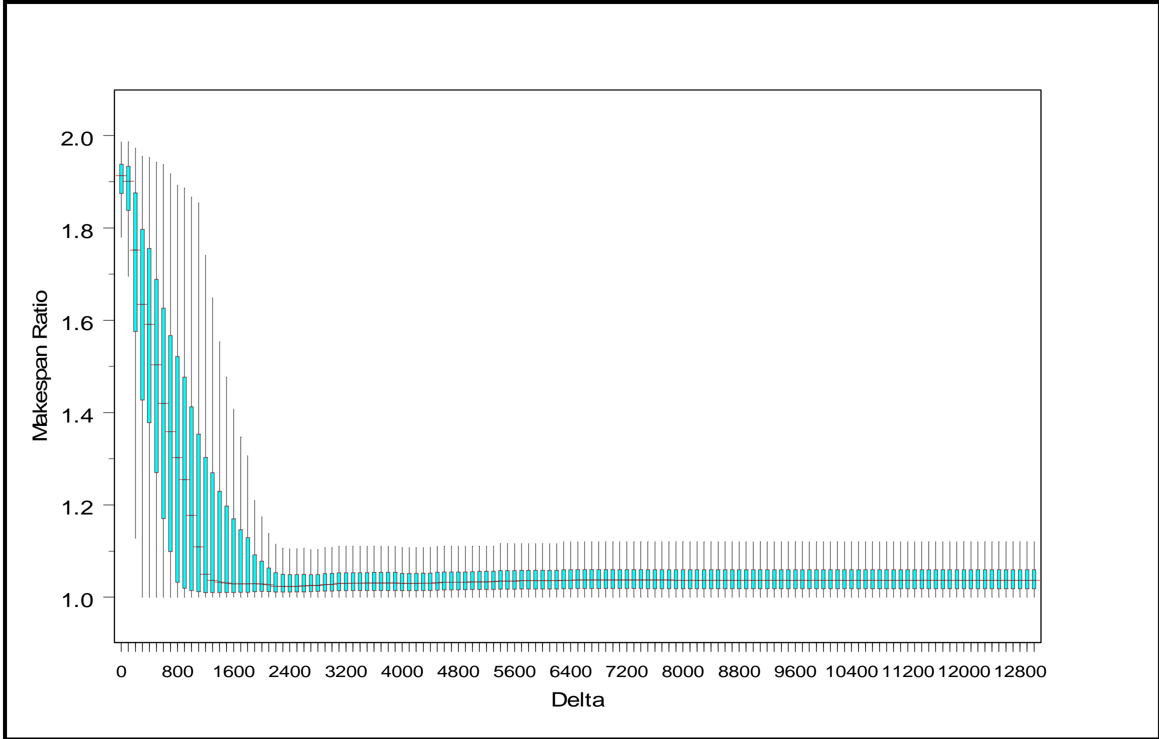

Figure 14: Boxplots of makespan ratios for the 10 job instance dataset, Delta LPT-Sum algorithm

Table 21: Selected Δ parameter values

Number of Jobs	Heuristic	Δ (sec)	Makespan Ratio 90th-percentile
10	Delta LPT-Sum	4,000	1.074
	Delta LPT-Max	4,000	1.074
	Delta LPT-Min	2,700	1.321
20	Delta LPT-Sum	2,700	1.046
	Delta LPT-Max	2,700	1.047
	Delta LPT-Min	3,500	1.184

Though it is known that searching the Δ parameter space will have an adverse effect on the algorithm performance measure of CPU time, the resulting scheduling heuristic performance measure, the makespan ratio, will be the best possible. To understand the best possible scheduling heuristic performance for the Delta LPT heuristics, the search algorithms that identify the best Δ parameter value, makespan

ratio, and assignment (Best Delta algorithms) are included in the algorithm comparisons made in this thesis.

5.2.2 Initial Assign Heuristic – Choosing Φ

The Initial Assign heuristic, as described in Chapter 4, is a simplification of the Delta heuristics. Like the Delta algorithms, the Initial Assign algorithm begins by making an initial decision based on the comparison of the required processing time for each task on each processor. Rather than using a continuous parameter in the decision process, such as Δ, the Initial Assign algorithm uses a discrete parameter, Φ. The parameter Φ is the number of jobs to be initially assigned, ranging from 0 to the total number of jobs being considered. The Φ jobs having the largest absolute difference in processing time are assigned to the processor that processes those tasks the quickest. The remaining unassigned tasks are assigned using the LPT-Sum assignment rule. The ideal Φ parameter value would be such that the resulting makespan ratio is the minimum over all possible Φ parameter values.

As mentioned, the parameter Φ can take on discrete values ranging from 0 to the total number of jobs being considered. As was done when considering the Δ parameter, an algorithm was developed to calculate the makespan ratios that result from a number of Φ parameter values to assess the impact the Φ parameter has on the scheduling heuristic performance. In fact, since the Φ parameter can take on only a limited number of values, all possible Φ parameter values are explored with this algorithm.

A typical example of how the makespan ratio responded as a function of Φ is displayed in Figure 15.

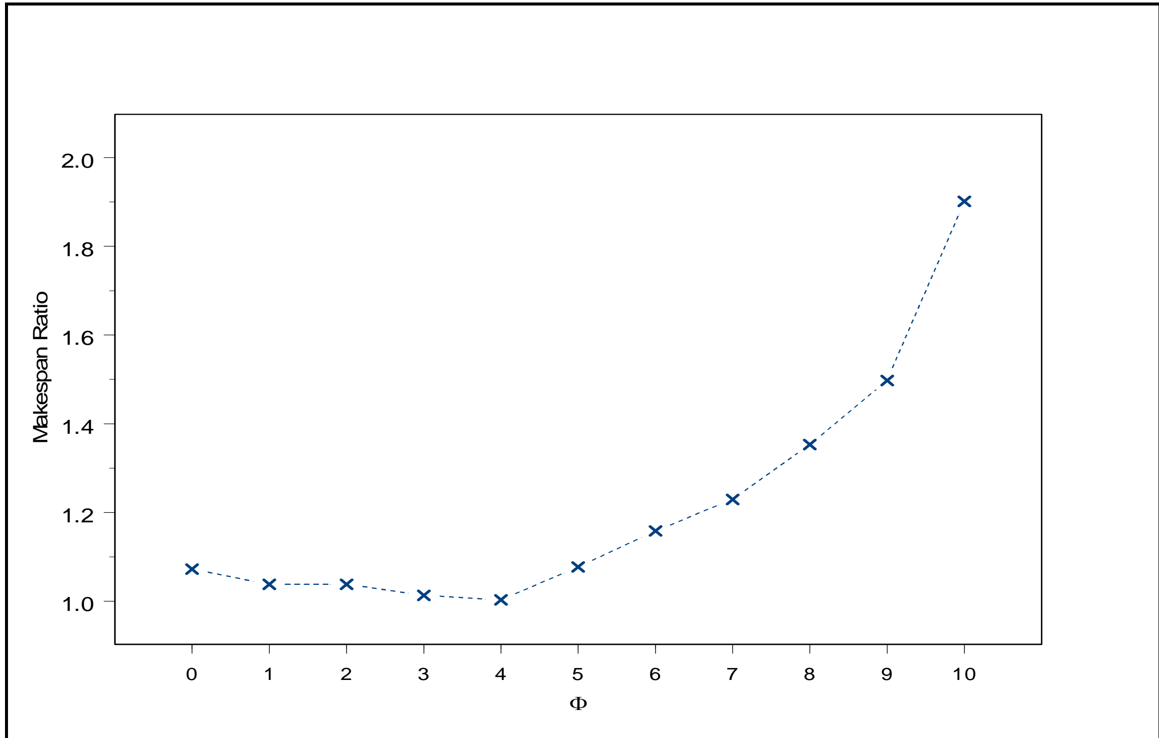

Figure 15: Typical makespan ratio response curve, Initial Assign algorithm

As with the exploration of the Δ parameter, the idea of a quasi-convex response function was considered. And again, the theory that all response functions for all job instances were quasi-convex was proven false with several counter examples. Figure 16 displays one such counterexample where multiple local minima exist.

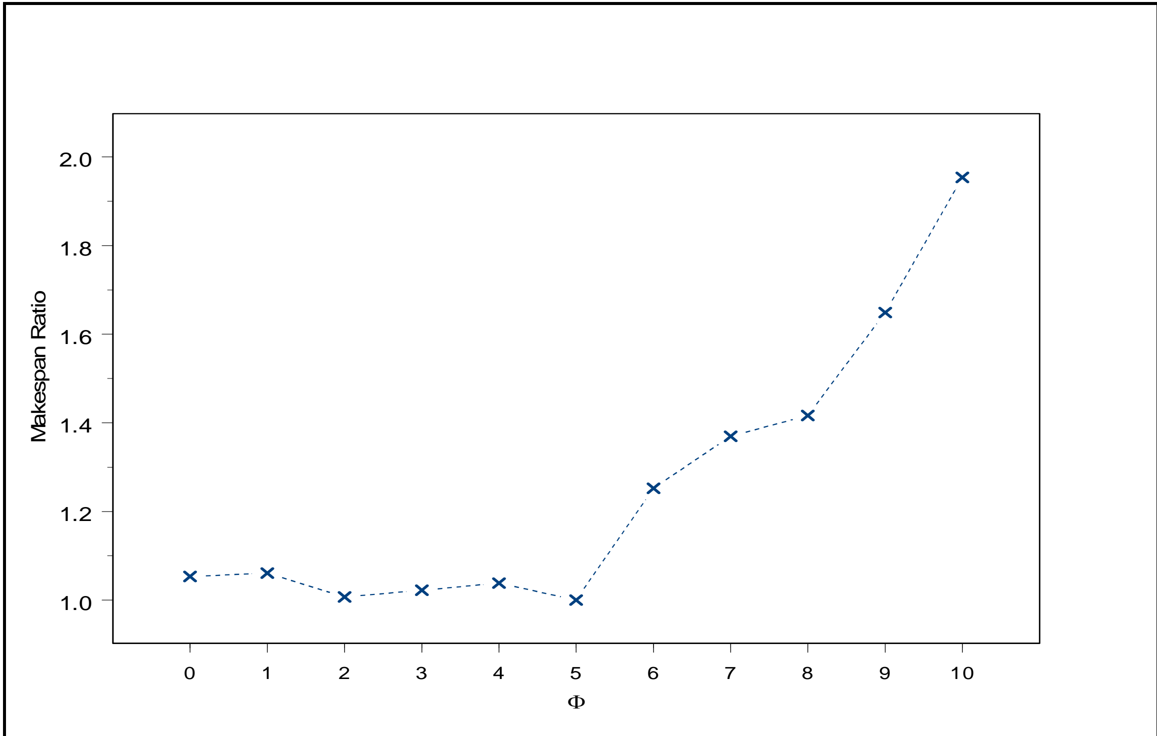

Figure 16: Makespan ratio response curve with multiple local minima

The distribution of the Φ values for which the minimum makespan ratio was observed for each of the 1,000 samples of the 10 job instance when subjected to the Initial Assign heuristic algorithm was explored. It was found that these "best" Φ parameter values ranged from a minimum of 0 jobs initially assigned to a maximum of 7 jobs initially assigned with a mean and median of 2.249 and 2 jobs respectively. A histogram of the best Φ parameter values for the 10 job instance dataset when subjected to the Initial Assign algorithm is displayed in Figure 17.

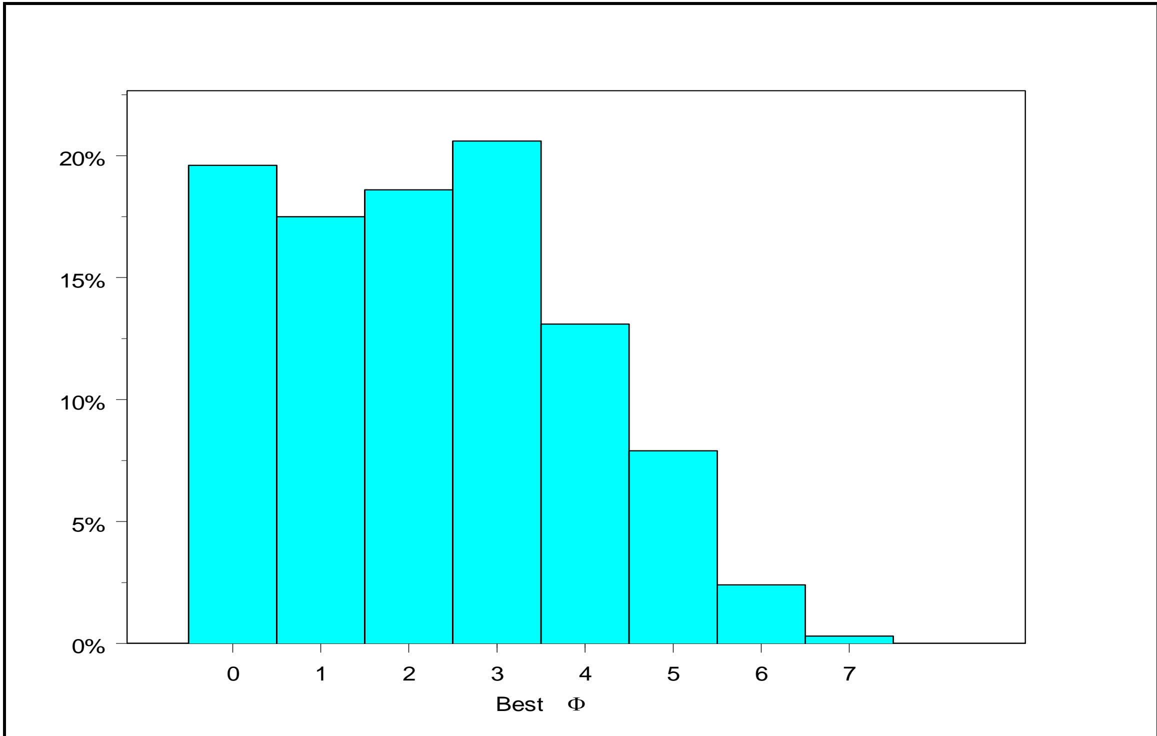

Figure 17: Histogram of "best" Φ parameter values for the 10 job instance dataset, Initial Assign algorithm

As with the Δ parameter, neither the mean nor the median of this distribution will be selected as the Φ parameter. In this case a larger penalty in heuristic performance is paid for an overestimate in the Φ parameter, as seen from the response function curves displayed in Figure 15 and Figure 16.

Box plots for the 10 job instance dataset using the Initial Assign algorithm are displayed in Figure 18. The Φ parameter value that produced the smallest 90th-percentile makespan ratio across the 1,000 job instances was chosen as the Φ parameter value to be used in the heuristic comparisons in this thesis. By choosing this Φ parameter value, the algorithm assignment performance will be at least as good as the 90th-percentile point for 90% of the job instances considered. Table 22

displays the Φ parameter values chosen and associated 90th-percentile makespan ratios.

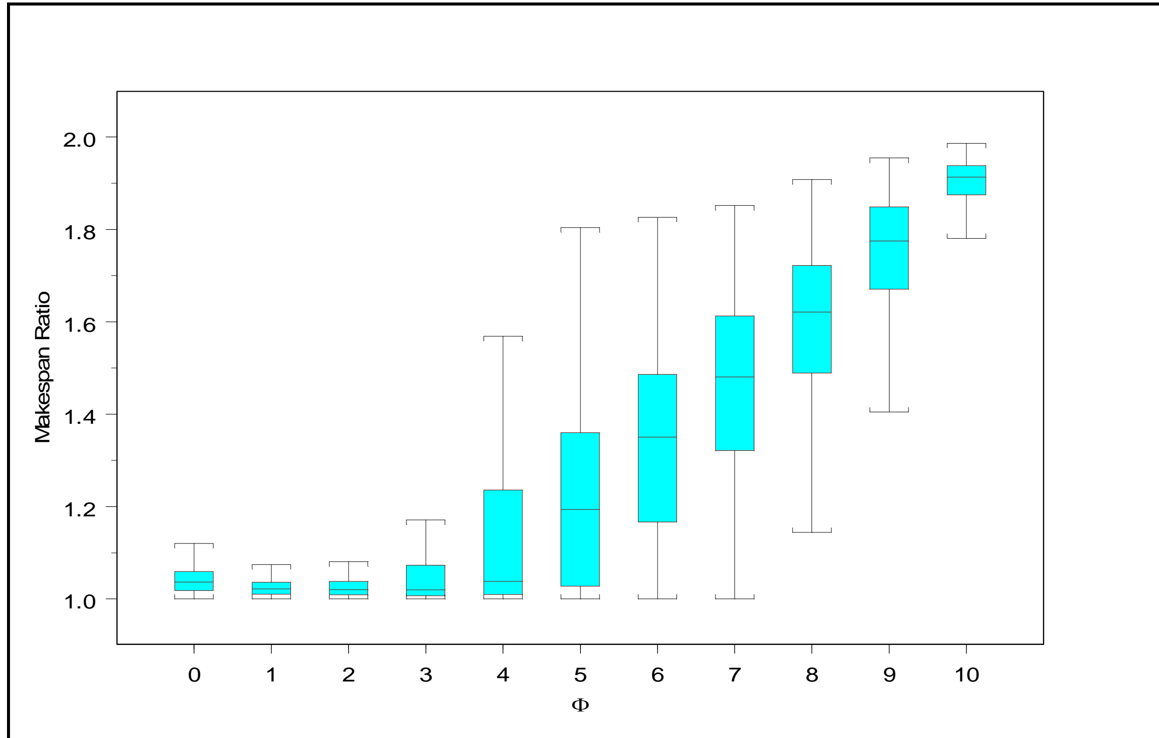

Figure 18: Boxplots of makespan ratios for the 10 job instance dataset, Initial Assign algorithm

Table 22: Selected Φ parameter values

Number of Jobs	Heuristic	Φ (jobs)	Makespan Ratio 90th-percentile
10	Initial Assign	1	1.050
20	Initial Assign	3	1.038

As with the Delta algorithms, a search algorithm that identifies the best Φ parameter value, makespan ratio, and assignment (Best Initial Assign algorithm) is included in the algorithm comparisons made in this thesis as a means to understand the best possible scheduling heuristic performance for the Initial Assign heuristic. Although

we were unable to make use of the shape of the response function and the quasi-convex principle, the distribution of the best Φ parameter did allow us to make some improvement in the search algorithm. For the 10 job instances, it is seen that a minimum makespan ratio is never obtained with a Φ parameter value greater than 7. For the 20 job instances a minimum makespan ratio is never obtained with a Φ parameter value greater than 13. This allows for a reduction in the Φ parameter space to be searched.

5.3 Scheduling Heuristic Performance Results

With the choice of parameter values made, the parameterized algorithms as well as the non-parameterized algorithms were run against both job instance datasets and their scheduling performance results were obtained. A summary of the algorithms considered are displayed in Table 23. Boxplots of the scheduling heuristic performance measure, the makespan ratio, for each algorithm for the 10 and 20 job instance datasets are displayed side-by-side in Figure 19.

Table 23: Algorithm summary

Algorithm	Parameter	
	Number of Jobs	Value
Complete Enumeration (Classical & Matrix Multiplication)	Baseline	
LPT Sum	-	
LPT Max	-	
LPT Min	-	
Delta-LPT Sum	10	4,000
	20	2,700
Best Delta-LPT Sum	-	
Delta-LPT Max	10	4,000
	20	2,700
Best Delta-LPT Max	-	
Delta LPT-Min	10	2,700
	20	3,500
Best Delta LPT-Min	-	
Initial Assign	10	1
	20	3
Best Initial Assign	-	
Ibarra-Kim Algorithm F	-	
Large *k*	-	

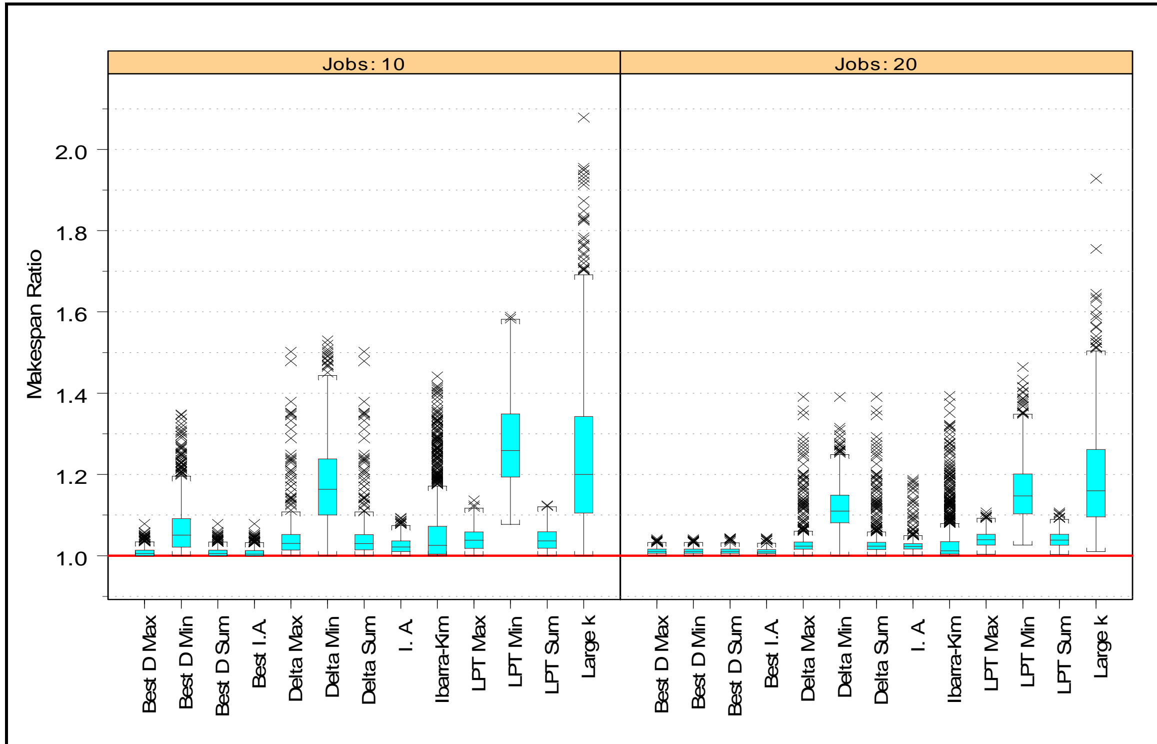

Figure 19: Boxplots of makespan ratios

Recall that the makespan results from all algorithms are compared to the optimal makepan obtained from the complete enumeration algorithm making the optimal and minimum obtainable makespan ratio performance measure 1.0. Several immediate observations, in regard to the algorithm assignment performance can be made from Figure 19:

1. The Delta-LPT Min, LPT Min, and Large *k* algorithms do not perform as well as other algorithms.
2. The LPT Min algorithm does not obtain the optimal makespan ratio for any job instance.

3. All algorithms perform, overall, better when 20 jobs are considered for assignment. In particular, the distributions of the "Best" algorithms' results are much tighter when 20 jobs are assigned.

Several key statistics from the heuristic performance (makespan ratios) distributions are provided in Table 24.

Table 24: Scheduling heuristic performance (makespan ratio) statistics

Algorithm	10 Job Instances Percentiles			20 Job Instances Percentiles		
	0th (min)	50th (median)	90th	0th (min)	50th (median)	90th
Best Delta-LPT Max	1.000	1.005	1.023	1.000	1.011	1.023
Best Delta LPT-Min	1.000	1.051	1.150	1.000	1.011	1.023
Best Delta-LPT Sum	1.000	1.005	1.022	1.000	1.010	1.022
Best Initial Assign	1.000	1.004	1.021	1.000	1.009	1.022
Delta-LPT Max	1.000	1.030	1.074	1.000	1.024	1.047
Delta LPT-Min	1.000	1.163	1.321	1.000	1.110	1.184
Delta-LPT Sum	1.000	1.030	1.074	1.000	1.023	1.046
Initial Assign	1.000	1.022	1.050	1.001	1.023	1.038
Ibarra-Kim	1.000	1.025	1.195	1.000	1.012	1.099
LPT Max	1.000	1.038	1.076	1.003	1.039	1.065
LPT Min	1.077	1.259	1.443	1.026	1.147	1.260
LPT Sum	1.000	1.037	1.077	1.003	1.038	1.064
Large *k*	1.000	1.200	1.487	1.010	1.159	1.374

From Table 24, additional observations regarding the algorithm assignment performance can be made:

1. In addition to the LPT Min not obtaining the optimal makespan ratio for any job instance, when 20 jobs are considered for assignment the Initial Assign, LPT Max, LPT Sum, and Large *k* algorithms also do not obtain the optimal makespan ratio (minimum makespan ratio statistic is greater than 1.0).
2. The "Best" algorithms perform almost identically when assigning 20 jobs with a median makespan only 1% greater than the optimal and 90% of the

jobs instances assigned resulted in makespans no worse than 2% greater than the optimal. When 10 jobs are assigned, the Best Delta-LPT Min algorithm does not perform as well as the other "Best" algorithms. The Best Delta-LPT Max, Best Delta-LPT Sum, and Best Initial Assign algorithms all produce a median makespan within 0.5% of the optimal.

3. As expected, there is more variation between the performances of the parameterized algorithms with a set parameter value than between the "Best" algorithms. The Delta-LPT Min algorithm does not perform as well as the other parameterized algorithms for either job instance size. The remaining parameterized algorithms perform equally well when 20 jobs are assigned, with median and 90th percentile makespans approximately 2% and 4% greater than the optimal, respectively. When 10 jobs are assigned, the Initial Assign parameterized algorithm performs slightly better than the Delta-LPT Max and Delta-LPT Sum algorithms. The median makespan for the Initial Assign algorithm when assigning 10 jobs is 2% greater than the optimal with a 90th percentile makespan 5% greater than the optimal. The median and 90th percentile makespans for the Delta-LPT Max and Delta-LPT Sum algorithms are 3% and 7% greater than the optimal, respectively.

4. Among the remaining algorithms, even greater variability is observed. Of these, the LPT Max and LPT Sum algorithms perform the best with 90% of the assignments having a makespan ratio within 7% of the optimal. The Ibarra-Kim algorithm produces a 90th percentile makespan within 10% of the optimal when assigning 20 jobs, but to only within 20% of the optimal when

assigning 10 jobs. The LPT Min and the Large *k* algorithms bring up the tail in terms of performance.

Final comparisons of the assignment performance results of the algorithms are made in Figure 20 and Figure 21, the empirical cumulative distribution functions. Three general groups are observed for the 10 job instance case, Figure 20:

1. Wide distribution with a long tail – algorithms included in this group are: Delta-LPT Min, LPT Min, and Large *k*.
2. Tighter distribution but yet still a long tail – algorithms included in this group are: Delta-LPT Max, Delta-LPT Sum, Best Delta-LPT Min, and Ibarra-Kim Algorithm F.
3. Tight distribution with relatively short tail – algorithms included in this group are: Best Delta-LPT Max, Best Delta-LPT Sum, Best Initial Assign, Initial Assign, LPT Sum, and LPT Max.

These groupings remain in the 20 job instances, Figure 21, with slight variations however, the bodies of the distributions are observed to be much closer together.

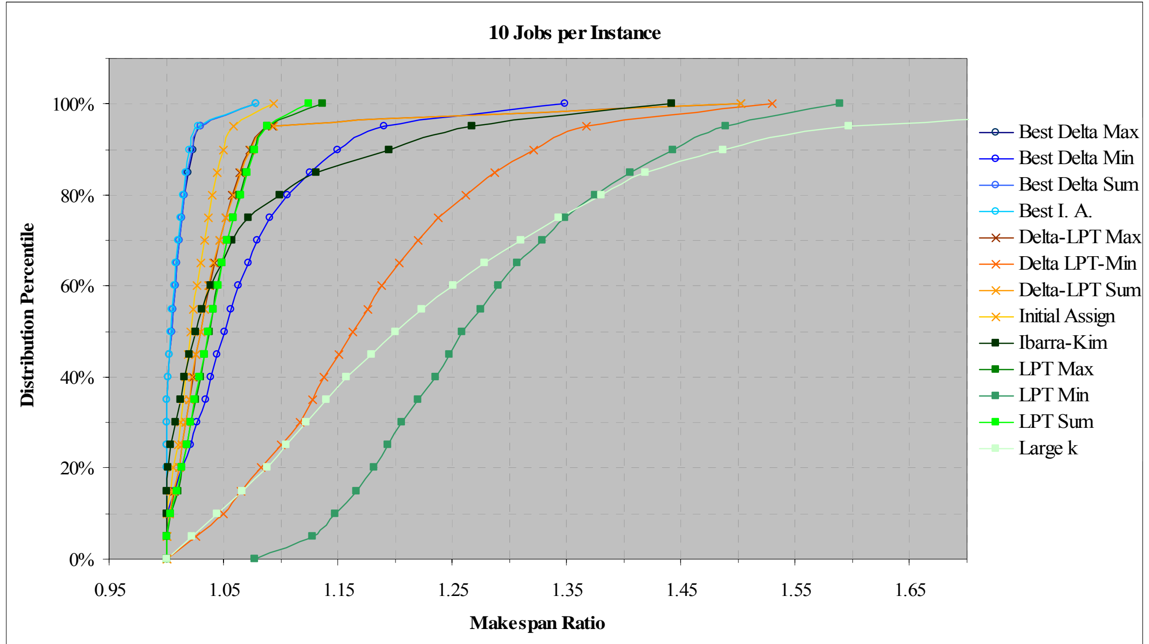

Figure 20: Empirical cumulative distribution functions, 10 jobs per instance

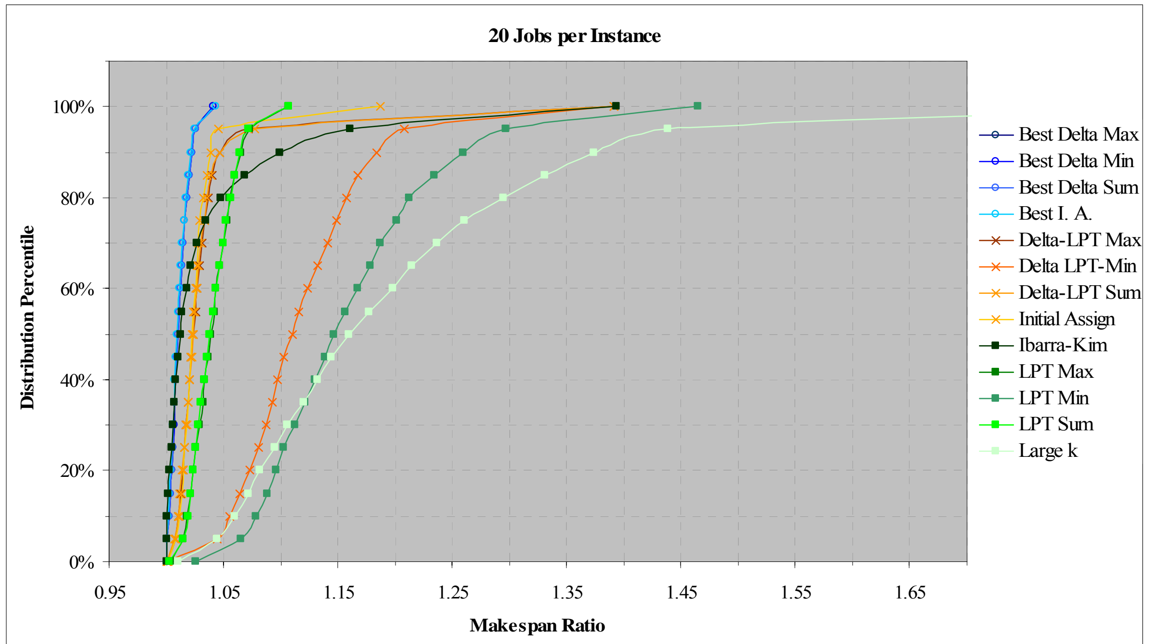

Figure 21: Empirical cumulative distribution functions, 20 jobs per instance

The following are some final observations on the scheduling heuristic performance made from the empirical cumulative distribution functions in Figure 20 and Figure 21:

1. The Best Delta-LPT Max, Best Delta-LPT Sum, and Best Initial Assign algorithms performed the best and are the most robust across the number of jobs being assigned.
2. The Large *k* algorithm is also robust across the number of jobs being assigned, however its resulting makespan ratios are less desirable than most other algorithms considered.
3. The parameterized algorithm Initial Assign performs very well with over 90% of the job instances assigned resulting in makespans that fall within 5% of the optimal makespan.
4. When assigning 20 jobs, the Ibarra-Kim algorithm F is seen to perform very well for 80% of the job instances considered (makespan within 5% of optimal), however, the performance for the remaining 20% of job instances quickly deteriorates.

5.4 Algorithm Performance Results

There is a cost associated with the production of each assignment schedule for each of the algorithms considered. It is hypothesized that the more complicated and burdensome the assignment algorithm, the more costly the algorithm. The cost considered in this thesis, as discussed in Section 3.5.3, is the CPU processing time, in

seconds, required to complete the assignment for each job instance. The CPU time is utilized as our measure of algorithm performance.

The required CPU processing time is captured using the S-Plus function ‘sys.time’. The function returns the CPU and elapsed times (in seconds) required to evaluate the associated expression, in our case, the assignment algorithms. A brief exploration of the S-Plus ‘sys.time’ function was performed to gain an understanding of the function’s abilities. Using the ‘sys.time’ function, the CPU time required to perform the summation $\sum_{i=1}^{1,000,000} i$ by means of looping was obtained. This calculation was performed 100 times, each time the required CPU processing time was recorded. The CPU time results (Figure 22) display that there is some amount of variation in the CPU processing time reported by the ‘sys.time’ function when the exact same calculation is performed repeatedly. It is suspected that this variation is attributable to the CPU performing additional underlying processing tasks as well as the way in which the ‘sys.time’ function calculates the CPU processing time. Due to this variability in CPU time output, the performance of the algorithms will be evaluated as distributions of CPU times as opposed to a single CPU processing time measurement.

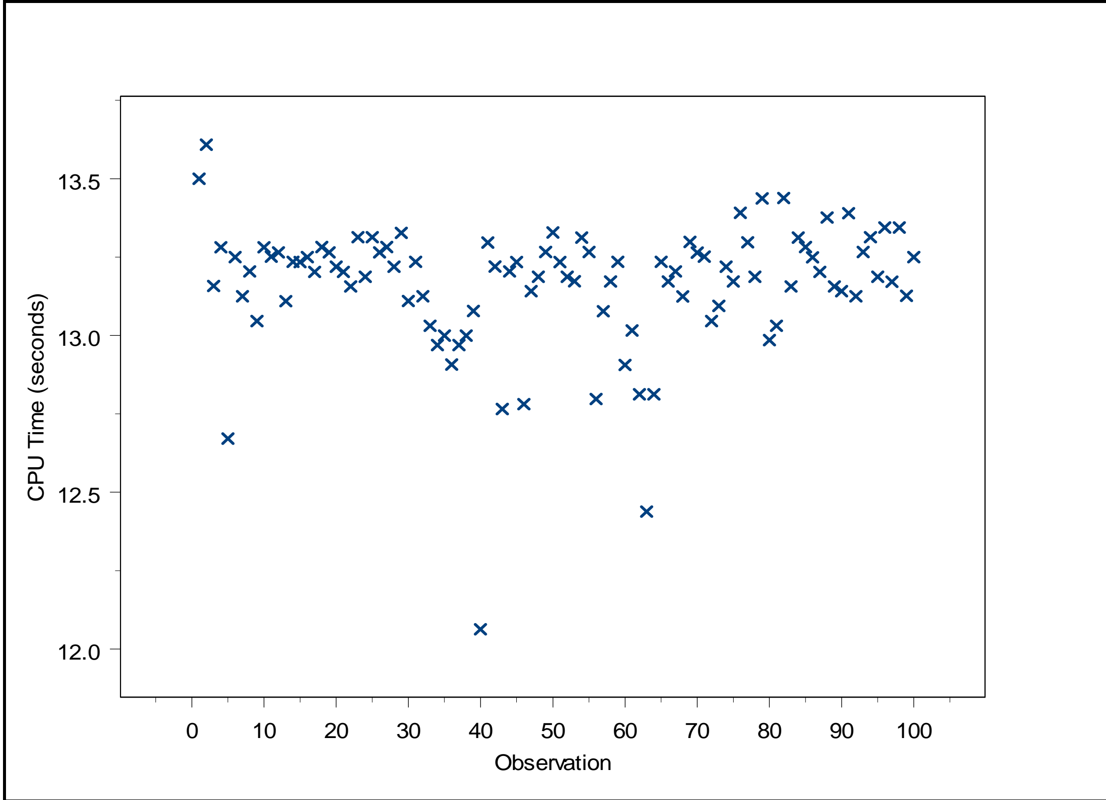

Figure 22: CPU processing time for $\sum_{i=1}^{1,000,000} i$ summation

5.4.1 Algorithm Results

In parallel to the scheduling performance measure described in Section 5.3, the algorithm performance measure of CPU time was captured for each assignment schedule produced by each assignment algorithm. Though the makespan ratio scheduling performance measure was not of interest for the Complete Enumeration algorithms as the resulting makespans were used as the baseline, the required CPU processing time for these algorithms is certainly of interest. Of particular interest is how the complete enumeration approach considering the statistical concept of the two-level, full factorial experimental design and matrix multiplication to obtain the optimal assignment schedule compares to the more classical approach entailing a one

at a time evaluation and comparison. Boxplots of the algorithm performance measure, the CPU time, for each algorithm for the 10 and 20 job instance datasets are displayed side-by-side in Figure 23.

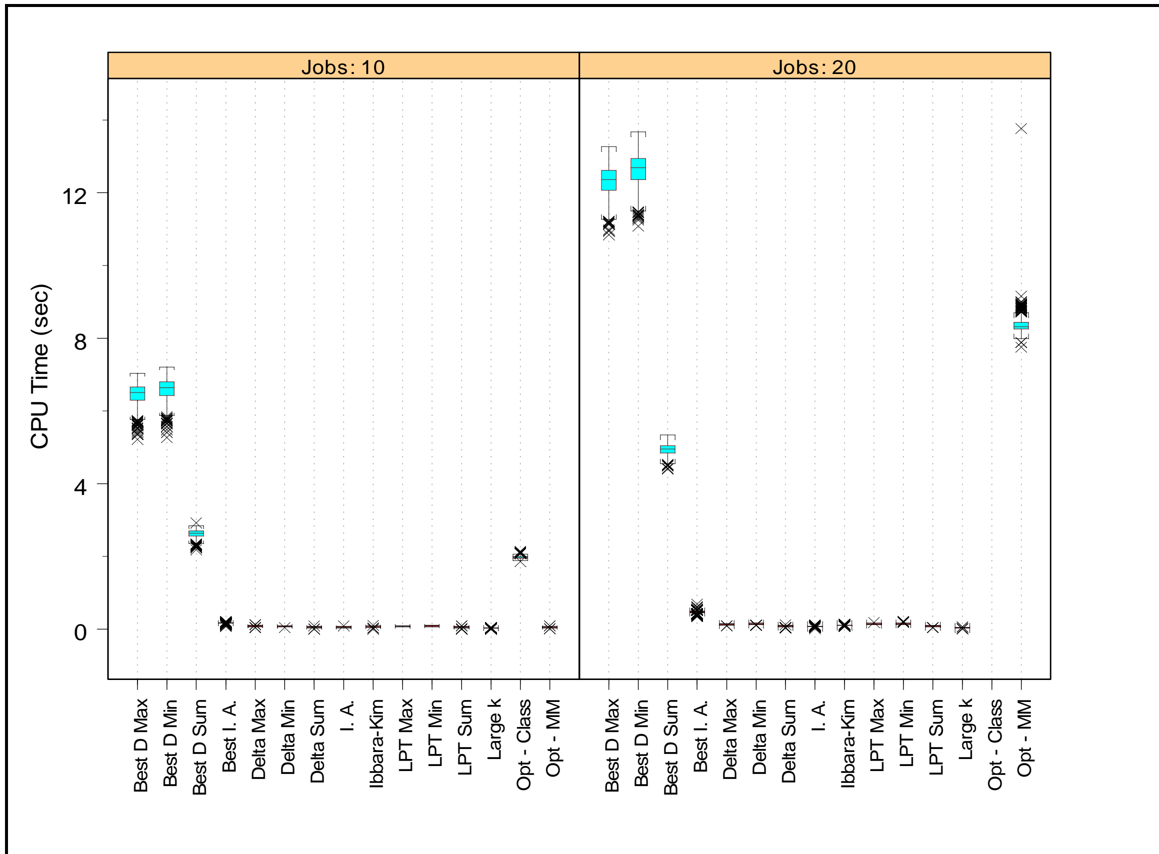

Figure 23: Boxplots of CPU times

Several immediate observations can be made from Figure 23:

1. The CPU times associated with the Best Delta-LPT Max, Best Delta-LPT Min, and Best Delta-LPT Sum algorithms are significantly higher than those associated with the other algorithms for both job instance sizes as the “best” algorithms must search across many possible parameter values.

2. Job instance size has a large impact on the CPU time for the Complete Enumeration via Matrix Multiplication (Opt-MM) algorithm. It is seen that the CPU time associated with the Opt-MM algorithm is relatively low when assigning 10 jobs, however a drastic increase is seen when considering 20 jobs.
3. Observations for the Complete Enumeration using the classical approach (Opt-Class) algorithm are included only for the 10 job instance. Due to the necessary continuous computer processing time of more than 48 hours, results for the 20 job instance were not obtained.

Several key statistics from the algorithm performance (CPU times) distributions are provided in Table 25.

Table 25: Algorithm performance (CPU time) statistics

Algorithm	10 Job Instances			20 Job Instances		
	Percentiles			Percentiles		
	0th (min)	50th (median)	90th	0th (min)	50th (median)	90th
Best Delta-LPT Max	5.219	6.501	6.782	10.844	12.360	12.844
Best Delta LPT-Min	5.266	6.641	6.908	11.078	12.687	13.141
Best Delta-LPT Sum	2.188	2.640	2.750	4.407	4.953	5.140
Best Initial Assign	0.078	0.172	0.173	0.343	0.484	0.500
Delta-LPT Max	0.047	0.079	0.094	0.094	0.140	0.141
Delta LPT-Min	0.047	0.078	0.094	0.109	0.141	0.157
Delta-LPT Sum	0.000	0.062	0.063	0.031	0.078	0.094
Initial Assign	0.030	0.062	0.063	0.016	0.078	0.094
Ibarra-Kim	0.000	0.063	0.078	0.078	0.109	0.110
LPT Max	0.062	0.093	0.094	0.124	0.141	0.157
LPT Min	0.062	0.094	0.094	0.124	0.156	0.157
LPT Sum	0.000	0.062	0.063	0.047	0.079	0.094
Large *k*	0.000	0.031	0.032	0.015	0.047	0.047
Optimal – Classical	1.860	1.969	2.031	NA	NA	NA
Optimal – Matrix Mult	0.016	0.047	0.063	7.750	8.312	8.578

From Table 25 additional observations regarding the algorithm performance can be made:

1. Though subtle in some algorithms, job instance size has an effect on the CPU time for all algorithms as seen by the increase in the CPU time statistics from 10 jobs instances to 20 job instances.
2. Of the "Best" algorithms, which are amongst the most costly algorithms, the Best Delta-LPT Max and Best Delta-LPT Min algorithms require the most CPU processing time. The Best Initial Assign algorithm is somewhat comparable in terms of CPU time to most other algorithms, particularly when considering only 10 jobs.
3. The parameterized heuristics (Delta-LPT Max, Delta-LPT Min, Delta-LPT Sum, and Initial Assign) are in line with the remaining heuristics. The Delta-LPT Sum and Initial Assign heuristics are slightly less costly and less affected by job instance size than the Delta-LPT Max and Delta-LPT Min heuristics.
4. As expected, the Large k heuristic, with its low degree of complexity, requires the least amount of CPU processing time.
5. When considering the 10 job instances, the Optimal – Matrix Multiplication algorithm greatly out performs its Optimal – Classical counterpart in addition to most other heuristics. The impact of the increased job instance size is dramatic for the Optimal – Matrix Multiplication algorithm, with its cost being among the highest when 20 jobs are considered for assignment.

Final comparisons of the algorithm performance are made in Figure 24 and Figure 25, the empirical cumulative distribution functions. Three groups are observed for the 10 job instance case, Figure 24:

1. High CPU Time – algorithms included in this group are: Best Delta-LPT Max and Best Delta-LPT Min.
2. Moderate CPU Time – algorithms included in this group are: Best Delta-LPT Sum, Complete Enumeration – Classical approach.
3. All other algorithms appear with small CPU times.

This pattern of several algorithms requiring larger amounts of CPU time with the others requiring small amounts of CPU time is also displayed when considering 20 jobs for assignment. Figure 26 and Figure 27 have an adjusted CPU Time axis to allow for greater resolution when considering the algorithms requiring small amounts of CPU time.

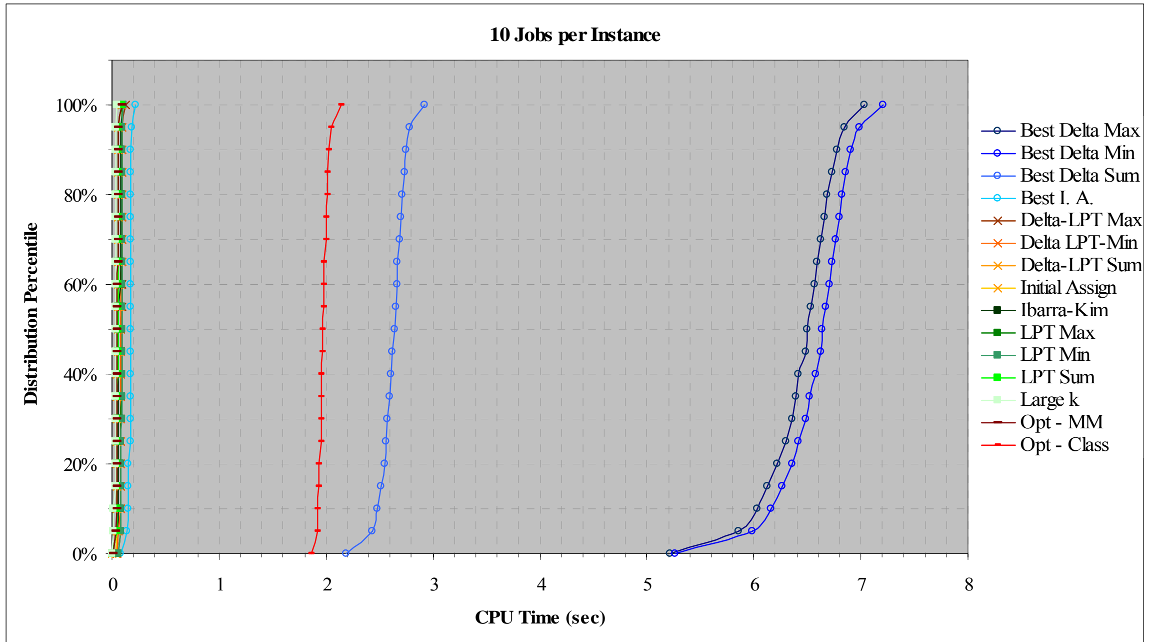

Figure 24: Empirical cumulative distribution functions, 10 jobs per instance

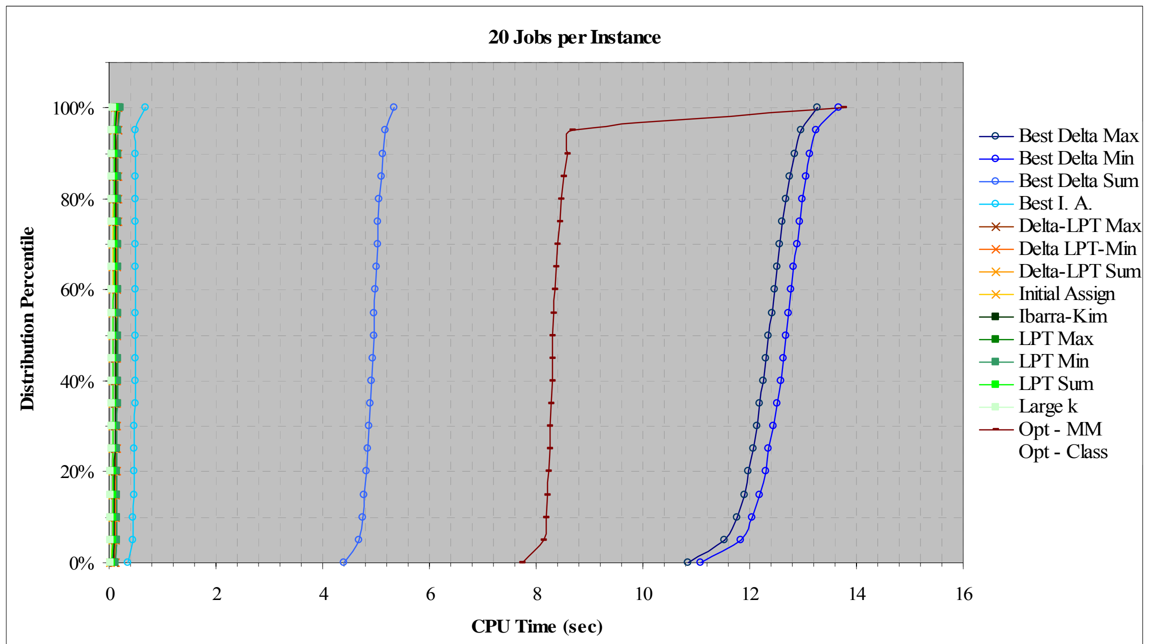

Figure 25: Empirical cumulative distribution functions, 20 jobs per instance

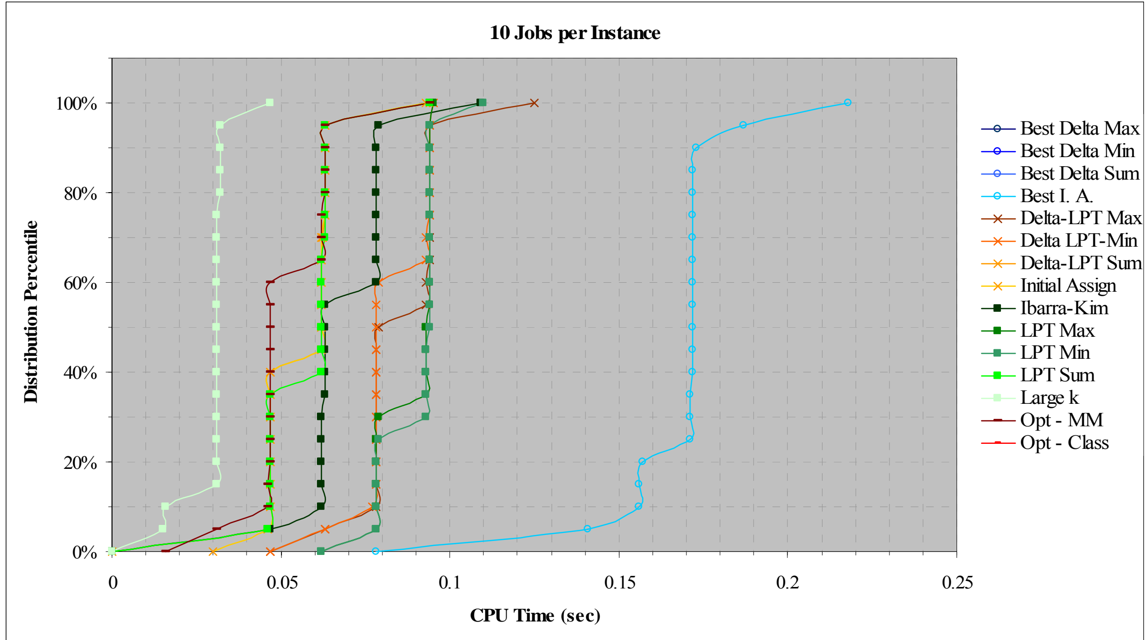

Figure 26: Empirical cumulative distribution functions, 10 jobs per instance, adjusted CPU Time axis

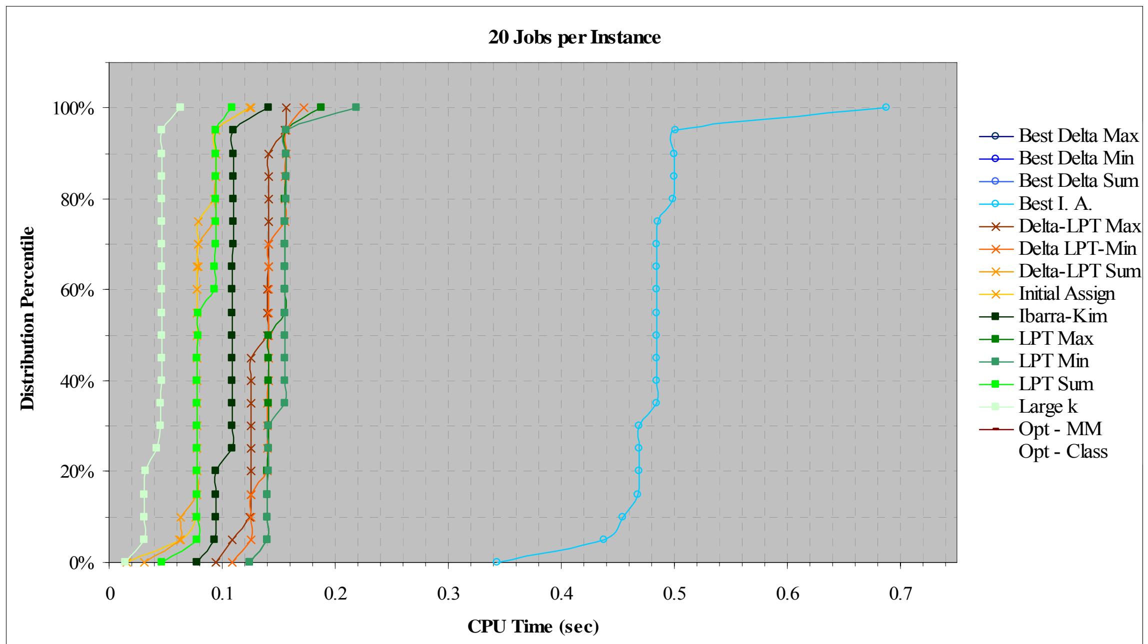

Figure 27: Empirical cumulative distribution functions, 20 jobs per instance, adjusted CPU Time axis

Following are some final observations on the algorithm performance made from the empirical cumulative distribution functions in Figure 24 through Figure 27:

1. The Large *k* algorithm requires the least amount of CPU time, regardless of the number of jobs to be assigned.
2. The Best Delta-LPT Max and Best Delta-LPT Min algorithms require the most amount of amount of CPU time, regardless of the number of jobs to be assigned.
3. When assigning only 10 jobs, the Optimal – Matrix Multiplication algorithm is only second to the Large *k* algorithm in terms of CPU time required.
4. The Optimal – Classical approach algorithm, where data has only been collected for the 10 job instance, requires a significant amount of processing time with the fourth largest CPU time.

5.4.2 CPU Time as a Function of Number of Jobs

As it was observed in Section 5.4.1, the number of jobs considered for assignment has an impact on the CPU time required for an algorithm to make its assignment. This impact ranged greatly. Three algorithms have been selected to further investigate this impact of job instance size on CPU time. The algorithms selected are: the Large *k* algorithm which had a seemingly small change in CPU time due to job instance size; the Optimal – Matrix Multiplication algorithm with a large change in CPU time; and the Best Initial Assign algorithm with a moderate change in CPU time.

Twenty datasets, each containing 1,000 sample instances, were created for job instance sizes of 1 through 20. Each of the three algorithms was run and the average

CPU time required to perform each of the 1,000 assignments for each of the 20 job instance sizes was recorded. The average CPU time as a function of number of jobs considered for assignment for the three algorithms is displayed in Figure 28.

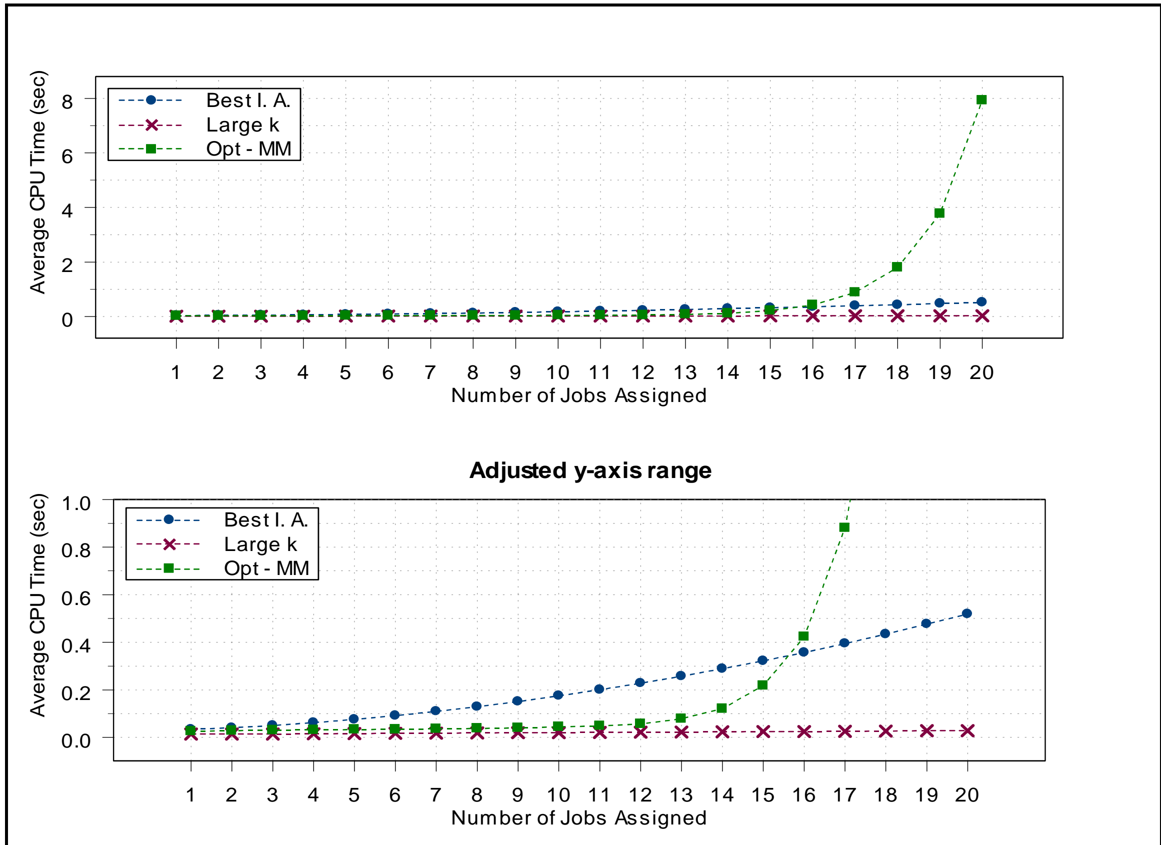

Figure 28: CPU time as a function of number of jobs assigned

From the upper graph in Figure 28 it is seen that the three algorithms require approximately the same amount of CPU processing time until 15 jobs are considered for assignment when the Optimal – Matrix Multiplication algorithm begins to display an exponential response for CPU time. By adjusting the y-axis range in the lower graph in Figure 28, a closer look at the algorithm comparisons can be had. This closer look displays that until more than 15 jobs are considered for assignment, the

Optimal – Matrix Multiplication algorithm actually requires less CPU processing time than the Best Initial Assign algorithm that does not guarantee an optimal solution!

5.5 Cost – Benefit Tradeoff

Given a group of jobs the ideal manufacturing line assignment is the assignment that achieves the minimum possible makespan. The minimum makespan is achieved with the complete enumeration algorithms, and it has been shown to often be achieved with several of the heuristics presented in this thesis. The cost associated with achieving this optimal, or near optimal makespan, however, may be more than what is practical in many rapid paced production settings.

With the cost of an assignment algorithm measured in CPU processing time and the benefit received from the resulting assignment schedule measured by the makesan ratio, the most desirable algorithm would minimize both measures. That is, a CPU time as close to zero as possible with a makespan ratio as close to one as possible. Selecting the median CPU time and median makespan ratio as the representative cost and benefit values for the considered algorithms, Figure 29 and Figure 30 present a graphical display of the cost-benefit tradeoff while Table 26 presents the associated values.

.

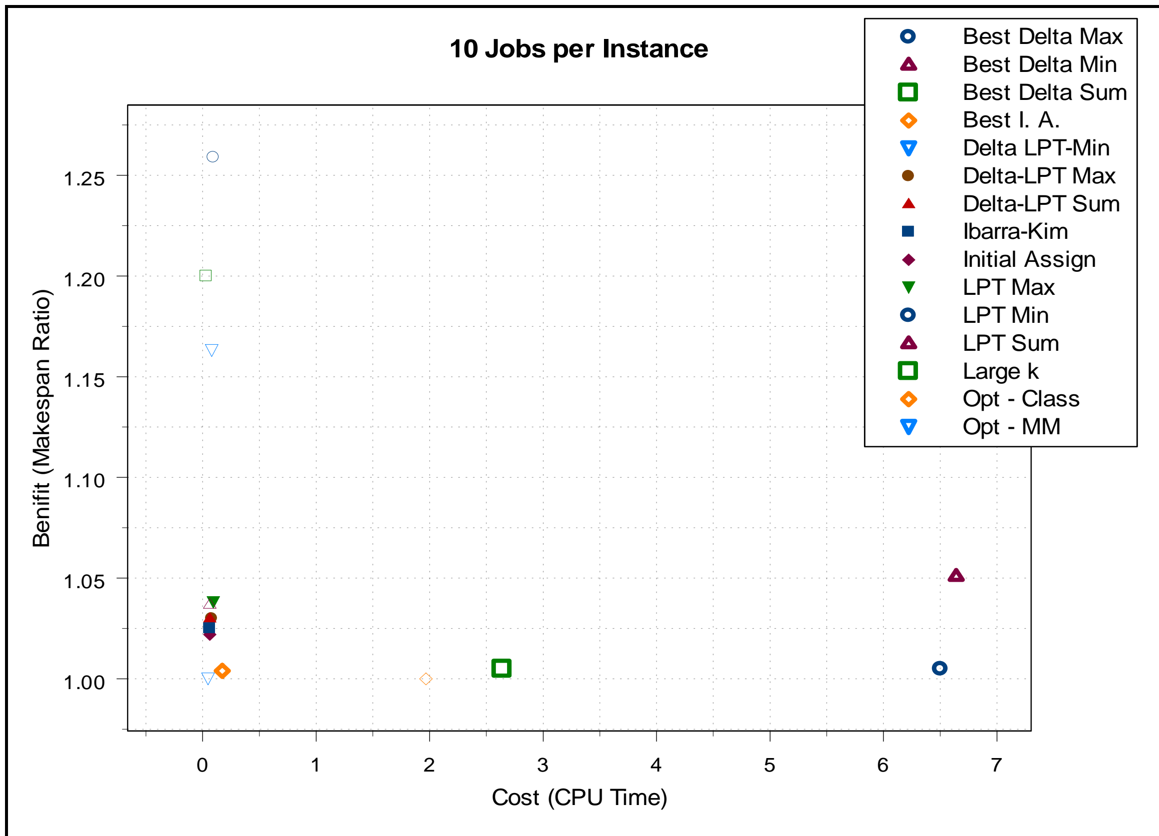

Figure 29: Cost-benefit tradeoff, 10 jobs per instance

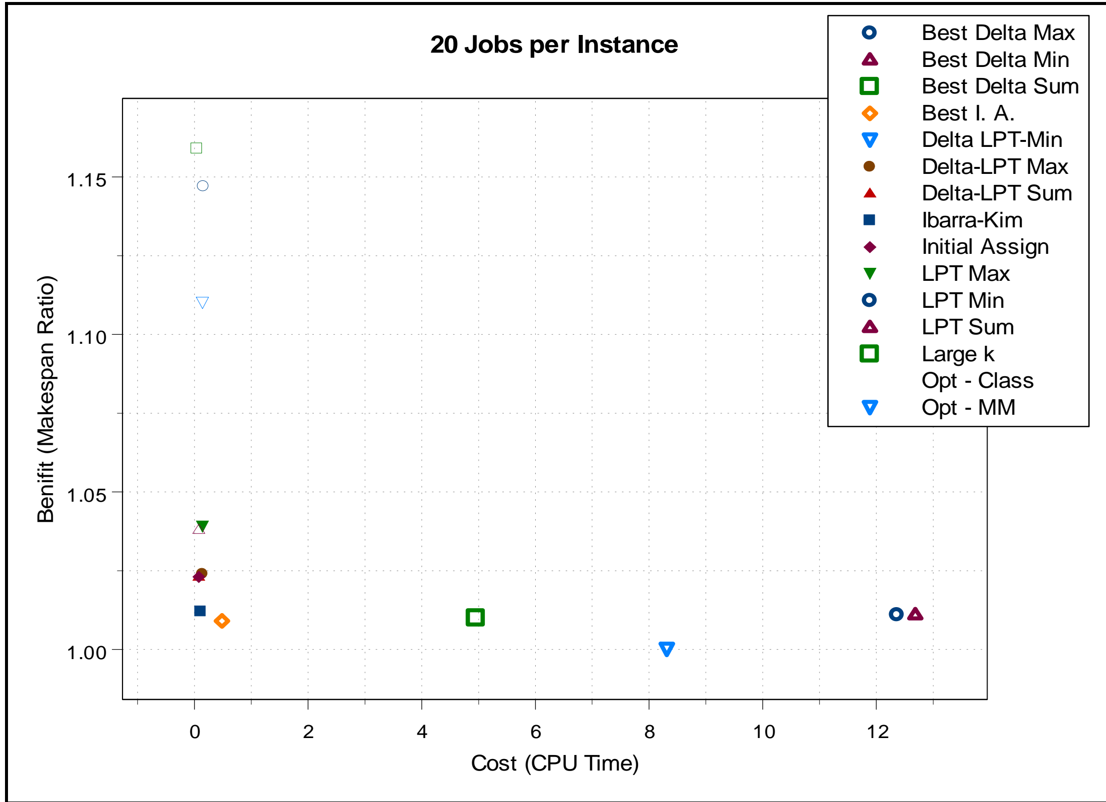

Figure 30: Cost-benefit tradeoff, 20 jobs per instance

Table 26: Cost-benefit values

Algorithm	10 Jobs per Instance		20 Jobs per Instance	
	Cost (Median CPU Time)	Benefit (Median Makespan Ratio)	Cost (Median CPU Time)	Benefit (Median Makespan Ratio)
Best Delta-LPT Max	6.501	1.005	12.360	1.011
Best Delta LPT-Min	6.641	1.051	12.687	1.011
Best Delta-LPT Sum	2.640	1.005	4.953	1.010
Best Initial Assign	0.172	1.004	0.484	1.009
Delta-LPT Max	0.079	1.030	0.140	1.024
Delta LPT-Min	0.078	1.163	0.141	1.110
Delta-LPT Sum	0.062	1.030	0.078	1.023
Initial Assign	0.062	1.022	0.078	1.023
Ibarra-Kim	0.063	1.025	0.109	1.012
LPT Max	0.093	1.038	0.141	1.039
LPT Min	0.094	1.259	0.156	1.147
LPT Sum	0.062	1.037	0.079	1.038
Large *k*	0.031	1.200	0.047	1.159
Optimal – Classical	1.969	1.000	NA	1.000
Optimal – Matrix Mult	0.047	1.000	8.312	1.000

A decision alternative A is said to be dominated by another alternative B if all performance measures associated with alternative B are more desirable than those associated with alternative A [15]. The choice strategy of dominance is one useful and procedurally rational [16] approach to narrowing the alternative space and making a choice in a decision problem.

In the 10 jobs per instance case, all algorithms are seen to be dominated by either the Large *k* or the Optimal – Matrix Multiplication algorithms. The resulting subset of dominating alternatives from which to choose for the use in assigning job instances of size 10 is the Large *k* heuristic with a 1.2 makespan ratio and 0.031 CPU time and the Complete Enumeration via Matrix Multiplication algorithm with a makespan ratio and CPU time of 1.0 and 0.047 respectively.

Unfortunately, the algorithm alternative space is not as drastically simplified when 20 jobs are considered for assignment. Six algorithms are identified as dominating alternatives. These alternatives and associated cost-benefit characteristics are listed in Table 27. A graphical display of the cost-benefit tradeoff curve for these alternatives is presented in Figure 31.

Table 27: Dominating algorithm alternatives, 20 job instance

Algorithm	Cost (Median CPU Time)	Benefit (Median Makespan Ratio)
Best Initial Assign	0.484	1.009
Delta-LPT Sum	0.078	1.023
Initial Assign	0.078	1.023
Ibarra-Kim	0.109	1.012
Large *k*	0.047	1.159
Optimal – Matrix Mult	8.312	1.000

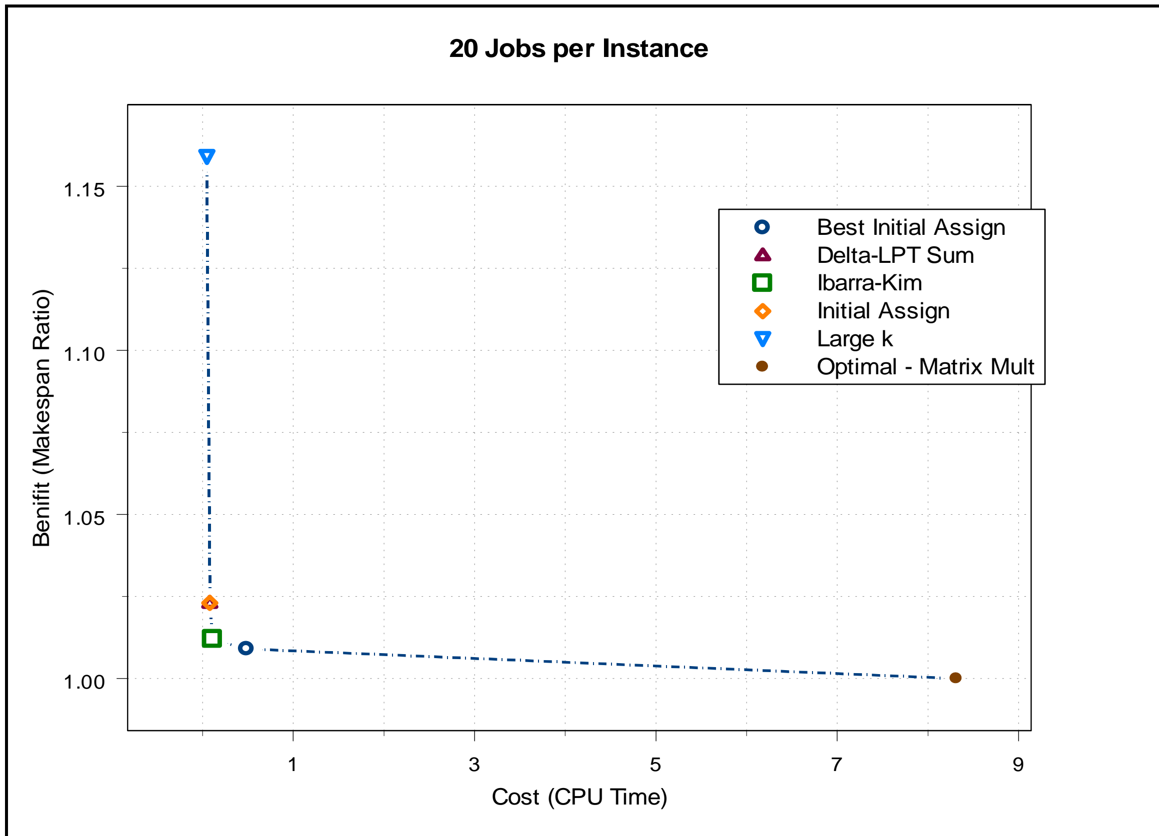

Figure 31: Cost-benefit tradeoff curve, dominating algorithms, 20 jobs per instance

5.6 Chapter Summary

This chapter presented the results from the simulation work performed for this thesis.

The selection of the parameter values for the parameterized Delta and Initial Assign

heuristics was discussed. Results and observations of the scheduling heuristic performance measure of the makespan ratio were presented. The performance of the algorithms, using the measure of required CPU processing time, was also discussed. A deeper look at how the number of jobs affected the performance of several algorithms was made.

With the cost of the algorithm in terms of CPU time and the benefit provided by the algorithm as the makespan ratio in hand, a cost-benefit tradeoff was presented. Of the 15 algorithms considered, two algorithms, the Large k and Optimal – Matrix Multiplication algorithms were found to be dominate in the tradeoff analysis of the 10 job instance. When 20 jobs were considered for assignment, six algorithms: the Best Initial Assign, Delta-LPT Sum, Initial Assign, Ibarra-Kim, Large *k*, and the Optimal – Matrix Multiplication algorithms; were identified as the dominating algorithms in the tradeoff analysis.

Chapter 6: Summary and Conclusions

6.1 Work Performed

This thesis presented the development and evaluation of several algorithms for scheduling two unrelated parallel processors with the objective function of minimizing the makespan. The fabricating process of a local manufacturer who produces printed circuit boards was considered as the foundational basis and motivation for this work. The manufacturing setting consists of two non identical parallel manufacturing lines. Job build data for over 425 actual jobs were supplied by the manufacturer. This data consists of job characteristics such as the number of boards to be built, the number and types of components to be placed on the board, and the necessary component distribution systems.

A deterministic analytical model describing the job manufacturing process time for each of the two manufacturing lines was defined. The model included the set up of the manufacturing lines and the complete production of all boards within a given job. Both manufacturing lines perform basically the same manufacturing functions consisting of the following automated stations connected by automatic conveyer transfers: Board Loader, Stencil Print, Loctite Adhesive Dispense, Component Auto Placement machines, Automated Optical Inspection, Hand Placement, Oven, and a final Board Loader. The major difference in the two lines is that Line 2 has an additional auto placement machine dedicated to placing the circuit board components that are common to most jobs manufactured. Though this dedicated machine

increases the number of necessary build steps on Line 2, the required set up time for many jobs is drastically reduced.

A typical production week for the circuit board manufacturer consists of the production of 10 to 20 jobs. Two datasets of problem instances used in the simulations presented in this thesis were created from the job data provided by the manufacturer. The first dataset consists of 1,000 samples, each consisting of 10 jobs randomly selected with replacement, from the supplied jobs. The second dataset also contains 1,000 samples from the jobs supplied; however each sample in this dataset consists of 20 jobs randomly selected with replacement.

Presented with the problem of assigning a group of jobs to the two manufacturing lines in such a way as to minimize the time required to complete the entire group of jobs, several assignment algorithms were considered. Two approaches, a classical approach and a matrix multiplication approach, were taken to obtain the optimal assignment solution through complete enumeration. Two heuristics, the Ibarra-Kim Algorithm F and the Longest Processing Time first (LPT) rule, from the published scheduling literature were considered. An expansion of the standard LPT rule was necessary to allow the rule to be suitable for the unrelated machine problem. This expansion resulted in three variants: the LPT Sum, LPT Max, and LPT Min heuristics. Three heuristics were newly developed: the Delta, Initial Assign, and Large *k* heuristics.

The Delta heuristic also has three variants: the Delta-LPT Sum, Delta-LPT Max, and Delta-LPT Min. The Delta heuristic is a two step heuristic. The first step considers an elementary comparison of the time required to process each job on each line. If that absolute difference in processing time is greater than some value, Δ, then the job is assigned to the line that processes it quicker. The second step then employs the more computationally intensive LPT variants to assign the reduced set of unassigned jobs.

The Initial Assign heuristic was developed as an improvement to the Delta heuristics. Recognizing that the number of jobs initially assigned in the first step of the Delta heuristic could only be a discrete value ranging from zero to the total number of jobs being considered, the parameter that controls the initial assignment was simplified from the continuous and boundless parameter Δ to a bounded discrete parameter Φ. The Initial Assign heuristic is also a two step heuristic. The first step again considers the elementary comparison of the time required to process a job on each line. The Φ number of jobs with the largest absolute difference in processing times is assigned to the line that processes it quicker. The second step uses the LPT Sum heuristic to assign the reduced set of unassigned jobs as it was observed that the LPT Sum heuristic performed better than its Max and Min counterparts on a consistent basis.

The Ibarra-Kim Algorithm F, LPT algorithms, Delta-LPT algorithms, and the Initial Assign algorithm all utilize the processing time of each job on each line. To obtain this information, all of the individual job characteristics must be accessed and the

total line processing time model must be relied upon and run. In contrast, the Large k heuristic requires only one job attribute to perform the assignment, the number of boards within a job, k. The Large k algorithm assigns the job with the largest number of boards to a pre-determined manufacturing line, in our case Line 2. The remaining jobs are then assigned in a *LPT-type* manner. The remaining unassigned job with the largest k is assigned to the manufacturing line with the smallest number of boards assigned to it thus far.

Each assignment algorithm was used to assign each of the 1,000 groups of jobs in the two datasets of problem instances. The resulting makespan from each of the heuristic algorithms was compared to the optimal makespan obtained by the complete enumeration algorithm for each group of jobs. This comparison was defined as the makespan ratio and was used to quantify the performance of the scheduling heuristic. In addition to the scheduling performance, the algorithm performance – or cost at which the assignment schedule was produced was of interest. This was quantified by measure of CPU processing time.

By investigating the scheduling performance measure, values for the parameters Δ and Φ in the Delta and Initial Assign heuristics, respectively, were chosen. The resulting performance measures from all of the algorithms were compared and a cost-benefit tradeoff was considered. Results from the comparisons are summarized below in Section 6.2.

6.2 Results and Conclusions

Prior to running the algorithms, the parameter values for the parameterized Delta and Initial Assign heuristics were investigated and chosen. The ideal parameter value would be such that the resulting makespan ratio is the minimum over all possible parameter values. It was seen that a choice in the Δ parameter that underestimated the ideal choice in parameter value would have a much larger negative impact on the resulting makespan ratio than a choice in parameter that overestimated the ideal value. Due to the nature of the Initial Assign algorithm, the reverse relation was seen to hold true. A choice in the Φ parameter that overestimated the ideal choice had a larger negative impact on the resulting makespan ratio than a choice in parameter that underestimated the ideal value. The choices for the various Δ and Φ parameters were discussed in Section 5.2.

The heuristic scheduling performance measures of the makespan ratios obtained by the assignment for each of the 1,000 groups of jobs in the two datasets of problem instances were compared. Though the performance and the ranking of the algorithms differed across the 10 and 20 job instance datasets, it was observed that the Best Delta-LPT Max, Best Delta-LPT Sum, and Best Initial Assign algorithms produced the assignments resulting in the most desirable makespan ratios and were the most robust across the number of jobs being assigned. The Large *k* algorithm was also observed to be robust across the number of jobs being assigned, however the resulting makespan ratios were less desirable than most other algorithms considered. Complete

heuristic performance results, including empirical cumulative distribution functions for the makespan ratio of each algorithm were presented in Section 5.3.

The CPU processing time required for each of the algorithms to make an assignment was considered as the assignment cost, or algorithm performance measure. As expected, the Best Delta algorithms required a significant amount of CPU processing time to search the Δ parameter space to identify the best choice in Δ. Somewhat surprising was that when assigning 10 jobs, the Complete Enumeration via Matrix Multiplication algorithm which guarantees an assignment with the minimum makespan required less CPU processing time than all other algorithms except the Large *k* algorithm. However, when considering 20 jobs for assignment, the Complete Enumeration via Matrix Multiplication algorithm required much more processing time. The best performing algorithm in regards to the smallest amount of CPU processing time required was the Large *k* algorithm that only considers one job characteristic, the number of boards, for assignment.

It was observed that the number of jobs to be assigned had an impact on the CPU processing time required by the algorithms. This impact varied significantly across the algorithms. Three algorithms seen to be affected minimally (Large *k*), moderately (Best Initial Assign), and dramatically (Complete Enumeration via Matrix Multiplication) were further investigated. It was seen that the shape of the response curve for CPU time as a function of the number of jobs to be assigned varied across the three algorithms. In fact, it was seen (Figure 32) that the Complete Enumeration

via Matrix Multiplication algorithm required little CPU time until more than 15 jobs were considered for assignment where the response began to grow exponentially.

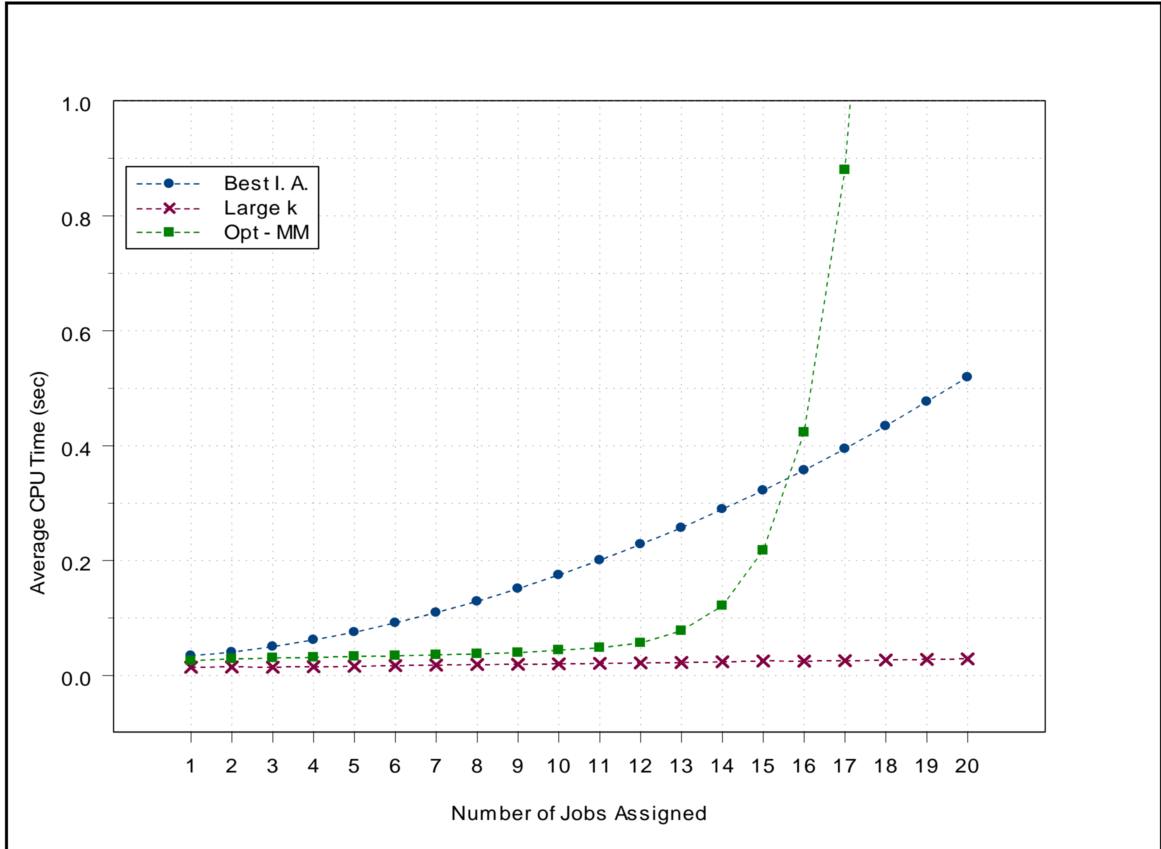

Figure 32: CPU time as a function of number of jobs assigned

With the desire to minimize the cost of an assignment algorithm measured in CPU time and maximize the benefit received from the resulting assignment schedule with a minimal makesan ratio, a cost-benefit tradeoff analysis was considered. Using the choice strategy of dominance, the number of alternative algorithms was reduced from 15 to 2 for the 10 jobs per instance case, and from 14 to 6 in the 20 jobs per instance case. (Data was not generated for the 20 jobs per instance case for the Classical

Complete Enumeration). The resulting algorithms and tradeoff curves are displayed in Figure 33.

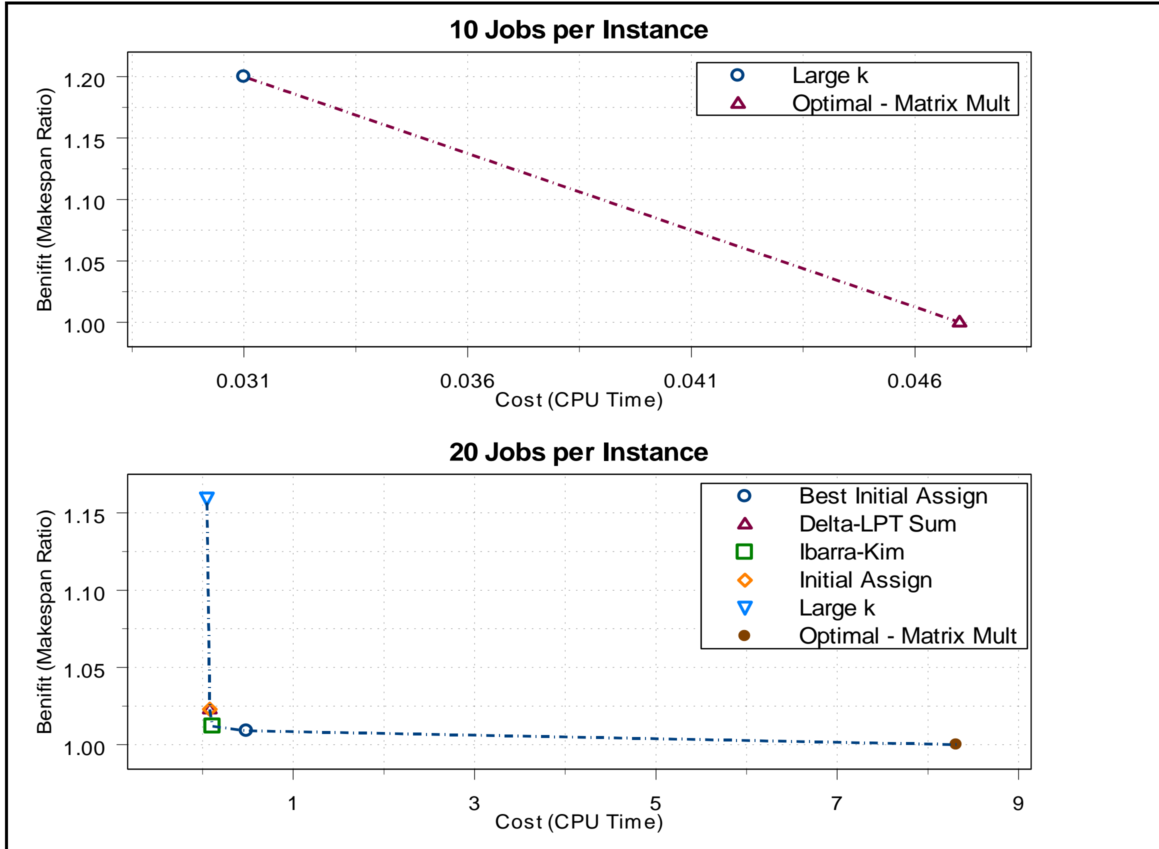

Figure 33: Tradeoff curves for dominating algorithms

All of the dominating algorithms are seen to require less than 10 seconds of CPU processing time. In reality, for the manufacturing process specifically considered in this thesis that needs to assign 20 jobs or less, algorithms requiring less than 10 seconds of CPU processing time to develop the line assignment schedule for the week are very feasible. Therefore, the most desirable optimal minimum makespan could always be achieved by utilizing the assignment schedule produced by the Complete Enumeration – Matrix Multiplication algorithm.

6.3 Future Work

Recognizing that the results presented in this thesis have a very narrow scope, i.e. they are related to a specific two line manufacturing setting considering only 10 and 20 job instances, follow up work would include a general expansion of the scope to include additional manufacturing settings (*m*-machine settings) and a more robust set of job instances. Most algorithms presented (Classical Complete Enumeration, LPT algorithms, Delta algorithms, Initial Assign algorithm, and the Large *k* algorithm) could easily be expanded to consider the *m*-machine setting. The Ibarra-Kim Algorithm F was specifically developed to handle the two-machine case; however, Ibarra and Kim [12] do present other algorithms for the general *m*-machine setting. The Complete Enumeration via Matrix Multiplication algorithm would require a larger effort to expand to the *m*-machine setting as it too was developed specifically for the two-machine case. Higher order arrays may have to be employed to represent the job assignment indicators on the additional machines. Also to be included in this expansion of scope would be the inclusion of a stochastic processing model and the assessment of the effects that variability has on the assignment algorithms.

By considering a classical statistical concept of the two-level, full factorial experimental design, a new approach to obtain the optimal assignment schedule through complete enumeration has been displayed. Extensive work in the field of statistics has been done with two-level factorial designs in the area of sample reduction with an important concept being the orthogonal fractional factorial design. An investigation of the ability to expand the approach taken in this thesis with the use

of the full factorial design to the fractional factorial design may provide extremely valuable solutions for the optimal assignment schedule.

The Large k algorithm has the positive characteristic of not relying upon the total line processing time model and was seen to have a low associated cost, however its scheduling performance was poor. This appears to be due to the fact that the number of boards, k, was not a high quality predictor of total processing time. Further work to develop a quality predictor of total processing time that still maintains the simplicity of one or several job characteristics would be beneficial. This predictor may be a linear combination of several job characteristic variables found through a principal components analysis. An algorithm found to perform well that was only dependent upon a small subset of job characteristics could be quick, easily implemented, and prove to be very valuable on the floor of a manufacturing setting.

Further heuristic development work would be to consider borrowing the ratio concept from the Ibarra-Kim algorithm for use in the Delta heuristics. The ratio used in the reassignment of jobs in the Ibarra-Kim algorithm considers those jobs that would maximize the decrease in processing time on the current line and minimize the increase in processing time on the new line to be of highest priority. By applying this concept to the Delta heuristics that currently employ only a simple difference may improve the heuristic's performance.

And finally, it has been shown that both of the parameterized heuristics, the Delta algorithms and the Initial Assign algorithm, provide high performance values of the scheduling heuristic when the optimal parameter value is selected through the "best" search algorithms. However, due to the need to search over multiple solutions, the price paid for these assignment solutions is often high. By predicting the optimal parameter based on the job characteristics prior to running the algorithm, the search would no longer be necessary and the algorithm would then only need to be run once. The result would be the same high performance at a much reduced cost.

Appendix A: Algorithm S Codes

Compute Line Processing Times Function

```
ProcessTime<-function(DataSample)
{

# Values of line performance parameters t1 – t19 in seconds
t<-c(600,60,60,60,45,3,3,5,10,8,8,8,8,20,30,30,20,120,15)


# Calculation of Line 1 processing time for each job j
TP1 <- t[1] + t[2]*(DataSample$X1 + DataSample$X2) + t[3]*DataSample$X3 +
t[4]*DataSample$X4 + t[5]* DataSample$X5 + t[14] + t[15] + t[16] + {0.5*t[6]*(DataSample$Z1 +
DataSample$Z2) + 0.5*t[7]*DataSample$Z3 + t[11]} + {0.5*t[6]*(DataSample$Z1 +
DataSample$Z2) + .5*t[7]*DataSample$Z3 + t[8]* DataSample$Z4 + t[12]} + t[17] + {t[9]*
DataSample$Z5 + t[13]} + t[18] + t[19] + (DataSample$n – 1) * pmax(t[14], t[15], t[16], (0.5*t[6]*(
DataSample$Z1 + DataSample$Z2) + 0.5*t[7]* DataSample$Z3 + t[11]), (0.5*t[6]*( DataSample$Z1
+ DataSample$Z2) + 0.5*t[7]* DataSample$Z3 + t[8]* DataSample$Z4 + t[12]), t[17], (t[9]*
DataSample$Z5 + t[13]), t[18], t[19])


# Calculation of Line 2 processing time for each job j
TP2 <- t[1] + t[2]* DataSample$X2 + t[3]* DataSample$X3 + t[4]* DataSample$X4 + t[5]*
DataSample$X5+ t[14] + t[15] + t[16] + (t[6]* DataSample$Z1 + t[10]) + (0.5*t[6]* DataSample$Z2
+ 0.5*t[7]* DataSample$Z3 + t[11]) + (0.5*t[6]* DataSample$Z2 + 0.5*t[7]* DataSample$Z3 + t[8]*
DataSample$Z4 + t[12]) + t[17] + (t[9]* DataSample$Z5 + t[13]) + t[18] + t[19]+ (DataSample$n – 1)
* pmax(t[14], t[15], t[16], (t[6]* DataSample$Z1 + t[10]), (0.5*t[6]* DataSample$Z2 + 0.5*t[7]*
DataSample$Z3 + t[11]), (0.5*t[6]* DataSample$Z2 + 0.5*t[7]* DataSample$Z3 + t[8]*
DataSample$Z4 + t[12]), t[17], (t[9]* DataSample$Z5 + t[13]), t[18], t[19])

# Add Line 1 and Line 2 processing times to Jobs data
DataSample<-data.frame(DataSample,TP1,TP2)
DataSample
}
```

Optimum Schedule via Looping Function

```
CreateMatrix<-function(num.jobs)
{
fnames<-list(c(0,1))
for (i in 1:num.jobs)
{
fnames[[i]]<-c(0,1)
}
Schedule<-as.matrix(expand.grid(fnames))
Schedule
}

Schedule10<-CreateMatrix(10)
Schedule20<-CreateMatrix(20)


OptimalLoop10<-function(SampleID,DataSample)
{

# Calculate Processing Times on Line 1 and Line 2
DataSample<-ProcessTime(DataSample)
TP1<-DataSample$TP1
TP2<-DataSample$TP2

num.jobs<-length(TP1)
BestSchedule<-array(dim=c(1,num.jobs))

# calculate the line 1 processing time for schedule 1 by using matrix multiplication
Time1<-Schedule10[1,]%*%TP1

# calculate the line 2 processing time for schedule 1 by using matrix multiplication
Time2<-(1-Schedule10[1,])%*%TP2

# calculate makespan as the maximum of line 1 processing time and line 2 processing time for
schedule 1
MS<-pmax(Time1,Time2)

#Set Best solution to solution 1 (schedule and makespan)
BestSchedule[1,]<-2-Schedule10[1,]
BestMS<-MS

#Evaluate all other solutions, one at a time – if makespan is less than best makespan then set makespan
and schedule as best

for (i in 2:2^num.jobs)
{
Time1<-Schedule10[i,]%*%TP1
Time2<-(1-Schedule10[i,])%*%TP2
MS<-pmax(Time1,Time2)
if(MS<BestMS)
{
BestSchedule[1,]<-2-Schedule10[i,]
BestMS<-MS
}
}
```

```
OptSchedule<-data.frame(SampleID,0,BestMS,1, BestSchedule)
OptSchedule
}


OptimalLoop20<-function(SampleID,DataSample)
{

# Calculate Processing Times on Line 1 and Line 2
DataSample<-ProcessTime(DataSample)
TP1<-DataSample$TP1
TP2<-DataSample$TP2

num.jobs<-length(TP1)
BestSchedule<-array(dim=c(1,num.jobs))

# calculate the line 1 processing time for schedule 1 by using matrix multiplication
Time1<-Schedule20[1,]%*%TP1

# calculate the line 2 processing time for schedule 1 by using matrix multiplication
Time2<-(1-Schedule20[1,])%*%TP2

# calculate makespan as the maximum of line 1 processing time and line 2 processing time for
schedule 1
MS<-pmax(Time1,Time2)

#Set Best solution to solution 1 (schedule and makespan)
BestSchedule[1,]<-2-Schedule20[1,]
BestMS<-MS

#Evaluate all other solutions, one at a time – if makespan is less than best makespan then set makespan
and schedule as best

for (i in 2:2^num.jobs)
{
Time1<-Schedule20[i,]%*%TP1
Time2<-(1-Schedule20[i,])%*%TP2
MS<-pmax(Time1,Time2)
if(MS<BestMS)
{
BestSchedule[1,]<-2-Schedule20[i,]
BestMS<-MS
}
}

OptSchedule<-data.frame(SampleID,0,BestMS,1, BestSchedule)
OptSchedule
}
```

Run Optimal Schedule Looping – 10 jobs, 1000 sample

```
num.samps<-1000                          #Define number of samples to run
num.jobs<-10                             # Define number of jobs in each sample

OptimalLoopSchedule10<-
as.data.frame(array(dim=c(num.samps,(4+num.jobs)),dimnames=list(NULL,c("Sample
ID","Parameter","Makespan","Relative Makespan",paste("Job",(1:num.jobs),"Assign")))))

Cost<-as.data.frame(array(dim=c(num.samps,3),dimnames=list(NULL,c("CPU Time","Elapsed
Time","Max Memory"))))

for(i in 1:num.samps)
{
mem.tally.reset()
Cost[i,1:2]<-sys.time({
OptimalLoopSchedule10[i,]<- OptimalLoop10(i,DataSample10[DataSample10$SampID==i,])
})
Cost[i,3]<- mem.tally.report()[2]
}

Method<-rep("Optimal Loop",dim(OptimalLoopSchedule10)[1])

OptimalLoopSchedule10<-data.frame(Method,Cost,OptimalLoopSchedule10)
```

Run Optimal Schedule Looping – 20 jobs, 1000 sample

```
num.samps<-1000                          #Define number of samples to run
num.jobs<-20                             # Define number of jobs in each sample

OptimalLoopSchedule20<-
as.data.frame(array(dim=c(num.samps,(4+num.jobs)),dimnames=list(NULL,c("Sample
ID","Parameter","Makespan","Relative Makespan",paste("Job",(1:num.jobs),"Assign")))))

Cost<-as.data.frame(array(dim=c(num.samps,3),dimnames=list(NULL,c("CPU Time","Elapsed
Time","Max Memory"))))

for(i in 1:num.samps)
{
mem.tally.reset()
Cost[i,1:2]<-sys.time({
OptimalLoopSchedule20[i,]<- OptimalLoop20(i,DataSample20[DataSample20$SampID==i,])
})
Cost[i,3]<- mem.tally.report()[2]
}

Method<-rep("Optimal Loop",dim(OptimalLoopSchedule20)[1])

OptimalLoopSchedule20<-data.frame(Method,Cost,OptimalLoopSchedule20)
```

Optimum Schedule via Matrix Multiplication Function

```
Optimal<-function(SampleID,DataSample)
{

# Calculate Processing Times on Line 1 and Line 2
DataSample<-ProcessTime(DataSample)
TP1<-DataSample$TP1
TP2<-DataSample$TP2


Assignment<-array(dim=c(1,length(TP1)))

# create schedule matrix where 1 indicates job scheduled on line 1 (0 indicates line 2) – this matrix is
all possible schedules for the number of jobs indicated
num.jobs<-length(TP1)
fnames<-list(c(0,1))
for (i in 1:num.jobs)
{
fnames[[i]]<-c(0,1)
}
Schedule <- as.matrix(expand.grid(fnames))

# calculate the line 1 processing time for each schedule by using matrix multiplication
Time1<-Schedule%*%TP1

# calculate the line 2 processing time for each schedule by using matrix multiplication
Time2<-(1-Schedule)%*%TP2

# calculate makespan as the maximum of line 1 processing time and line 2 processing time for each
schedule
MS<-pmax(Time1,Time2)

# find schedule that creates minimal makespan – this is the optimal schedule
k<-1:length(MS)
k<-k[MS==min(MS)][1]
Assignment[1,]<-2-Schedule[k,]

Makespan<-min(MS)

OptSchedule<-data.frame(SampleID,0,Makespan,1, Assignment)
OptSchedule

}
```

Run Optimal Schedule Matrix Multiplication – 10 jobs, 1000 sample

```
num.samps<-1000                          #Define number of samples to run
num.jobs<-10                             # Define number of jobs in each sample

OptimalSchedule10<-
as.data.frame(array(dim=c(num.samps,(4+num.jobs)),dimnames=list(NULL,c("Sample
ID","Parameter","Makespan","Relative Makespan",paste("Job",(1:num.jobs),"Assign")))))

Cost<-as.data.frame(array(dim=c(num.samps,3),dimnames=list(NULL,c("CPU Time","Elapsed
Time","Max Memory"))))

for(i in 1:num.samps)
{
mem.tally.reset()
Cost[i,1:2]<-sys.time({
OptimalSchedule10[i,]<- Optimal(i,DataSample10[DataSample10$SampID==i,])
})
Cost[i,3]<- mem.tally.report()[2]
}

Method<-rep("Optimal",dim(OptimalSchedule10)[1])
OptimalSchedule10<-data.frame(Method,Cost,OptimalSchedule10)
```

Run Optimal Schedule Matrix Multiplication – 20 jobs, 1000 sample

```
num.samps<-1000                          #Define number of samples to run
num.jobs<-20                             # Define number of jobs in each sample

OptimalSchedule20<-
as.data.frame(array(dim=c(num.samps,(4+num.jobs)),dimnames=list(NULL,c("Sample
ID","Parameter","Makespan","Relative Makespan",paste("Job",(1:num.jobs),"Assign")))))

Cost<-as.data.frame(array(dim=c(num.samps,3),dimnames=list(NULL,c("CPU Time","Elapsed
Time","Max Memory"))))

for(i in 1:num.samps)
{
mem.tally.reset()
Cost[i,1:2]<-sys.time({
OptimalSchedule20[i,]<- Optimal(i,DataSample20[DataSample20$SampID==i,])
})
Cost[i,3]<- mem.tally.report()[2]
}

Method<-rep("Optimal",dim(OptimalSchedule20)[1])
OptimalSchedule20<-data.frame(Method,Cost,OptimalSchedule20)
```

LPT Sum Function

```
LPTSum<- function(SampleID,DataSample,OptMakespan)
{

DataSample<-ProcessTime(DataSample)
TP1<-DataSample$TP1
TP2<-DataSample$TP2

Assignment<-array(dim=c(1,length(TP1)))
Schd<-rep(0, length(TP1))

nonassign<-length(TP1)                #number of non assigned jobs

#LPT rule will be used to assign unassigned jobs

#since all jobs are unassigned, assign job with largest sum processing time to line that processes it the
quickest

Schd[Schd==0][TP1[Schd==0]+TP2[Schd==0]==max(TP1[Schd==0]+TP2[Schd==0])][1]<-
((TP2<TP1)+1) [Schd==0][TP1[Schd==0]+TP2[Schd==0]==max(TP1[Schd==0]+TP2[Schd==0])][1]
nonassign<-nonassign-1

for(j in 1:nonassign)
{

Time1<-(Schd==1)%*%TP1          #processing time on line 1 for jobs currently assigned
Time2<- (Schd==2)%*%TP2         #processing time on line 2 for jobs currently assigned

Schd[Schd==0][TP1[Schd==0]+TP2[Schd==0]==max(TP1[Schd==0]+TP2[Schd==0])][1]<-
(Time2<Time1)+1            #of the subset of jobs not assigned, assign the job with the largest sum of
TP1 and TP2 to the line with the least processing time thus far; if more than one job fits criteria, assign
the first job (position 1)
}

Time1<-(Schd==1)%*%TP1          #processing time on line 1
Time2<- (Schd==2)%*%TP2         #processing time on line 2

Makespan<-max(Time1,Time2)
Assignment[1,]<-Schd

RelativeMakespan<- Makespan/OptMakespan

LPTSumSchedule<- data.frame(SampleID,"LPT",Makespan,RelativeMakespan,Assignment)
LPTSumSchedule
}
```

Run LPT Sum – 10 jobs, 1000 sample

```
num.samps<-1000                          #Define number of samples to run
num.jobs<-10                             # Define number of jobs in each sample

LPTSumSchedule10<-
as.data.frame(array(dim=c(num.samps,(4+num.jobs)),dimnames=list(NULL,c("Sample
ID","Parameter","Makespan","Relative Makespan",paste("Job",(1:num.jobs),"Assign")))))

Cost<-as.data.frame(array(dim=c(num.samps,3),dimnames=list(NULL,c("CPU Time","Elapsed
Time","Max Memory"))))

for(i in 1:num.samps)
{
mem.tally.reset()
Cost[i,1:2]<-sys.time({
LPTSumSchedule10[i,]<-
LPTSum(i,DataSample10[DataSample10$SampID==i,],OptimalSchedule10[OptimalSchedule10$Sam
ple.ID==i,]$Makespan)
})
Cost[i,3]<- mem.tally.report()[2]
}

Method<-rep("LPT Sum",dim(LPTSumSchedule10)[1])

LPTSumSchedule10<-data.frame(Method, Cost,LPTSumSchedule10)
```

Run LPT Sum – 20 jobs, 1000 sample

```
num.samps<-1000                          #Define number of samples to run
num.jobs<-20                             # Define number of jobs in each sample

LPTSumSchedule20<-
as.data.frame(array(dim=c(num.samps,(4+num.jobs)),dimnames=list(NULL,c("Sample
ID","Parameter","Makespan","Relative Makespan",paste("Job",(1:num.jobs),"Assign")))))

Cost<-as.data.frame(array(dim=c(num.samps,3),dimnames=list(NULL,c("CPU Time","Elapsed
Time","Max Memory"))))

for(i in 1:num.samps)
{
mem.tally.reset()
Cost[i,1:2]<-sys.time({
LPTSumSchedule20[i,]<-
LPTSum(i,DataSample20[DataSample20$SampID==i,],OptimalSchedule20[OptimalSchedule20$Sam
ple.ID==i,]$Makespan)
})
Cost[i,3]<- mem.tally.report()[2]
}

Method<-rep("LPT Sum",dim(LPTSumSchedule20)[1])

LPTSumSchedule20<-data.frame(Method, Cost,LPTSumSchedule20)
```

LPT Max Function

```
LPTMax<- function(SampleID,DataSample,OptMakespan)
{

DataSample<-ProcessTime(DataSample)
TP1<-DataSample$TP1
TP2<-DataSample$TP2

Assignment<-array(dim=c(1,length(TP1)))
Schd<-rep(0, length(TP1))

nonassign<-length(TP1)            #number of non assigned jobs

#LPT rule will be used to assign unassigned jobs

#since all jobs are unassigned, assign job with largest max processing time to line that processes it the quickest

Schd[Schd==0][pmax(TP1[Schd==0],TP2[Schd==0])==max(TP1[Schd==0],TP2[Schd==0])][1]<-
((TP2<TP1)+1)[Schd==0][pmax(TP1[Schd==0],TP2[Schd==0])==max(TP1[Schd==0],TP2[Schd==0]
)][1]
nonassign<-nonassign-1

for(j in 1:nonassign)
{

Time1<-(Schd==1)%*%TP1          #processing time on line 1 for jobs currently assigned
Time2<- (Schd==2)%*%TP2         #processing time on line 2 for jobs currently assigned

Schd[Schd==0][pmax(TP1[Schd==0],TP2[Schd==0])==max(TP1[Schd==0],TP2[Schd==0])][1]<-
(Time2<Time1)+1                 #of the subset of jobs not assigned, assign the job with the largest
max of TP1 and TP2 to the line with the least processing time thus far, if more than one meet the
criteria, assign the first (position 1)
}

Time1<-(Schd==1)%*%TP1          #processing time on line 1
Time2<- (Schd==2)%*%TP2         #processing time on line 2

Makespan<-max(Time1,Time2)
Assignment[1,]<-Schd

RelativeMakespan<- Makespan/OptMakespan

LPTMaxSchedule<- data.frame(SampleID,"LPT",Makespan,RelativeMakespan,Assignment)
LPTMaxSchedule
}
```

Run LPT Max – 10 jobs, 1000 sample

```
num.samps<-1000                          #Define number of samples to run
num.jobs<-10                             # Define number of jobs in each sample

LPTMaxSchedule10<-
as.data.frame(array(dim=c(num.samps,(4+num.jobs)),dimnames=list(NULL,c("Sample
ID","Parameter","Makespan","Relative Makespan",paste("Job",(1:num.jobs),"Assign")))))

Cost<-as.data.frame(array(dim=c(num.samps,3),dimnames=list(NULL,c("CPU Time","Elapsed
Time","Max Memory"))))

for(i in 1:num.samps)
{
mem.tally.reset()
Cost[i,1:2]<-sys.time({
LPTMaxSchedule10[i,]<-
LPTMax(i,DataSample10[DataSample10$SampID==i,],OptimalSchedule10[OptimalSchedule10$Sam
ple.ID==i,]$Makespan)
})
Cost[i,3]<- mem.tally.report()[2]
}

Method<-rep("LPT Max",dim(LPTMaxSchedule10)[1])

LPTMaxSchedule10<-data.frame(Method, Cost,LPTMaxSchedule10)
```

Run LPT Max – 20 jobs, 1000 sample

```
num.samps<-1000                          #Define number of samples to run
num.jobs<-20                             # Define number of jobs in each sample

LPTMaxSchedule20<-
as.data.frame(array(dim=c(num.samps,(4+num.jobs)),dimnames=list(NULL,c("Sample
ID","Parameter","Makespan","Relative Makespan",paste("Job",(1:num.jobs),"Assign")))))

Cost<-as.data.frame(array(dim=c(num.samps,3),dimnames=list(NULL,c("CPU Time","Elapsed
Time","Max Memory"))))

for(i in 1:num.samps)
{
mem.tally.reset()
Cost[i,1:2]<-sys.time({
LPTMaxSchedule20[i,]<-
LPTMax(i,DataSample20[DataSample20$SampID==i,],OptimalSchedule20[OptimalSchedule20$Sam
ple.ID==i,]$Makespan)
})
Cost[i,3]<- mem.tally.report()[2]
}

Method<-rep("LPT Max",dim(LPTMaxSchedule20)[1])

LPTMaxSchedule20<-data.frame(Method, Cost,LPTMaxSchedule20)
```

LPT Min Function

```
LPTMin<- function(SampleID,DataSample,OptMakespan)
{

DataSample<-ProcessTime(DataSample)
TP1<-DataSample$TP1
TP2<-DataSample$TP2

Assignment<-array(dim=c(1,length(TP1)))
Schd<-rep(0, length(TP1))

nonassign<-length(TP1)                    #number of non assigned jobs

#LPT rule will be used to assign unassigned jobs

#since all jobs are unassigned, assign job with largest min processing time to line that processes it the
quickest

Schd[Schd==0][pmin(TP1[Schd==0],TP2[Schd==0])==max(min(TP1[Schd==0],TP2[Schd==0]))][1]
<-
((TP2<TP1)+1)[Schd==0][pmin(TP1[Schd==0],TP2[Schd==0])==max(min(TP1[Schd==0],TP2[Schd
==0]))][1]
nonassign<-nonassign-1

for(j in 1:nonassign)
{

Time1<-(Schd==1)%*%TP1          #processing time on line 1 for jobs currently assigned
Time2<- (Schd==2)%*%TP2         #processing time on line 2 for jobs currently assigned

Schd[Schd==0][pmin(TP1[Schd==0],TP2[Schd==0])==max(min(TP1[Schd==0],TP2[Schd==0]))][1]
<- (Time2<Time1)+1                #of the subset of jobs not assigned, assign the job with the largest
min of TP1 and TP2 to the line with the least processing time thus far, if more than one meet the
criteria, assign the first (position 1)
}

Time1<-(Schd==1)%*%TP1          #processing time on line 1
Time2<- (Schd==2)%*%TP2         #processing time on line 2

Makespan<-max(Time1,Time2)
Assignment[1,]<-Schd

RelativeMakespan<- Makespan/OptMakespan

LPTMinSchedule<- data.frame(SampleID,"LPT",Makespan,RelativeMakespan,Assignment)
LPTMinSchedule
}
```

Run LPT Min – 10 jobs, 1000 sample

```
num.samps<-1000                                    #Define number of samples to run
num.jobs<-10                                       # Define number of jobs in each sample

LPTMinSchedule10<-
as.data.frame(array(dim=c(num.samps,(4+num.jobs)),dimnames=list(NULL,c("Sample
ID","Parameter","Makespan","Relative Makespan",paste("Job",(1:num.jobs),"Assign")))))

Cost<-as.data.frame(array(dim=c(num.samps,3),dimnames=list(NULL,c("CPU Time","Elapsed
Time","Max Memory"))))

for(i in 1:num.samps)
{
mem.tally.reset()
Cost[i,1:2]<-sys.time({
LPTMinSchedule10[i,]<-
LPTMin(i,DataSample10[DataSample10$SampID==i,],OptimalSchedule10[OptimalSchedule10$Sam
ple.ID==i,]$Makespan)
})
Cost[i,3]<- mem.tally.report()[2]
}

Method<-rep("LPT Min",dim(LPTMinSchedule10)[1])

LPTMinSchedule10<-data.frame(Method, Cost,LPTMinSchedule10)
```

Run LPT Min – 20 jobs, 1000 sample

```
num.samps<-1000                                    #Define number of samples to run
num.jobs<-20                                       # Define number of jobs in each sample

LPTMinSchedule20<-
as.data.frame(array(dim=c(num.samps,(4+num.jobs)),dimnames=list(NULL,c("Sample
ID","Parameter","Makespan","Relative Makespan",paste("Job",(1:num.jobs),"Assign")))))

Cost<-as.data.frame(array(dim=c(num.samps,3),dimnames=list(NULL,c("CPU Time","Elapsed
Time","Max Memory"))))

for(i in 1:num.samps)
{
mem.tally.reset()
Cost[i,1:2]<-sys.time({
LPTMinSchedule20[i,]<-
LPTMin(i,DataSample20[DataSample20$SampID==i,],OptimalSchedule20[OptimalSchedule20$Sam
ple.ID==i,]$Makespan)
})
Cost[i,3]<- mem.tally.report()[2]
}

Method<-rep("LPT Min",dim(LPTMinSchedule20)[1])

LPTMinSchedule20<-data.frame(Method, Cost,LPTMinSchedule20)
```

Delta LPT Sum Function

```
DeltaLPTSum<-function(SampleID,DataSample,OptMakespan)
{
DataSample<-ProcessTime(DataSample)
TP1<-DataSample$TP1
TP2<-DataSample$TP2

interval<-100                  #distance between Deltas – to span from 0 to 13,000
ints<-round(13000/interval)          #number of deltas to be explored

SampleID<-rep(SampleID,ints+1)
Delta<-rep(0,ints+1)
Makespan<-rep(0,ints+1)
RelativeMakespan<-rep(0,ints+1)
Assignment<-array(dim=c(ints+1,length(TP1)))

for(i in 0:ints)
{

D<-i*interval

# create schedule that assigns each job to the line that processes the job the quickest, with those jobs
that are within Delta seconds of one another assigned by LPT on TP1+TP2 rule

Schd<-rep(0, length(TP1))
Schd[TP1- TP2<(-D)]<-1
Schd[TP1- TP2>D]<-2

nonassign<-length(Schd[Schd==0])            #number of non assigned jobs

#if there are unassigned jobs, LPT rule will be used to assign these jobs

if(nonassign>0)
{

#if all jobs are unassigned, assign job with largest sum processing time to line that processes it the
quickest

if(nonassign==length(Schd))
{
Schd[Schd==0][TP1[Schd==0]+TP2[Schd==0]==max(TP1[Schd==0]+TP2[Schd==0])][1]<-
((TP2<TP1)+1) [Schd==0][TP1[Schd==0]+TP2[Schd==0]==max(TP1[Schd==0]+TP2[Schd==0])][1]
nonassign<-nonassign-1
}

for(j in 1:nonassign)
{

Time1<-(Schd==1)%*%TP1          #processing time on line 1 for jobs currently assigned
Time2<- (Schd==2)%*%TP2         #processing time on line 2 for jobs currently assigned

Schd[Schd==0][TP1[Schd==0]+TP2[Schd==0]==max(TP1[Schd==0]+TP2[Schd==0])][1]<-
(Time2<Time1)+1            #of the subset of jobs not assigned, assign the job with the maximum sum of
TP1 and TP2 to the line with the least processing time thus far; if more than one job fits criteria, assign
the first job (position 1)
```

```
}
}

Time1<-(Schd==1)%*%TP1          #processing time on line 1
Time2<- (Schd==2)%*%TP2         #processing time on line 2

Makespan[i+1]<-max(Time1,Time2)
Delta[i+1]<-D
Assignment[i+1,]<-Schd
}

RelativeMakespan<- Makespan/OptMakespan

LPTSumSchedule<- data.frame(SampleID,Delta,Makespan,RelativeMakespan, Assignment)
LPTSumSchedule
}
```

Best Delta LPT Sum Function

```
#Calls Delta LPT Sum function to return all possible delta solutions; then searches and returns the best
delta solution

BestDeltaLPTSum<-function(SampleID,DataSample,OptMakespan)
{

All<-DeltaLPTSum(SampleID,DataSample,OptMakespan)    #Return all possible delta solutions

Best<-All[All$RelativeMakespan==min(All$RelativeMakespan),][1,]   #Find solution with minimal
Relative Makespan; if more than one exists, return the first

Best
}
```

Run Best Delta LPT Sum – 10 jobs, 1000 sample

```
num.samps<-1000                    #Define number of samples to run
num.jobs<-10

BestDeltaSumSchedule10<-as.data.frame(array(dim=c(num.samps,
(4+num.jobs)),dimnames=list(NULL,c("Sample ID","Parameter","Makespan","Relative
Makespan",paste("Job",(1:num.jobs),"Assign")))))

Cost<-as.data.frame(array(dim=c(num.samps,3),dimnames=list(NULL,c("CPU Time","Elapsed
Time","Max Memory"))))

for(i in 1:num.samps)
{
mem.tally.reset()
Cost[i,1:2]<-sys.time({
BestDeltaSumSchedule10[i,]<-
BestDeltaLPTSum(i,DataSample10[DataSample10$SampID==i,],OptimalSchedule10[OptimalSchedul
e10$Sample.ID==i,]$Makespan)
})
Cost[i,3]<- mem.tally.report()[2]
}

Method<-rep("Best Delta LPT Sum",dim(BestDeltaSumSchedule10)[1])

BestDeltaSumSchedule10<-data.frame(Method,Cost,BestDeltaSumSchedule10)
```

Run Best Delta LPT Sum – 20 jobs, 1000 sample

```
num.samps<-1000                    #Define number of samples to run
num.jobs<-20

BestDeltaSumSchedule20<-as.data.frame(array(dim=c(num.samps,
(4+num.jobs)),dimnames=list(NULL,c("Sample ID","Parameter","Makespan","Relative
Makespan",paste("Job",(1:num.jobs),"Assign")))))

Cost<-as.data.frame(array(dim=c(num.samps,3),dimnames=list(NULL,c("CPU Time","Elapsed
Time","Max Memory"))))

for(i in 1:num.samps)
{
mem.tally.reset()
Cost[i,1:2]<-sys.time({
BestDeltaSumSchedule20[i,]<-
BestDeltaLPTSum(i,DataSample20[DataSample20$SampID==i,],OptimalSchedule20[OptimalSchedul
e20$Sample.ID==i,]$Makespan)
})
Cost[i,3]<- mem.tally.report()[2]
}

Method<-rep("Best Delta LPT Sum",dim(BestDeltaSumSchedule20)[1])

BestDeltaSumSchedule20<-data.frame(Method,Cost,BestDeltaSumSchedule20)
```

Delta LPT Sum Function (Δ = 4000)

```
DeltaLPTSum<-function(SampleID,DataSample,OptMakespan)
{

DataSample<-ProcessTime(DataSample)
TP1<-DataSample$TP1
TP2<-DataSample$TP2

Assignment<-array(dim=c(1,length(TP1)))

D<-4000

# create schedule that assigns each job to the line that processes the job the quickest, with those jobs
that are within 4000 seconds of one another assigned by LPT on TP1+TP2 rule
Schd<-rep(0, length(TP1))
Schd[TP1- TP2<(-D)]<-1
Schd[TP1- TP2>D]<-2

nonassign<-length(Schd[Schd==0])              #number of non assigned jobs

#if there are unassigned jobs, LPT rule will be used to assign these jobs
if(nonassign>0)
{

#if all jobs are unassigned, assign job with largest sum processing time to line that processes it the
quickest
if(nonassign==length(Schd))
{
Schd[Schd==0][TP1[Schd==0]+TP2[Schd==0]==max(TP1[Schd==0]+TP2[Schd==0])][1]<-
((TP2<TP1)+1) [Schd==0][TP1[Schd==0]+TP2[Schd==0]==max(TP1[Schd==0]+TP2[Schd==0])][1]
nonassign<-nonassign-1
}

for(j in 1:nonassign)
{

Time1<-(Schd==1)%*%TP1          #processing time on line 1 for jobs currently assigned
Time2<- (Schd==2)%*%TP2         #processing time on line 2 for jobs currently assigned

Schd[Schd==0][TP1[Schd==0]+TP2[Schd==0]==max(TP1[Schd==0]+TP2[Schd==0])][1]<-
(Time2<Time1)+1            #of the subset of jobs not assigned, assign the job with the largest sum of
TP1 and TP2 to the line with the least processing time thus far; if more than one job fits criteria, assign
the first job (position 1)
}
}

Time1<-(Schd==1)%*%TP1          #processing time on line 1
Time2<- (Schd==2)%*%TP2         #processing time on line 2
Makespan<-max(Time1,Time2)
Assignment[1,]<-Schd

RelativeMakespan<- Makespan/OptMakespan
LPTSumSchedule<- data.frame(SampleID,4000,Makespan,RelativeMakespan,Assignment)
LPTSumSchedule
}
```

Run Delta LPT Sum Δ = 4000 – 10 jobs, 1000 sample

```
num.samps<-1000                        #Define number of samples to run
num.jobs<-10                           # Define number of jobs in each sample

Delta4000SumSchedule10<-
as.data.frame(array(dim=c(num.samps,(4+num.jobs)),dimnames=list(NULL,c("Sample
ID","Parameter","Makespan","Relative Makespan",paste("Job",(1:num.jobs),"Assign")))))

Cost<-as.data.frame(array(dim=c(num.samps,3),dimnames=list(NULL,c("CPU Time","Elapsed
Time","Max Memory"))))

for(i in 1:num.samps)
{
mem.tally.reset()
Cost[i,1:2]<-sys.time({
Delta4000SumSchedule10[i,]<-
DeltaLPTSum(i,DataSample10[DataSample10$SampID==i,],OptimalSchedule10[OptimalSchedule10
$Sample.ID==i,]$Makespan)
})
Cost[i,3]<- mem.tally.report()[2]
}

Method<-rep("Delta LPT Sum = 4000",dim(Delta4000SumSchedule10)[1])

Delta4000SumSchedule10<-data.frame(Method,Cost,Delta4000SumSchedule10)
```

Delta LPT Sum (Δ = 2700) Function

```
DeltaLPTSum<-function(SampleID,DataSample,OptMakespan)
{

DataSample<-ProcessTime(DataSample)
TP1<-DataSample$TP1
TP2<-DataSample$TP2

Assignment<-array(dim=c(1,length(TP1)))
D<-2700

# create schedule that assigns each job to the line that processes the job the quickest, with those jobs
that are within 2700 seconds of one another assigned by LPT on TP1+TP2 rule
Schd<-rep(0, length(TP1))
Schd[TP1- TP2<(-D)]<-1
Schd[TP1- TP2>D]<-2

nonassign<-length(Schd[Schd==0])             #number of non assigned jobs

#if there are unassigned jobs, LPT rule will be used to assign these jobs
if(nonassign>0)
{

#if all jobs are unassigned, assign job with largest sum processing time to line that processes it the
quickest
if(nonassign==length(Schd))
{
Schd[Schd==0][TP1[Schd==0]+TP2[Schd==0]==max(TP1[Schd==0]+TP2[Schd==0])][1]<-
((TP2<TP1)+1) [Schd==0][TP1[Schd==0]+TP2[Schd==0]==max(TP1[Schd==0]+TP2[Schd==0])][1]
nonassign<-nonassign-1
}

for(j in 1:nonassign)
{

Time1<-(Schd==1)%*%TP1          #processing time on line 1 for jobs currently assigned
Time2<- (Schd==2)%*%TP2         #processing time on line 2 for jobs currently assigned

Schd[Schd==0][TP1[Schd==0]+TP2[Schd==0]==max(TP1[Schd==0]+TP2[Schd==0])][1]<-
(Time2<Time1)+1            #of the subset of jobs not assigned, assign the job with the largest sum of
TP1 and TP2 to the line with the least processing time thus far; if more than one job fits criteria, assign
the first job (position 1)
}
}

Time1<-(Schd==1)%*%TP1          #processing time on line 1
Time2<- (Schd==2)%*%TP2         #processing time on line 2

Makespan<-max(Time1,Time2)
Assignment[1,]<-Schd

RelativeMakespan<- Makespan/OptMakespan
LPTSumSchedule<- data.frame(SampleID,4000,Makespan,RelativeMakespan,Assignment)
LPTSumSchedule
}
```

Run Delta LPT Sum (Δ = 2700) – 20 jobs, 1000 sample

```
num.samps<-1000                          #Define number of samples to run
num.jobs<-20                             # Define number of jobs in each sample

Delta2700SumSchedule20<-
as.data.frame(array(dim=c(num.samps,(4+num.jobs)),dimnames=list(NULL,c("Sample
ID","Parameter","Makespan","Relative Makespan",paste("Job",(1:num.jobs),"Assign")))))

Cost<-as.data.frame(array(dim=c(num.samps,3),dimnames=list(NULL,c("CPU Time","Elapsed
Time","Max Memory"))))

for(i in 1:num.samps)
{
mem.tally.reset()
Cost[i,1:2]<-sys.time({
Delta2700SumSchedule20[i,]<-
DeltaLPTSum(i,DataSample20[DataSample20$SampID==i,],OptimalSchedule20[OptimalSchedule20
$Sample.ID==i,]$Makespan)
})
Cost[i,3]<- mem.tally.report()[2]
}

Method<-rep("Delta LPT Sum = 2700",dim(Delta2700SumSchedule20)[1])

Delta2700SumSchedule20<-data.frame(Method, Cost,Delta2700SumSchedule20)
```

Delta LPT Max Function

```
DeltaLPTMax<-function(SampleID,DataSample,OptMakespan)
{
DataSample<-ProcessTime(DataSample)
TP1<-DataSample$TP1
TP2<-DataSample$TP2

interval<-100                    #distance between Deltas – to span from 0 to 13,000
ints<-round(13000/interval)            #number of deltas to be explored

SampleID<-rep(SampleID,ints+1)
Delta<-rep(0,ints+1)
Makespan<-rep(0,ints+1)
RelativeMakespan<-rep(0,ints+1)
Assignment<-array(dim=c(ints+1,length(TP1)))

for(i in 0:ints)
{

D<-i*interval

# create schedule that assigns each job to the line that processes the job the quickest, with those jobs
that are within Delta seconds of one another assigned by LPT on max(TP1,TP2) rule

Schd<-rep(0, length(TP1))
Schd[TP1- TP2<(-D)]<-1
Schd[TP1- TP2>D]<-2

nonassign<-length(Schd[Schd==0])             #number of non assigned jobs

#if there are unassigned jobs, LPT rule will be used to assign these jobs

if(nonassign>0)
{

#if all jobs are unassigned, assign job with largest max processing time to line that processes it the
quickest

if(nonassign==length(Schd))
{
Schd[Schd==0][pmax(TP1[Schd==0],TP2[Schd==0])==max(TP1[Schd==0],TP2[Schd==0])][1]<-
((TP2<TP1)+1)[Schd==0][pmax(TP1[Schd==0],TP2[Schd==0])==max(TP1[Schd==0],TP2[Schd==0]
)][1]
nonassign<-nonassign-1
}

for(j in 1:nonassign)
{

Time1<-(Schd==1)%*%TP1          #processing time on line 1 for jobs currently assigned
Time2<- (Schd==2)%*%TP2         #processing time on line 2 for jobs currently assigned

Schd[Schd==0][pmax(TP1[Schd==0],TP2[Schd==0])==max(TP1[Schd==0],TP2[Schd==0])][1]<-
(Time2<Time1)+1                 #of the subset of jobs not assigned, assign the job with the
```

```
maximum max of TP1 and TP2 to the line with the least processing time thus far, if more than one
meet the criteria, assign the first (position 1)
}
}

Time1<-(Schd==1)%*%TP1          #processing time on line 1
Time2<- (Schd==2)%*%TP2         #processing time on line 2

Makespan[i+1]<-max(Time1,Time2)
Delta[i+1]<-D
Assignment[i+1,]<-Schd
}

RelativeMakespan<- Makespan/OptMakespan

LPTMaxSchedule<- data.frame(SampleID,Delta,Makespan,RelativeMakespan, Assignment)
LPTMaxSchedule
}
```

Best Delta LPT Max Function

```
#Calls Delta LPT Max function to return all possible delta solutions; then searches and returns the best
delta solution

BestDeltaLPTMax<-function(SampleID,DataSample,OptMakespan)
{

All<-DeltaLPTMax(SampleID,DataSample,OptMakespan)    #Return all possible delta solutions

Best<-All[All$RelativeMakespan==min(All$RelativeMakespan),][1,]   #Find solution with minimal
Relative Makespan; if more than one exists, return the first

Best
}
```

Run Best Delta LPT Max Function – 10 jobs, 1000 sample

```
num.samps<-1000                    #Define number of samples to run
num.jobs<-10

BestDeltaMaxSchedule10<-as.data.frame(array(dim=c(num.samps,
(4+num.jobs)),dimnames=list(NULL,c("Sample ID","Parameter","Makespan","Relative
Makespan",paste("Job",(1:num.jobs),"Assign")))))

Cost<-as.data.frame(array(dim=c(num.samps,3),dimnames=list(NULL,c("CPU Time","Elapsed
Time","Max Memory"))))

for(i in 1:num.samps)
{
mem.tally.reset()
Cost[i,1:2]<-sys.time({
BestDeltaMaxSchedule10[i,]<-
BestDeltaLPTMax(i,DataSample10[DataSample10$SampID==i,],OptimalSchedule10[OptimalSchedul
e10$Sample.ID==i,]$Makespan)
})
Cost[i,3]<- mem.tally.report()[2]
}

Method<-rep("Best Delta LPT Max",dim(BestDeltaMaxSchedule10)[1])

BestDeltaMaxSchedule10<-data.frame(Method,Cost,BestDeltaMaxSchedule10)
```

Run Best Delta LPT Max Function – 20 jobs, 1000 sample

```
num.samps<-1000                    #Define number of samples to run
num.jobs<-20

BestDeltaMaxSchedule20<-as.data.frame(array(dim=c(num.samps,
(4+num.jobs)),dimnames=list(NULL,c("Sample ID","Parameter","Makespan","Relative
Makespan",paste("Job",(1:num.jobs),"Assign")))))


Cost<-as.data.frame(array(dim=c(num.samps,3),dimnames=list(NULL,c("CPU Time","Elapsed
Time","Max Memory"))))

for(i in 1:num.samps)
{
mem.tally.reset()
Cost[i,1:2]<-sys.time({
BestDeltaMaxSchedule20[i,]<-
BestDeltaLPTMax(i,DataSample20[DataSample20$SampID==i,],OptimalSchedule20[OptimalSchedul
e20$Sample.ID==i,]$Makespan)
})
Cost[i,3]<- mem.tally.report()[2]
}

Method<-rep("Best Delta LPT Max",dim(BestDeltaMaxSchedule20)[1])

BestDeltaMaxSchedule20<-data.frame(Method,Cost,BestDeltaMaxSchedule20)
```

Delta LPT Max (Δ = 4000) Function

```
DeltaLPTMax<-function(SampleID,DataSample,OptMakespan)
{

DataSample<-ProcessTime(DataSample)
TP1<-DataSample$TP1
TP2<-DataSample$TP2

Assignment<-array(dim=c(1,length(TP1)))
D<-4000

# create schedule that assigns each job to the line that processes the job the quickest, with those jobs
that are within 4000 seconds of one another assigned by LPT on Max(TP1,TP2) rule
Schd<-rep(0, length(TP1))
Schd[TP1- TP2<(-D)]<-1
Schd[TP1- TP2>D]<-2

nonassign<-length(Schd[Schd==0])              #number of non assigned jobs

#if there are unassigned jobs, LPT rule will be used to assign these jobs
if(nonassign>0)
{

#if all jobs are unassigned, assign job with largest max processing time to line that processes it the
quickest
if(nonassign==length(Schd))
{
Schd[Schd==0][pmax(TP1[Schd==0],TP2[Schd==0])==max(TP1[Schd==0],TP2[Schd==0])][1]<-
((TP2<TP1)+1)[Schd==0][pmax(TP1[Schd==0],TP2[Schd==0])==max(TP1[Schd==0],TP2[Schd==0]
)][1]
nonassign<-nonassign-1
}

for(j in 1:nonassign)
{

Time1<-(Schd==1)%*%TP1          #processing time on line 1 for jobs currently assigned
Time2<- (Schd==2)%*%TP2         #processing time on line 2 for jobs currently assigned

Schd[Schd==0][pmax(TP1[Schd==0],TP2[Schd==0])==max(TP1[Schd==0],TP2[Schd==0])][1]<-
(Time2<Time1)+1                 #of the subset of jobs not assigned, assign the job with the largest
max of TP1 and TP2 to the line with the least processing time thus far, if more than one meet the
criteria, assign the first (position 1)
}
}

Time1<-(Schd==1)%*%TP1          #processing time on line 1
Time2<- (Schd==2)%*%TP2         #processing time on line 2

Makespan<-max(Time1,Time2)
Assignment[1,]<-Schd
RelativeMakespan<- Makespan/OptMakespan
LPTMaxSchedule<- data.frame(SampleID,4000,Makespan,RelativeMakespan,Assignment)
LPTMaxSchedule
}
```

Run Delta LPT Max (Δ = 4000) – 10 jobs, 1000 sample

```
num.samps<-1000                           #Define number of samples to run
num.jobs<-10                              # Define number of jobs in each sample

Delta4000MaxSchedule10<-
as.data.frame(array(dim=c(num.samps,(4+num.jobs)),dimnames=list(NULL,c("Sample
ID","Parameter","Makespan","Relative Makespan",paste("Job",(1:num.jobs),"Assign")))))

Cost<-as.data.frame(array(dim=c(num.samps,3),dimnames=list(NULL,c("CPU Time","Elapsed
Time","Max Memory"))))

for(i in 1:num.samps)
{
mem.tally.reset()
Cost[i,1:2]<-sys.time({
Delta4000MaxSchedule10[i,]<-
DeltaLPTMax(i,DataSample10[DataSample10$SampID==i,],OptimalSchedule10[OptimalSchedule10
$Sample.ID==i,]$Makespan)
})
Cost[i,3]<- mem.tally.report()[2]
}

Method<-rep("Delta LPT Max = 4000",dim(Delta4000MaxSchedule10)[1])

Delta4000MaxSchedule10<-data.frame(Method, Cost,Delta4000MaxSchedule10)
```

Delta LPT Max (Δ = 2700) Function

```
DeltaLPTMax<-function(SampleID,DataSample,OptMakespan)
{

DataSample<-ProcessTime(DataSample)
TP1<-DataSample$TP1
TP2<-DataSample$TP2

Assignment<-array(dim=c(1,length(TP1)))
D<-2700

# create schedule that assigns each job to the line that processes the job the quickest, with those jobs
that are within 4000 seconds of one another assigned by LPT on Max(TP1,TP2) rule
Schd<-rep(0, length(TP1))
Schd[TP1- TP2<(-D)]<-1
Schd[TP1- TP2>D]<-2

nonassign<-length(Schd[Schd==0])            #number of non assigned jobs

#if there are unassigned jobs, LPT rule will be used to assign these jobs
if(nonassign>0)
{

#if all jobs are unassigned, assign job with largest max processing time to line that processes it the
quickest
if(nonassign==length(Schd))
{
Schd[Schd==0][pmax(TP1[Schd==0],TP2[Schd==0])==max(TP1[Schd==0],TP2[Schd==0])][1]<-
((TP2<TP1)+1)[Schd==0][pmax(TP1[Schd==0],TP2[Schd==0])==max(TP1[Schd==0],TP2[Schd==0]
)][1]
nonassign<-nonassign-1
}

for(j in 1:nonassign)
{

Time1<-(Schd==1)%*%TP1          #processing time on line 1 for jobs currently assigned
Time2<- (Schd==2)%*%TP2         #processing time on line 2 for jobs currently assigned

Schd[Schd==0][pmax(TP1[Schd==0],TP2[Schd==0])==max(TP1[Schd==0],TP2[Schd==0])][1]<-
(Time2<Time1)+1                 #of the subset of jobs not assigned, assign the job with the largest
max of TP1 and TP2 to the line with the least processing time thus far, if more than one meet the
criteria, assign the first (position 1)
}
}

Time1<-(Schd==1)%*%TP1          #processing time on line 1
Time2<- (Schd==2)%*%TP2         #processing time on line 2

Makespan<-max(Time1,Time2)
Assignment[1,]<-Schd
RelativeMakespan<- Makespan/OptMakespan
LPTMaxSchedule<- data.frame(SampleID,4000,Makespan,RelativeMakespan,Assignment)
LPTMaxSchedule
}
```

Run Delta LPT Max (Δ = 2700) – 20 jobs, 1000 sample

```
num.samps<-1000                          #Define number of samples to run
num.jobs<-20                             # Define number of jobs in each sample

Delta2700MaxSchedule20<-
as.data.frame(array(dim=c(num.samps,(4+num.jobs)),dimnames=list(NULL,c("Sample
ID","Parameter","Makespan","Relative Makespan",paste("Job",(1:num.jobs),"Assign")))))

Cost<-as.data.frame(array(dim=c(num.samps,3),dimnames=list(NULL,c("CPU Time","Elapsed
Time","Max Memory"))))

for(i in 1:num.samps)
{
mem.tally.reset()
Cost[i,1:2]<-sys.time({
Delta2700MaxSchedule20[i,]<-
DeltaLPTMax(i,DataSample20[DataSample20$SampID==i,],OptimalSchedule20[OptimalSchedule20
$Sample.ID==i,]$Makespan)
})
Cost[i,3]<- mem.tally.report()[2]
}

Method<-rep("Delta LPT Max = 2700",dim(Delta2700MaxSchedule20)[1])

Delta2700MaxSchedule20<-data.frame(Method,Cost,Delta2700MaxSchedule20)
```

Delta LPT Min Function

```
DeltaLPTMin<-function(SampleID,DataSample,OptMakespan)
{

DataSample<-ProcessTime(DataSample)
TP1<-DataSample$TP1
TP2<-DataSample$TP2

interval<-100                   #distance between Deltas – to span from 0 to 13,000
ints<-round(13000/interval)            #number of deltas to be explored

SampleID<-rep(SampleID,ints+1)
Delta<-rep(0,ints+1)
Makespan<-rep(0,ints+1)
RelativeMakespan<-rep(0,ints+1)
Assignment<-array(dim=c(ints+1,length(TP1)))

for(i in 0:ints)
{

D<-i*interval

# create schedule that assigns each job to the line that processes the job the quickest, with those jobs
that are within Delta seconds of one another assigned by LPT on min(TP1,TP2) rule

Schd<-rep(0, length(TP1))
Schd[TP1- TP2<(-D)]<-1
Schd[TP1- TP2>D]<-2

nonassign<-length(Schd[Schd==0])             #number of non assigned jobs

#if there are unassigned jobs, LPT rule will be used to assign these jobs

if(nonassign>0)
{

#if all jobs are unassigned, assign job with largest Min processing time to line that processes it the
quickest

if(nonassign==length(Schd))
{
Schd[Schd==0][pmin(TP1[Schd==0],TP2[Schd==0])==max(min(TP1[Schd==0],TP2[Schd==0]))][1]
<-
((TP2<TP1)+1)[Schd==0][pmin(TP1[Schd==0],TP2[Schd==0])==max(min(TP1[Schd==0],TP2[Schd
==0]))][1]
nonassign<-nonassign-1
}

for(j in 1:nonassign)
{

Time1<-(Schd==1)%*%TP1          #processing time on line 1 for jobs currently assigned
Time2<- (Schd==2)%*%TP2         #processing time on line 2 for jobs currently assigned
```

```
Schd[Schd==0][pmin(TP1[Schd==0],TP2[Schd==0])==max(min(TP1[Schd==0],TP2[Schd==0]))][1]
<- (Time2<Time1)+1                    #of the subset of jobs not assigned, assign the job with the
maximum min of TP1 and TP2 to the line with the least processing time thus far, if more than one
meet the criteria, assign the first (position 1)
}
}

Time1<-(Schd==1)%*%TP1          #processing time on line 1
Time2<- (Schd==2)%*%TP2         #processing time on line 2

Makespan[i+1]<-max(Time1,Time2)
Delta[i+1]<-D
Assignment[i+1,]<-Schd
}

RelativeMakespan<- Makespan/OptMakespan

LPTMinSchedule<- data.frame(SampleID,Delta,Makespan,RelativeMakespan, Assignment)
LPTMinSchedule
}
```

Best Delta LPT Min Function

```
#Calls Delta LPT Min function to return all possible delta solutions; then searches and returns the best
delta solution

BestDeltaLPTMin<-function(SampleID,DataSample,OptMakespan)
{

All<-DeltaLPTMin(SampleID,DataSample,OptMakespan)    #Return all possible delta solutions

Best<-All[All$RelativeMakespan==min(All$RelativeMakespan),][1,]    #Find solution with minimal
Relative Makespan; if more than one exists, return the first

Best
}
```

Run Best Delta LPT Min Function – 10 jobs, 1000 sample

```
num.samps<-1000                    #Define number of samples to run
num.jobs<-10

BestDeltaMinSchedule10<-as.data.frame(array(dim=c(num.samps,
(4+num.jobs)),dimnames=list(NULL,c("Sample ID","Parameter","Makespan","Relative
Makespan",paste("Job",(1:num.jobs),"Assign")))))

Cost<-as.data.frame(array(dim=c(num.samps,3),dimnames=list(NULL,c("CPU Time","Elapsed
Time","Max Memory"))))

for(i in 1:num.samps)
{
mem.tally.reset()
Cost[i,1:2]<-sys.time({
BestDeltaMinSchedule10[i,]<-
BestDeltaLPTMin(i,DataSample10[DataSample10$SampID==i,],OptimalSchedule10[OptimalSchedul
e10$Sample.ID==i,]$Makespan)
})
Cost[i,3]<- mem.tally.report()[2]
}

Method<-rep("Best Delta LPT Min",dim(BestDeltaMinSchedule10)[1])

BestDeltaMinSchedule10<-data.frame(Method, Cost,BestDeltaMinSchedule10)
```

Run Best Delta LPT Min Schedule – 20 jobs, 1000 sample

```
num.samps<-1000                    #Define number of samples to run
num.jobs<-20

BestDeltaMinSchedule20<-as.data.frame(array(dim=c(num.samps,
(4+num.jobs)),dimnames=list(NULL,c("Sample ID","Parameter","Makespan","Relative
Makespan",paste("Job",(1:num.jobs),"Assign")))))


Cost<-as.data.frame(array(dim=c(num.samps,3),dimnames=list(NULL,c("CPU Time","Elapsed
Time","Max Memory"))))

for(i in 1:num.samps)
{
mem.tally.reset()
Cost[i,1:2]<-sys.time({
BestDeltaMinSchedule20[i,]<-
BestDeltaLPTMin(i,DataSample20[DataSample20$SampID==i,],OptimalSchedule20[OptimalSchedul
e20$Sample.ID==i,]$Makespan)
})
Cost[i,3]<- mem.tally.report()[2]
}

Method<-rep("Best Delta LPT Min",dim(BestDeltaMinSchedule20)[1])

BestDeltaMinSchedule20<-data.frame(Method,Cost,BestDeltaMinSchedule20)
```

Delta LPT Min (Δ = 2700) Function

```
DeltaLPTMin<-function(SampleID,DataSample,OptMakespan)
{

DataSample<-ProcessTime(DataSample)
TP1<-DataSample$TP1
TP2<-DataSample$TP2

Assignment<-array(dim=c(1,length(TP1)))
D<-2700

# create schedule that assigns each job to the line that processes the job the quickest, with those jobs
that are within 4000 seconds of one another assigned by LPT on Min(TP1,TP2) rule
Schd<-rep(0, length(TP1))
Schd[TP1- TP2<(-D)]<-1
Schd[TP1- TP2>D]<-2

nonassign<-length(Schd[Schd==0])            #number of non assigned jobs

#if there are unassigned jobs, LPT rule will be used to assign these jobs
if(nonassign>0)
{

#if all jobs are unassigned, assign job with largest min processing time to line that processes it the
quickest
if(nonassign==length(Schd))
{
Schd[Schd==0][pmin(TP1[Schd==0],TP2[Schd==0])==max(min(TP1[Schd==0],TP2[Schd==0]))][1]
<-
((TP2<TP1)+1)[Schd==0][pmin(TP1[Schd==0],TP2[Schd==0])==max(min(TP1[Schd==0],TP2[Schd
==0]))][1]
nonassign<-nonassign-1
}

for(j in 1:nonassign)
{

Time1<-(Schd==1)%*%TP1          #processing time on line 1 for jobs currently assigned
Time2<- (Schd==2)%*%TP2         #processing time on line 2 for jobs currently assigned

Schd[Schd==0][pmin(TP1[Schd==0],TP2[Schd==0])==max(min(TP1[Schd==0],TP2[Schd==0]))][1]
<- (Time2<Time1)+1              #of the subset of jobs not assigned, assign the job with the largest
min of TP1 and TP2 to the line with the least processing time thus far, if more than one meet the
criteria, assign the first (position 1)
}
}
Time1<-(Schd==1)%*%TP1          #processing time on line 1
Time2<- (Schd==2)%*%TP2         #processing time on line 2

Makespan<-max(Time1,Time2)
Assignment[1,]<-Schd
RelativeMakespan<- Makespan/OptMakespan
LPTMinSchedule<- data.frame(SampleID,2700,Makespan,RelativeMakespan,Assignment)
LPTMinSchedule
}
```

Run Delta LPT Min (Δ = 2700) – 10 jobs, 1000 sample

```
num.samps<-1000                         #Define number of samples to run
num.jobs<-10                            # Define number of jobs in each sample

Delta27000MinSchedule10<-
as.data.frame(array(dim=c(num.samps,(4+num.jobs)),dimnames=list(NULL,c("Sample
ID","Parameter","Makespan","Relative Makespan",paste("Job",(1:num.jobs),"Assign")))))

Cost<-as.data.frame(array(dim=c(num.samps,3),dimnames=list(NULL,c("CPU Time","Elapsed
Time","Max Memory"))))

for(i in 1:num.samps)
{
mem.tally.reset()
Cost[i,1:2]<-sys.time({
Delta27000MinSchedule10[i,]<-
DeltaLPTMin(i,DataSample10[DataSample10$SampID==i,],OptimalSchedule10[OptimalSchedule10$
Sample.ID==i,]$Makespan)
})
Cost[i,3]<- mem.tally.report()[2]
}

Method<-rep("Delta LPT Min = 27000",dim(Delta27000MinSchedule10)[1])

Delta27000MinSchedule10<-data.frame(Method, Cost,Delta27000MinSchedule10)
```

Delta LPT Min (Δ = 3500)

```
DeltaLPTMin<-function(SampleID,DataSample,OptMakespan)
{

DataSample<-ProcessTime(DataSample)
TP1<-DataSample$TP1
TP2<-DataSample$TP2

Assignment<-array(dim=c(1,length(TP1)))
D<-3500

# create schedule that assigns each job to the line that processes the job the quickest, with those jobs
that are within 4000 seconds of one another assigned by LPT on Min(TP1,TP2) rule
Schd<-rep(0, length(TP1))
Schd[TP1- TP2<(-D)]<-1
Schd[TP1- TP2>D]<-2

nonassign<-length(Schd[Schd==0])             #number of non assigned jobs

#if there are unassigned jobs, LPT rule will be used to assign these jobs
if(nonassign>0)
{

#if all jobs are unassigned, assign job with largest min processing time to line that processes it the
quickest
if(nonassign==length(Schd))
{
Schd[Schd==0][pmin(TP1[Schd==0],TP2[Schd==0])==max(min(TP1[Schd==0],TP2[Schd==0]))][1]
<-
((TP2<TP1)+1)[Schd==0][pmin(TP1[Schd==0],TP2[Schd==0])==max(min(TP1[Schd==0],TP2[Schd
==0]))][1]
nonassign<-nonassign-1
}

for(j in 1:nonassign)
{

Time1<-(Schd==1)%*%TP1          #processing time on line 1 for jobs currently assigned
Time2<- (Schd==2)%*%TP2         #processing time on line 2 for jobs currently assigned

Schd[Schd==0][pmin(TP1[Schd==0],TP2[Schd==0])==max(min(TP1[Schd==0],TP2[Schd==0]))][1]
<- (Time2<Time1)+1                      #of the subset of jobs not assigned, assign the job with the largest
min of TP1 and TP2 to the line with the least processing time thus far, if more than one meet the
criteria, assign the first (position 1)
}
}
Time1<-(Schd==1)%*%TP1          #processing time on line 1
Time2<- (Schd==2)%*%TP2         #processing time on line 2

Makespan<-max(Time1,Time2)
Assignment[1,]<-Schd
RelativeMakespan<- Makespan/OptMakespan
LPTMinSchedule<- data.frame(SampleID,2700,Makespan,RelativeMakespan,Assignment)
LPTMinSchedule
}
```

Run Delta LPT Min (Δ = 3500) – 20 jobs, 1000 sample

```
num.samps<-1000                                  #Define number of samples to run
num.jobs<-20                                     # Define number of jobs in each sample

Delta3500MinSchedule20<-
as.data.frame(array(dim=c(num.samps,(4+num.jobs)),dimnames=list(NULL,c("Sample
ID","Parameter","Makespan","Relative Makespan",paste("Job",(1:num.jobs),"Assign")))))

Cost<-as.data.frame(array(dim=c(num.samps,3),dimnames=list(NULL,c("CPU Time","Elapsed
Time","Max Memory"))))

for(i in 1:num.samps)
{
mem.tally.reset()
Cost[i,1:2]<-sys.time({
Delta3500MinSchedule20[i,]<-
DeltaLPTMin(i,DataSample20[DataSample20$SampID==i,],OptimalSchedule20[OptimalSchedule20$
Sample.ID==i,]$Makespan)
})
Cost[i,3]<- mem.tally.report()[2]
}

Method<-rep("Delta LPT Min = 3500",dim(Delta3500MinSchedule20)[1])

Delta3500MinSchedule20<-data.frame(Method,Cost,Delta3500MinSchedule20)
```

Initial Assign Function

```
#initially assign some number of jobs to the line that processes them the fastest; for the remaining jobs, assign using LPT sum rule

InitialAssign<-function(SampleID,DataSample,OptMakespan)
{

DataSample<-ProcessTime(DataSample)
TP1<-DataSample$TP1
TP2<-DataSample$TP2

ints<-13     #length(TP1)    #number of initial assigns to be explored – 1 as 0 is not represented

SampleID<-rep(SampleID,(ints+1))
Initial<-0:ints
Makespan<-rep(0, (ints+1))
RelativeMakespan<-rep(0, (ints+1))
Assignment<-array(dim=c((ints+1),length(TP1)))
diff<-abs(TP1-TP2)

for(i in 0:ints)
{
# create schedule that assigns the i number of jobs with the largest difference between TP1 and TP2 to the line that processes the job the quickest, the remaining jobs are assigned by LPT on Sum of TP1 & TP2 rule

Schd<-rep(0, length(TP1))

if(i > 0)
{
#assign the i jobs with the largest absolute TP1,TP2 difference to the line that processes it the quickest

Schd[order(diff)[length(diff):(length(diff)-(i-1))]]<-
((TP2<TP1)+1)[order(diff)[length(diff):(length(diff)-(i-1))]]
}

nonassign<-length(Schd[Schd==0])             #number of non assigned jobs

if(nonassign>0)
{

#if all jobs are unassigned, assign job with largest sum processing time to line that processes it the quickest

if(nonassign==length(Schd))
{
Schd[Schd==0][TP1[Schd==0]+TP2[Schd==0]==max(TP1[Schd==0]+TP2[Schd==0])][1]<-
((TP2<TP1)+1) [Schd==0][TP1[Schd==0]+TP2[Schd==0]==max(TP1[Schd==0]+TP2[Schd==0])][1]
nonassign<-nonassign-1
}

for(j in 1:nonassign)
{

Time1<-(Schd==1)%*%TP1          #processing time on line 1 for jobs currently assigned
```

```
Time2<- (Schd==2)%*%TP2          #processing time on line 2 for jobs currently assigned

Schd[Schd==0][TP1[Schd==0]+TP2[Schd==0]==max(TP1[Schd==0]+TP2[Schd==0])][1]<-
(Time2<Time1)+1             #of the subset of jobs not assigned, assign the job with the largest sum of
TP1 and TP2 to the line with the least processing time thus far; if more than one job fits criteria, assign
the first job (position 1)
}
}

Time1<-(Schd==1)%*%TP1          #processing time on line 1
Time2<- (Schd==2)%*%TP2         #processing time on line 2

Makespan[i+1]<-max(Time1,Time2)
Assignment[i+1,]<-Schd
}

RelativeMakespan<- Makespan/OptMakespan

InitialAssignSchedule<- data.frame(SampleID,Initial,Makespan,RelativeMakespan, Assignment)
InitialAssignSchedule
}
```

Best Initial Assign Function

```
#Calls initially assign function to return all possible initial assign solutions; then searches and returns
the best initial assign solution

BestInitialAssign<-function(SampleID,DataSample,OptMakespan)
{

All<-InitialAssign(SampleID,DataSample,OptMakespan)    #Return all possible initial assign solutions

Best<-All[All$RelativeMakespan==min(All$RelativeMakespan),][1,]   #Find solution with minimal
Relative Makespan; if more than one exists, return the first

Best
}
```

Run Best Initial Assign Function – 10 jobs, 1000 sample

```
num.samps<-1000                                  #Define number of samples to run
num.jobs<-10                                     # Define number of jobs in each sample

BestInitialAssignSchedule10<-
as.data.frame(array(dim=c(num.samps,(4+num.jobs)),dimnames=list(NULL,c("Sample
ID","Parameter","Makespan","Relative Makespan",paste("Job",(1:num.jobs),"Assign")))))

Cost<-as.data.frame(array(dim=c(num.samps,3),dimnames=list(NULL,c("CPU Time","Elapsed
Time","Max Memory"))))

for(i in 1:num.samps)
{
mem.tally.reset()
Cost[i,1:2]<-sys.time({
BestInitialAssignSchedule10[i,]<-
BestInitialAssign(i,DataSample10[DataSample10$SampID==i,],OptimalSchedule10[OptimalSchedule
10$Sample.ID==i,]$Makespan)
})
Cost[i,3]<- mem.tally.report()[2]
}

Method<-rep("Best Initial Assignment",dim(BestInitialAssignSchedule10)[1])

BestInitialAssignSchedule10<-data.frame(Method, Cost,BestInitialAssignSchedule10)
```

Run Best Initial Assign Function – 20 jobs, 1000 sample

```
num.samps<-1000                                  #Define number of samples to run
num.jobs<-20                                     # Define number of jobs in each sample

BestInitialAssignSchedule20<-
as.data.frame(array(dim=c(num.samps,(4+num.jobs)),dimnames=list(NULL,c("Sample
ID","Parameter","Makespan","Relative Makespan",paste("Job",(1:num.jobs),"Assign")))))

Cost<-as.data.frame(array(dim=c(num.samps,3),dimnames=list(NULL,c("CPU Time","Elapsed
Time","Max Memory"))))

for(i in 1:num.samps)
{
mem.tally.reset()
Cost[i,1:2]<-sys.time({
BestInitialAssignSchedule20[i,]<-
BestInitialAssign(i,DataSample20[DataSample20$SampID==i,],OptimalSchedule20[OptimalSchedule
20$Sample.ID==i,]$Makespan)
})
Cost[i,3]<- mem.tally.report()[2]
}

Method<-rep("Best Initial Assignment",dim(BestInitialAssignSchedule20)[1])

BestInitialAssignSchedule20<-data.frame(Method, Cost,BestInitialAssignSchedule20)
```

Initial Assign (Φ = 1) Function

```
#initially assign 1 job to the line that processes it the fastest; for the remaining jobs, assign using LPT
sum rule

InitialAssign<-function(SampleID,DataSample,OptMakespan)
{

DataSample<-ProcessTime(DataSample)
TP1<-DataSample$TP1
TP2<-DataSample$TP2

diff<-abs(TP1-TP2)
Assignment<-array(dim=c(1,length(TP1)))

# create schedule that assigns the job with the largest difference between TP1 and TP2 to the line that
processes the job the quickest, the remaining jobs are assigned by LPT on Sum of TP1 & TP2 rule

Schd<-rep(0, length(TP1))

#assign the 1 job with the largest absolute TP1,TP2 difference to the line that processes it the quickest

Schd[order(diff)[length(diff)]]<-((TP2<TP1)+1)[order(diff)[length(diff)]]

nonassign<-length(Schd[Schd==0])            #number of non assigned jobs

for(j in 1:nonassign)
{

Time1<-(Schd==1)%*%TP1          #processing time on line 1 for jobs currently assigned
Time2<- (Schd==2)%*%TP2         #processing time on line 2 for jobs currently assigned

Schd[Schd==0][TP1[Schd==0]+TP2[Schd==0]==max(TP1[Schd==0]+TP2[Schd==0])][1]<-
(Time2<Time1)+1           #of the subset of jobs not assigned, assign the job with the largest sum of
TP1 and TP2 to the line with the least processing time thus far; if more than one job fits criteria, assign
the first job (position 1)
}

Time1<-(Schd==1)%*%TP1          #processing time on line 1
Time2<- (Schd==2)%*%TP2         #processing time on line 2

Makespan<-max(Time1,Time2)
Assignment[1,]<-Schd


RelativeMakespan<- Makespan/OptMakespan

InitialAssignSchedule<- data.frame(SampleID,1,Makespan,RelativeMakespan,Assignment)
InitialAssignSchedule
}
```

Run Initial Assign (Φ = 1) Function – 10 jobs, 1000 sample

```
num.samps<-1000                    #Define number of samples to run
num.jobs<-10                       # Define number of jobs in each sample

InitialAssign1Schedule10<-
as.data.frame(array(dim=c(num.samps,(4+num.jobs)),dimnames=list(NULL,c("Sample
ID","Parameter","Makespan","Relative Makespan",paste("Job",(1:num.jobs),"Assign")))))

Cost<-as.data.frame(array(dim=c(num.samps,3),dimnames=list(NULL,c("CPU Time","Elapsed
Time","Max Memory"))))

for(i in 1:num.samps)
{
mem.tally.reset()
Cost[i,1:2]<-sys.time({
InitialAssign1Schedule10[i,]<-
InitialAssign(i,DataSample10[DataSample10$SampID==i,],OptimalSchedule10[OptimalSchedule10$
Sample.ID==i,]$Makespan)
})
Cost[i,3]<- mem.tally.report()[2]
}

Method<-rep("Initial Assignment = 1",dim(InitialAssign1Schedule10)[1])

InitialAssign1Schedule10<-data.frame(Method,Cost,InitialAssign1Schedule10)
```

Initial Assign ($\Phi = 3$) Function

```
#initially assign 3 jobs to the line that processes them the fastest; for the remaining jobs, assign using
LPT sum rule

InitialAssign<-function(SampleID,DataSample,OptMakespan)
{

DataSample<-ProcessTime(DataSample)
TP1<-DataSample$TP1
TP2<-DataSample$TP2

diff<-abs(TP1-TP2)
Assignment<-array(dim=c(1,length(TP1)))

# create schedule that assigns the 3 jobs with the largest difference between TP1 and TP2 to the line
that processes the job the quickest, the remaining jobs are assigned by LPT on Sum of TP1 & TP2 rule

Schd<-rep(0, length(TP1))

#assign the 3 jobs with the largest absolute TP1,TP2 difference to the line that processes it the quickest

Schd[order(diff)[length(diff):(length(diff)-2)]]<-((TP2<TP1)+1)[order(diff)[length(diff):(length(diff)-
2)]]

nonassign<-length(Schd[Schd==0])          #number of non assigned jobs

for(j in 1:nonassign)
{

Time1<-(Schd==1)%*%TP1          #processing time on line 1 for jobs currently assigned
Time2<- (Schd==2)%*%TP2         #processing time on line 2 for jobs currently assigned

Schd[Schd==0][TP1[Schd==0]+TP2[Schd==0]==max(TP1[Schd==0]+TP2[Schd==0])][1]<-
(Time2<Time1)+1           #of the subset of jobs not assigned, assign the job with the largest sum of
TP1 and TP2 to the line with the least processing time thus far; if more than one job fits criteria, assign
the first job (position 1)
}

Time1<-(Schd==1)%*%TP1          #processing time on line 1
Time2<- (Schd==2)%*%TP2         #processing time on line 2

Makespan<-max(Time1,Time2)
Assignment[1,]<-Schd


RelativeMakespan<- Makespan/OptMakespan

InitialAssignSchedule<- data.frame(SampleID,1,Makespan,RelativeMakespan,Assignment)
InitialAssignSchedule
}
```

Run Initial Assign (Φ = 3) Function – 20 jobs, 1000 sample

```
num.samps<-1000                          #Define number of samples to run
num.jobs<-20                             # Define number of jobs in each sample

InitialAssign3Schedule20<-
as.data.frame(array(dim=c(num.samps,(4+num.jobs)),dimnames=list(NULL,c("Sample
ID","Parameter","Makespan","Relative Makespan",paste("Job",(1:num.jobs),"Assign")))))

Cost<-as.data.frame(array(dim=c(num.samps,3),dimnames=list(NULL,c("CPU Time","Elapsed
Time","Max Memory"))))

for(i in 1:num.samps)
{
mem.tally.reset()
Cost[i,1:2]<-sys.time({
InitialAssign3Schedule20[i,]<-
InitialAssign(i,DataSample20[DataSample20$SampID==i,],OptimalSchedule20[OptimalSchedule20$
Sample.ID==i,]$Makespan)
})
Cost[i,3]<- mem.tally.report()[2]
}

Method<-rep("Initial Assignment = 3",dim(InitialAssign3Schedule20)[1])

InitialAssign3Schedule20<-data.frame(Method,Cost,InitialAssign3Schedule20)
```

Ibarra-Kim Algorithm F Function

```
IbarraKimF<-function(SampleID,DataSample,OptMakespan)
{
DataSample<-ProcessTime(DataSample)
TP1<-DataSample$TP1
TP2<-DataSample$TP2

Assignment<-array(dim=c(1,length(TP1)))

# create schedule that assigns each job to the line that processes the job the quickest, if equal assign to
line 1
Schd<-rep(1, length(TP1))
Schd[TP1- TP2>0]<-2

Time1<-(Schd==1)%*%TP1          #processing time on line 1 for jobs currently assigned
Time2<- (Schd==2)%*%TP2         #processing time on line 2 for jobs currently assigned

Makespan<-max(Time1,Time2)      #determine current Makespan

if(Time1!=Time2)          #if Time1 = Time2 no need to proceed, optimal schedule obtained, else…
{
LongLine<-(Time2>Time1)+1       #determine which line has the longer processing time
Ind<-(Schd==LongLine)           #Indication of jobs originally assigned to the longer line

TP<-cbind(TP1,TP2)        #develop matrix of job processing times on line 1 and line 2
Rel<-(TP[,LongLine]/TP[,(-(LongLine-1)+2)])[Ind]   #calculate I-K ratio for jobs on long line

#Create loop to consider all jobs on long line in decreasing order of I-K ratio; reassign job to other line
if the reassignment decreases the makespan

for (i in length(Schd[Schd==LongLine]):1)   #loop in decreasing order of I-K ratio
{
AltSchd<-Schd                                     #create temporary new schedule
AltSchd[Ind][order(Rel)[i]]<-(-(LongLine-1)+2)    #assign job considered to other line
AltTime1<-(AltSchd==1)%*%TP1            #calculate resulting line 1 processing time
AltTime2<- (AltSchd==2)%*%TP2                     # calculate resulting line 2 processing time
AltMakespan<-max(AltTime1,AltTime2)     #calculate resulting makespan
if(AltMakespan<Makespan)                          #if resulting makespan is less than original
{                                                 # makespan then make resulting schedule and
Schd<-AltSchd                                     #resulting makespan permanent
Makespan<-AltMakespan
}
}
}
Assignment[1,]<-Schd
RelativeMakespan<- Makespan/OptMakespan

I.K.Schedule<-data.frame(SampleID,0,Makespan,RelativeMakespan, Assignment)
I.K.Schedule
}
```

Run Ibarra Kim Algorithm F Function – 10 jobs, 1000 sample

```
num.samps<-1000                              #Define number of samples to run
num.jobs<-10                                 # Define number of jobs in each sample

IKSchedule10<-
as.data.frame(array(dim=c(num.samps,(4+num.jobs)),dimnames=list(NULL,c("Sample
ID","Parameter","Makespan","Relative Makespan",paste("Job",(1:num.jobs),"Assign")))))

Cost<-as.data.frame(array(dim=c(num.samps,3),dimnames=list(NULL,c("CPU Time","Elapsed
Time","Max Memory"))))

for(i in 1:num.samps)
{
mem.tally.reset()
Cost[i,1:2]<-sys.time({
IKSchedule10[i,]<-
IbarraKimF(i,DataSample10[DataSample10$SampID==i,],OptimalSchedule10[OptimalSchedule10$S
ample.ID==i,]$Makespan)
})
Cost[i,3]<- mem.tally.report()[2]
}

Method<-rep("Ibarra-Kim",dim(IKSchedule10)[1])

IKSchedule10<-data.frame(Method,Cost,IKSchedule10)
```

Run Ibarra Kim Algorithm F Function – 20 jobs, 1000 sample

```
num.samps<-1000                              #Define number of samples to run
num.jobs<-20                                 # Define number of jobs in each sample

IKSchedule20<-
as.data.frame(array(dim=c(num.samps,(4+num.jobs)),dimnames=list(NULL,c("Sample
ID","Parameter","Makespan","Relative Makespan",paste("Job",(1:num.jobs),"Assign")))))

Cost<-as.data.frame(array(dim=c(num.samps,3),dimnames=list(NULL,c("CPU Time","Elapsed
Time","Max Memory"))))

for(i in 1:num.samps)
{
mem.tally.reset()
Cost[i,1:2]<-sys.time({
IKSchedule20[i,]<-
IbarraKimF(i,DataSample20[DataSample20$SampID==i,],OptimalSchedule20[OptimalSchedule20$S
ample.ID==i,]$Makespan)
})
Cost[i,3]<- mem.tally.report()[2]
}

Method<-rep("Ibarra-Kim",dim(IKSchedule20)[1])

IKSchedule20<-data.frame(Method,Cost,IKSchedule20)
```

Large *k* Function

```
LargeK<-function(SampleID,k,TP1,TP2,OptMakespan)
{

Assignment<-array(dim=c(1,length(k)))

# create schedule by assigning jobs based on number of boards within job:
Schd<-rep(0, length(k))

# 1.) assign job with most boards to line 2 as line 2 is most likely to complete the quickest:
Schd[k==max(k)]<-2

# 2.) assign job with second most boards to line 1
Schd[Schd==0][k[Schd==0]==max(k[Schd==0])]<-1

# 3.) assign remaining jobs in decreasing order of k to line with smallest n
nonassign<-length(Schd[Schd==0])          #number of non assigned jobs

for(j in 1:nonassign)
{

N1<-sum(k[Schd==1])              #number of boards scheduled on line 1 for jobs currently assigned
N2<- sum(k[Schd==2])             # number of boards scheduled on line 2 for jobs currently assigned

Schd[Schd==0][k[Schd==0]==max(k[Schd==0])][1]<-(N2<N1)+1              #of the subset of jobs
not assigned, assign the job with the maximum number of boards to the line with the least number of
boards thus far, if more than one meet the criteria, assign the first (position 1)
}

Time1<-(Schd==1)%*%TP1         #processing time on line 1
Time2<- (Schd==2)%*%TP2        #processing time on line 2

Makespan<-max(Time1,Time2)

Assignment[1,]<-Schd
RelativeMakespan<- Makespan/OptMakespan

K.Schedule<-data.frame(SampleID,0,Makespan,RelativeMakespan, Assignment)
K.Schedule
}
```

Run Large *k* Function – 10 jobs, 1000 sample

```
## TP1 and TP2 are needed to evaluate performance (makespan), but are not included in cost of
algorithm

DataSample<-ProcessTime(DataSample10)
TP1<-DataSample$TP1
TP2<-DataSample$TP2

num.samps<-1000                          #Define number of samples to run
num.jobs<-10                             # Define number of jobs in each sample

LargeKSchedule10<-
as.data.frame(array(dim=c(num.samps,(4+num.jobs)),dimnames=list(NULL,c("Sample
ID","Parameter","Makespan","Relative Makespan",paste("Job",(1:num.jobs),"Assign")))))

Cost<-as.data.frame(array(dim=c(num.samps,3),dimnames=list(NULL,c("CPU Time","Elapsed
Time","Max Memory"))))

for(i in 1:num.samps)
{
mem.tally.reset()
Cost[i,1:2]<-sys.time({
LargeKSchedule10[i,]<- LargeK(i,
DataSample10[DataSample10$SampID==i,]$k,TP1[DataSample10$SampID==i],TP2[DataSample10$
SampID==i],OptimalSchedule10[OptimalSchedule10$Sample.ID==i,]$Makespan)
})
Cost[i,3]<- mem.tally.report()[2]
}

Method<-rep("Large k",dim(LargeKSchedule10)[1])

LargeKSchedule10<-data.frame(Method,Cost,LargeKSchedule10)
```

Run Large *k* Function – 20 jobs, 1000 sample

```
## TP1 and TP2 are needed to evaluate performance (makespan), but are not included in cost of
algorithm

DataSample<-ProcessTime(DataSample20)
TP1<-DataSample$TP1
TP2<-DataSample$TP2

num.samps<-1000                          #Define number of samples to run
num.jobs<-20                             # Define number of jobs in each sample

LargeKSchedule20<-
as.data.frame(array(dim=c(num.samps,(4+num.jobs)),dimnames=list(NULL,c("Sample
ID","Parameter","Makespan","Relative Makespan",paste("Job",(1:num.jobs),"Assign")))))

Cost<-as.data.frame(array(dim=c(num.samps,3),dimnames=list(NULL,c("CPU Time","Elapsed
Time","Max Memory"))))
```

```
for(i in 1:num.samps)
{
mem.tally.reset()
Cost[i,1:2]<-sys.time({
LargeKSchedule20[i,]<- LargeN(i,
DataSample20[DataSample20$SampID==i,]$k,TP1[DataSample20$SampID==i],TP2[DataSample20$
SampID==i],OptimalSchedule20[OptimalSchedule20$Sample.ID==i,]$Makespan)
})
Cost[i,3]<- mem.tally.report()[2]
}

Method<-rep("Large k",dim(LargeKSchedule20)[1])

LargeKSchedule20<-data.frame(Method, Cost,LargeKSchedule20)
```

References

1 Graham R.L., Lawler E.L., Lenstra J.K. and Rinnooy Kan A.H.G., “Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey”, Annals of Discrete Mathematics 5, 1979, pp. 287–326.

2 Karp R.M., “Reducibility among combinatorial problems”, Complexity of Computer Computations (Edited by R.E. Miller and J.W. Thatcher) Plenum Press, New York, 1972, pp. 85-103.

3 Alexanderson G., "Euler and Königsberg's Bridges: A Historical View", Bulletin of the American Mathematical Society, July 2006.

4 Pinedo M., *Scheduling: Theory, Algorithms, and Systems*, Prentice-Hall Inc. Upper Saddle River, NJ, 2002.

5 Garey M.R. and Johnson D.S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, New York, 1979.

6 Lenstra J.K., Shmoys D.B., and Tardos E., “Approximation Algorithms for Scheduling Unrelated Parallel Machines”, Mathematical Programming 46, 1990, pp. 259–271.

7 Ghirardi M. and Potts C.N., “Makespan Minimization for Scheduling Unrelated Parallel Machines: A Recovering Beam Search Approach”, European Journal of Operational Research 165, 2005, pp. 457-467.

8 Martello S., Soumis F. and Toth P., “Exact Approximation Algorithm for Makespan Minimization on Unrelated Parallel Processors”, Discrete Applied Mathematics 75, 1997, pp. 169–188.

9 Stern H.I., “Minimizing Makespan for Independent Jobs on Nonidentical Parallel Processor – An Optimal Procedure”, Working Paper, Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, Beer-Sheva 1976.

10 Mokotoff E. and Jimeno J.L., “Heuristics Based on Partial Enumeration for the Unrelated Parallel Processor Scheduling Problem”, Annals of Operations Research 117, 2002, pp. 133–150.

11 Davis E. and Jaffe J.M., “Algorithms for Scheduling Tasks on Unrelated Parallel Processors”, Journal of the ACM 28, 1981, pp. 721–736.

12 Ibarra O.H. and Kim C.E., “Heuristic Algorithms for Scheduling Independent Tasks on Non-identical Processors”, Journal of the ACM 24, 1977, pp. 280–289.

13 Becker R.A. and Chambers J.M., *S: An Interactive Environment for Data Analysis and Graphics*, Wadsworth & Brooks/Cole, Pacific Grove, CA, 1984.

14 Hoaglin D.C., Mosteller F., and Tukey J.W., *Understanding Robust and Exploratory Data Analysis*, John Wiley & Sons, Inc., New York, 1983.

15 Hastie R. and Dawes R.M., *Rational Choice in an Uncertain World*, Sage, Thousand Oaks, CA, 2001.

16 Stirling W.C., *Satisficing Games and Decision Making*, Cambridge University Press, Cambridge, UK, 2003.