Winter 2010

Optimization Models and Algorithms for Spatial Scheduling

Christopher J. Garcia
Old Dominion University

OPTIMIZATION MODELS AND ALGORITHMS FOR SPATIAL SCHEDULING

by

Christopher J. Garcia
M.S. August 2008, Florida Institute of Technology
M.S. September 2004, Nova Southeastern University
B.S. May 2001, Old Dominion University

A Dissertation Submitted to the Faculty of

Old Dominion University in Partial Fulfillment of the

Requirement for the Degree of

DOCTOR OF PHILOSOPHY

ENGINEERING MANAGEMENT

OLD DOMINION UNIVERSITY
December 2010

Approved by:

Ghaith Rabadi (Director)

Shannon Bowling (Member)

Holly Handley (Member)

Steve Cotter (Member)

ABSTRACT

OPTIMIZATION MODELS AND ALGORITHMS FOR SPATIAL SCHEDULING

Christopher J. Garcia
Old Dominion University, 2010
Director: Dr. Ghaith Rabadi

Spatial scheduling problems involve scheduling a set of activities or jobs that each require a certain amount of physical space in order to be carried out. In these problems space is a limited resource, and the job locations, orientations, and start times must be simultaneously determined. As a result, spatial scheduling problems are a particularly difficult class of scheduling problems. These problems are commonly encountered in diverse industries including shipbuilding, aircraft assembly, and supply chain management. Despite its importance, there is a relatively scarce amount of research in the area of spatial scheduling.

In this dissertation, spatial scheduling problems are studied from a mathematical and algorithmic perspective. Optimization models based on integer programming are developed for several classes of spatial scheduling problems. While the majority of these models address problems having an objective of minimizing total tardiness, the models are shown to contain a core set of constraints that are common to most spatial scheduling problems. As a result, these constraints form the basis of the models given in this dissertation and many other spatial scheduling problems with different objectives as well. The complexity of these models is shown to be at least NP-complete, and spatial scheduling problems in general are shown to be NP-hard. A lower bound for the total

tardiness objective is shown, and a polynomial-time algorithm for computing this lower bound is given.

The computational complexity inherent to spatial scheduling generally prevents the use of optimization models to find solutions to larger, realistic problems in a reasonable time. Accordingly, two classes of approximation algorithms were developed: greedy heuristics for finding fast, feasible solutions; and hybrid meta-heuristic algorithms to search for near-optimal solutions. A flexible hybrid algorithm framework was developed, and a number of hybrid algorithms were devised from this framework that employ local search and several varieties of simulated annealing. Extensive computational experiments showed these hybrid meta-heuristic algorithms to be effective in finding high-quality solutions over a wide variety of problems. Hybrid algorithms based on local search generally provided both the best-quality solutions and the greatest consistency.

This dissertation is dedicated to my wife Kristin, whose love and support I cherish.

An excellent wife who can find? She is far more precious than jewels. The heart of her husband trusts in her, and he will have no lack of gain. She does him good, and not harm, all the days of her life.

Proverbs 31:10-12

ACKNOWLEDGMENTS

The writing of this dissertation has been a challenging undertaking and would not have been possible without the guidance and support of many people. First I would like to thank my mentor Dr. Ghaith Rabadi for helping me develop my abilities as a researcher and scholar. He has shown me by example what it means to do good research, and his high standards of scholarship have brought out my very best and enabled me to accomplish far more than I thought I was capable of. He has been instrumental in the writing of this dissertation, from introducing me to the topic of spatial scheduling to suggesting the use of meta-heuristics for approximation. These suggestions have fundamentally shaped this dissertation. Later in the research he strongly urged me to find a lower bound for the problem, which led to the most important theoretical result in this dissertation and greatly enhanced its rigor and quality. I am also most grateful to the members of my committee, who have taken the time to work with me and have provided many suggestions that have significantly improved this research. I would like to thank Dr. Shannon Bowling for his suggestion to consider alternative approaches, and not to overlook approaches that seemed simpler. This suggestion led directly to the best-performing algorithm developed in this research. I would also like to thank Dr. Steve Cotter for his suggestion to consider cases where job size is correlated with processing time. This led to the development of a specialized problem-generation algorithm and a whole new class of test problems that played an important role in validating the optimization algorithms developed.

I am also grateful for the wonderful friends and family who have given me support while I undertook this dissertation. I would first like to thank my wife Kristin for all of her love, support, and patience, as well as my daughters Keeli and Olivia. I would also like to thank Bob Willetts for his friendship and encouragement throughout the ups and downs of my doctoral studies. Finally, I would like to thank my parents Jaime and Patricia Garcia for all of their love and support over the years and for encouraging me to work hard and aim high.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION TO SPATIAL SCHEDULING

Scheduling is a common and often critical activity of many business areas and industries, including manufacturing, assembly, services, and supply chain management to name a few. Scheduling problems generally involve assigning tasks or activities to a set of resources, most often in a manner that optimizes some specified objective or performance measure. Beyond this, different types of scheduling problems present their own unique sets of objectives and constraints, and require individualized formulation and solution methods.

Scheduling problems can become very difficult to solve when limitations exist on the resources available to enable the carrying out of tasks or activities [24]. At the heart of many important assembly, manufacturing, and logistics problems is a relatively unstudied type of optimization problem: a *Spatial Scheduling* problem. A spatial scheduling problem, like other types of scheduling problems, involves a set of jobs where each job has a due date, processing time, and earliest start time. However, each job also requires a certain amount of physical space for processing. Furthermore, there is a limited amount of physical space available in which to process these jobs. The objective of such a problem generally involves best meeting the job due dates within the given processing space. Thus, solving a spatial scheduling problem involves the challenging task of simultaneously determining the start times for each job *as well as* their spatial locations and layouts inside the processing area.

There is very little existing research addressing this type of problem. Furthermore, the existing research is very industry-specific and relies heavily on heuristics based on

problem-specific domain knowledge, rather than a sound body of theory, algorithms, and general approaches for spatial scheduling. In light of this gap in research, the aim of this dissertation is to address the topic of spatial scheduling in a more systematic manner, and to develop more general exact and approximate methods that can be applied to a broader range of spatial scheduling problems. Progress in this research area will advance the research in various areas of applications, especially in industries that deal with scheduling large components over a limited space for manufacturing, assembly and maintenance of products such as ships, aircraft, space vehicles, cranes, cargo handlers, excavators, loaders, and mining trucks to name some. In fact, the research contribution can be utilized to include warehousing and supply chain problems as well. Traditional production scheduling algorithms, and layout/packing optimization methods separately are not appropriate for the problem addressed in this research due to the need to solve the spatial problem and temporal problem simultaneously (i.e., optimizing space dynamically over time).

1.1. AN EXAMPLE PROBLEM

To illustrate a basic spatial scheduling problem example, imagine a single rectangular area is given having a width of 10 and height of 8. Suppose the jobs in Table 1 below must all be processed inside this area, and the objective is to minimize the total amount of tardiness for all jobs. The Tardiness for job j is defined as $T_j = \max(0, C_j - d_j)$ where C_j is the job's completion time and d_j is its due date. The objective will then be $\sum_{j=1}^{n} T_j$ where n is the total number of jobs.

Job	Width	Height	Earliest Start	Processing Time	Due date d_j
1	4	5	2	4	8
2	2	4	0	5	6
3	1	3	2	6	10
4	3	1	0	2	9
5	3	10	4	4	8

Table 1: A small problem instance

Like any other scheduling problem, solving this problem requires determining the time each job must start and end. Unlike other types of scheduling problems, however, a solution also requires determining the *location* inside the area that each job will occupy. This involves assigning a coordinate to each job and, in some cases, may also involve rotating jobs so they can fit inside the processing area or best optimize the objective function. An optimal solution to this problem is given in Table 2 below, and is visualized in Figure 1. In this solution, each job is assigned a start time *and* location. Job 5 was also rotated by 90 degrees so it can fit inside the area. In Figure 1, the bottom-left corner of the processing area is the (0, 0) coordinate, and the coordinate of each job is the corner nearest to the (0, 0) point.

Job j	Start	Completion Time C_j	Due Date d_j	Tardiness T_j	X	Y	Width	Height
2	0	5	6	0	0	0	2	4
1	2	6	8	0	2	0	4	5
3	2	8	10	0	9	0	1	3
4	2	4	9	0	6	0	3	1
5	4	8	8	0	0	6	10	3

Table 2: An optimal solution to the small problem instance, in ascending start-time order

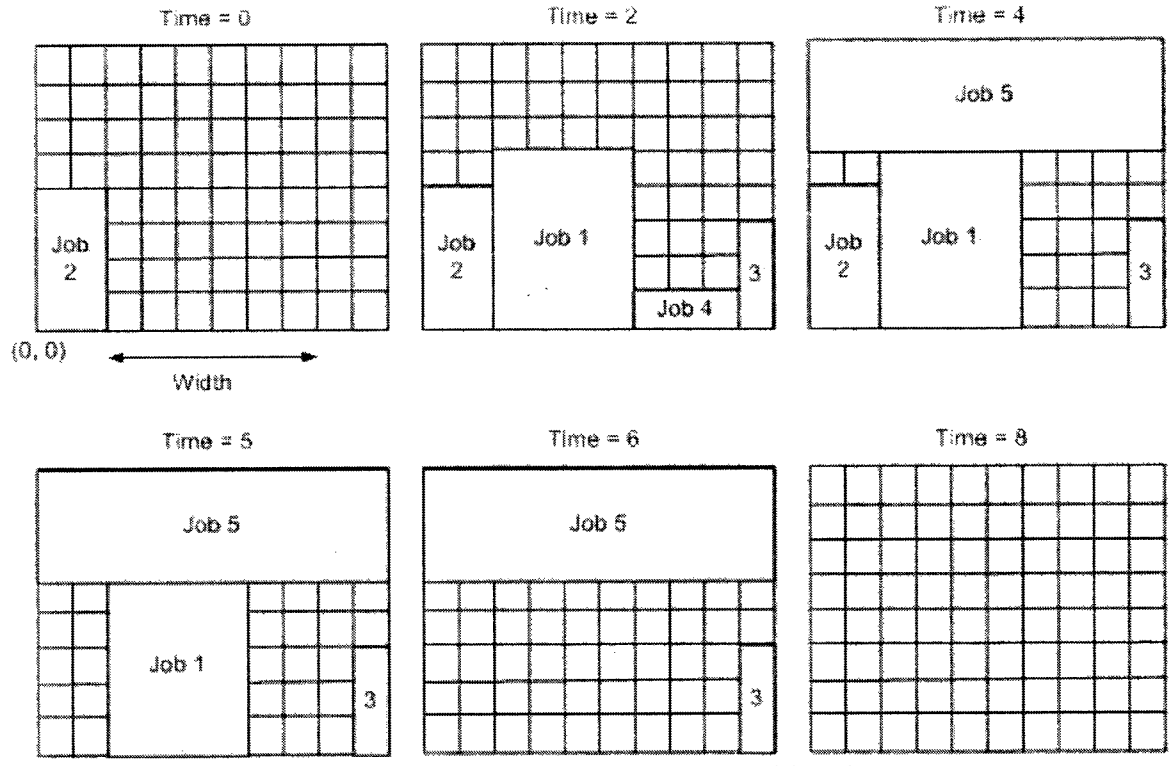

Figure 1: A visualized solution to the small problem instance

This example illustrates that spatial scheduling problems have both temporal and geometric aspects to them, and also that these two aspects are intertwined in a complex

way. Thus, *when* the jobs are scheduled affects *where* they can be placed and *how* they can be laid out, and vice versa. It is intuitively apparent that this makes spatial scheduling problems more complex than other types of scheduling problems. As a consequence, the unique combination of temporal and spatial aspects found in spatial scheduling problems makes them very difficult to solve in general.

CHAPTER 2: DISSERTATION SCOPE

In real-life situations it is possible, and perhaps even to be expected, that a spatial scheduling problem encountered in one context will differ from another in one or more aspects. Such differences may include certain constraints as well as the objective function. However, many kinds of spatial scheduling problems can be articulated in terms of one of two general problem models, referred to in this dissertation as the *single-area problem,* and the *multiple-area problem.* This dissertation research will address spatial scheduling problems having these forms. The general forms of these problems are stated in Table 3 below.

The Single-Area Problem *Given*: 1) a single processing area with a specified width and length, 2) n jobs each having a width, height, earliest start time, processing time, and due date, and 3) an objective function f *Objective:* Assign to each job 1) a start time, and 2) a location inside the processing area, so as to optimize f.
The Multiple-Area Problem *Given*: 1) m processing areas each having a specified width and height, 2) n jobs each having a width, height, earliest start time, processing time, and due date, and 3) an objective function f *Objective:* Assign to each job 1) a single processing area in which to be processed, 2) a start time, and 3) location inside the assigned processing area, so as to optimize f.

Table 3: Problem statements for the single-area and multiple-area problems

Because of the inherent limitation of physical space, certain constraints will apply to virtually all spatial scheduling problems. It is thus expected that individual problem instances will differ most from one another by their objective function f. Several types of objectives are inherent to spatial scheduling. These include due date-related objectives (e.g., minimizing total tardiness, maximum lateness, and total earliness and tardiness), throughput-related objectives (e.g. minimizing makespan, total completion time), and load-balancing objectives (e.g. minimizing the difference between number of jobs assigned to different areas). There may be other types as well that are more specific to certain problems. Thus, a spatial scheduling problem can involve any combination of different types of objectives. In order to accommodate multiple objectives while preserving the general problem structure, the objective function f can in many cases be treated as a linear combination of one or more weighted terms, where each term quantifies an individual objective.

The nature of the basic problem structures indicates that much can be learned about spatial scheduling problems in general by studying problems with a particular objective function. This is because certain constraints and other structural properties apply to all spatial scheduling problems, such as the requirement that at any given time no two jobs may collide and occupy the same space. As a result, this dissertation will primarily focus on a single objective: minimizing the total tardy time. As will be shown, however, by studying problems with this objective there is much that can be said about spatial scheduling problems in general. Consequently, most of the models and solution methods developed in this dissertation can be readily adapted for problems of different objective functions. Other objective functions will occasionally be touched upon, and it

will always be pointed out to the reader whenever a model, solution method, or theoretical result may be applied more broadly to other spatial scheduling problems.

In the next chapter, the relevant literature will be reviewed and it will be shown that spatial scheduling problems have been addressed primarily through methods relying heavily on industry domain and problem-specific knowledge as opposed to a general body of theory, models, and algorithms for such problems. In light of this gap in the literature, this research will seek to address spatial scheduling in a more general and systematic manner by addressing the single- and multiple-area problems described above, primarily focusing on the minimal total tardiness objective. The scope of this dissertation is based on the following research objectives: 1) development of exact and approximate methods for solving problems with the total tardiness objective, that can also be extended and applied to a broader and more general range of spatial scheduling problems, 2) an analysis of the computational complexity associated with spatial scheduling, 3) the derivation of a lower bound for problems having the total tardiness objective, and 4) extensive computational experimentation and analysis of solutions obtained through the developed methods.

In order to meet these objectives, this dissertation is organized as follows. In Chapter 3, a thorough literature review is conducted to review the state of the art not just of spatial scheduling literature, but also of other relevant types of scheduling and packing problems. In particular it will be seen that there is little systematic treatment of spatial scheduling, and of the scarce existing research in this area most is based on expert systems and is highly domain knowledge-dependent. In Chapter 4, optimization models

are developed for several classes of single- and multiple-area spatial scheduling problems. These models will enable optimal solutions to be found for small problem instances and also provide a significant amount of insight into the general nature of spatial scheduling problems. The models will include core constraints that apply to virtually all spatial scheduling problems, enabling them to be readily adapted to many individual problems and extended by other researchers. Additionally, such models will provide a basis for complexity analysis of solution by branch-and-bound, the most commonly implemented method for solving integer programs [27]. In Chapter 5, the computational complexity of spatial scheduling problems is addressed. The complexity of the optimization models developed in Chapter 4 is also addressed. In Chapter 6, a lower bound is derived for spatial scheduling problems involving the total tardiness objective. Additionally, a polynomial-time algorithm is given that will enable the calculation of this lower bound.

Chapters 7 and 8 deal with the development of approximation algorithms capable of finding near-optimal solutions in a reasonable time for larger, realistic problems. In Chapter 7 a number of greedy heuristic algorithms are developed for spatial scheduling. These heuristics are designed primarily to give feasible solutions quickly and involve no search. These heuristic algorithms also serve as important components of the hybrid meta-heuristic algorithms developed in Chapter 8. Chapter 8 introduces a general spatial scheduling algorithm framework that combines a greedy heuristic algorithm for finding feasible schedules with a meta-heuristic for searching for near-optimal solutions. Using this framework, several hybrid meta-heuristic algorithms are developed that employ local search as well as several varieties of simulated annealing.

In Chapter 9, extensive computational experiments are conducted on the hybrid meta-heuristic algorithms to assess their performance. Four different problem-generation algorithms were developed and used (detailed in Appendix 1) to generate large numbers of problems of differing sizes and qualitative characteristics. The hybrid meta-heuristic algorithms are tested on a wide variety of generated benchmark problems. The performance of these algorithms is assessed in comparison to the calculated lower bound, to optimal solutions found using the models (in small problem instances), and to each other. The hybrid meta-heuristic algorithms are found to be generally effective in obtaining good solutions, and the best-performing algorithm variants are identified. Finally, Chapter 10 summarizes the findings of this dissertation and highlights several future research areas.

CHAPTER 3: LITERATURE REVIEW

As has been discussed in the preceding chapters, spatial scheduling superimposes both temporal and geometric aspects into a single problem. Because of this dichotomy, a sizeable portion of research that addresses bin-packing or strip-packing problems is highly relevant to spatial scheduling. Literature related to spatial scheduling may thus be grouped into two broad categories: literature that directly addresses problems of spatial scheduling, and relevant literature that addresses problems related to bin, box, and tile-packing. Spatial scheduling appears to be a relatively new area of research; there are few papers published directly on the subject. However, there is a significant amount of literature addresses problems related to packing problems.

3.1. EXISTING SPATIAL SCHEDULING LITERATURE

Most of the research that directly addresses spatial scheduling deals with problems in the shipbuilding industry. One such problem, the block assembly scheduling problem, is addressed by several papers in the literature. This problem is succinctly described by Park et al. [13]. The hull of a ship is constructed from a set of sub-parts called *blocks*. Blocks are assembled inside of *bays*. Each block has an earliest- and latest-start-time, a set of acceptable bays in which it can be assembled, and a shape. The block's shape is specified by a convex polygon. The objective is to schedule the blocks to be assembled in bays and allocate their spatial layout in such a way that minimizes the maximum tardiness of any block assembly.

The approach in [13] relaxes the latest-start-time constraint in order to ensure feasibility. A decomposition and heuristic approach is used that involves breaking the

planning horizon into periods. A schedule is generated for each period and then these schedules are concatenated together to form the final schedule. In order to reduce the computational burden a predetermined set of block shapes was used based on the shipbuilder's practice. The algorithm for generating schedules makes use of a search tree that expands partial schedules until complete schedules have been generated or termination conditions occur. At each step, a set of possible schedules are generated by expanding the current partial schedule. These are evaluated, and the most promising ones are chosen as child nodes of the search tree. This process is repeated until a suitable schedule is found. The assignment of blocks to bays is a sub-algorithm of the overall scheduling process. It is essentially a spatial load-balancing procedure that assigns blocks to bays based on an over-estimated rectangularization of the blocks and the space availability of the bays. The spatial layout of blocks within bays is another sub-algorithm of the scheduling process. For a given bay at any time in the schedule, there are a set of blocks within that bay. A scanning algorithm traverses the bay and looks for an open space in which the next block can fit, and then allocates the block to that space.

The block assembly scheduling problem is also addressed Lee et al. [12]. In this paper the basis for an expert system to solve the block scheduling problem was detailed. The problem of placing two-dimensional convex polygons into fixed rectangular areas is explored. Specifically, four placement strategies are examined. The *Maximal Remnant Space Utilization Strategy* is a vertex-based strategy useful when the problem involves many trapezoidal or triangular blocks. The *Maximal Free Rectangular Space Strategy* is based on 90-degree corner-spaced areas and is useful when allocating rectangular-shaped objects. The *Initial Positioning Strategy* is useful on empty areas when positioning the

first object inside. The *Edging Strategy* is an edge-based strategy that aims to minimize the fractioning of space by properly positioning along edges. These four strategies are utilized by a composite partitioning algorithm that places objects based on its characteristics and that of the space. The scheduling of objects in two dimensions is then shown to be a type of three-dimensional packing problem, where the third dimension is time. This is not a general three-dimensional packing problem, however, because of start-time and end-time constraints. This three-dimensional representation is a kind of schedule search space, and schedules are generated in this space. Six types of backtracking operations are used to search for feasible solutions. These backtracking operations either select new bays or adjust the positioning of the blocks either spatially, temporally, or both.

The two papers mentioned thus far utilize a combination of searching and knowledge-based approaches to develop schedules for the block assembly scheduling problem. In Varghese et al.[30] a genetic algorithm is used to solve a related problem. In this problem, the dates of erection of each block are determined ahead of time, and the objective involves determining the appropriate locations for the blocks near appropriate machinery. The approach employed involves using a penalty function-based genetic algorithm.

A related problem to the block assembly in shipbuilding is the block painting problem. This problem is discussed by Cho et al. in [14]. Block painting consists of two phases: blasting and painting. These phases are carried out in sequence – every block is first blasted and then painted. The blasting and painting take place in different *cells*

within the paint shop, thus blocks go through the process in batches. The objective of this problem is to maximize space utilization while also balancing the workload of the teams. The approach developed in [14] entails the use of four separate algorithms that work in separate steps: operation strategy, block scheduling, block arrangement, block assignment. The operation strategy is applied first and determines which cells to use for blasting and which ones to use for painting. The block scheduling and block arrangement algorithms are then applied in tandem. The block scheduling algorithm determines the block operation schedules and the block arrangement algorithm geometrically positions the blocks within the cells. Finally, the block assignment algorithm assigns work teams to blocks.

Another type of problem that is very similar to spatial scheduling is called the *berth allocation problem*. This problem is encountered in maritime cargo terminals. Each terminal has cargo ships coming and going, loading and unloading, and the objective is to coordinate ship arrivals and departures as efficiently as possible. At any given time there is a set of known ships that must be scheduled, and re-scheduling occurs on an ongoing basis as conditions change. Ships utilize a certain amount of water space, and there is a limited amount of space surrounding the ports (called *quays*). Thus, this problem is essentially a spatial scheduling problem, although it is not referred to by this term in the literature. Imai et al. [28] developed a heuristic algorithm that incrementally places ships in the quay, working from the outer borders inwardly. At each step the target ship is placed within the open "window" and the window is updated to reflect the addition of the new target ship. However, an important insight is also made in this paper about the underlying nature of the problem: solving the scheduling problem at a given time is

equivalent to the *cutting stock problem* [29]. This problem involves determining how to cut stock material into rectangular pieces to be used in products in such a way that the minimum amount of stock is wasted (i.e. not able to be used in any product). This insight enabled an integer programming formulation of the problem.

In certain business and industrial scenarios it is critical to be able to rapidly detect and resolve spatial-temporal conflicts. Such problems are not concerned with optimization (as is the case with spatial scheduling) but rather the detection (and sometimes the resolution) of spatial-temporal conflicts. Song and Chua [62] address a problem in the construction industry involving the detection of spatial-temporal conflicts in project scheduling. They employ a vector- and logic-based approach for detection of such conflicts. Howorth and Sang [63] address a problem in airport operations that involves determining appropriate aircraft takeoff positions and times. They utilized a constraint-based approach that involves automated reasoning to detect and resolve potential spatial-temporal constraints. Their objective was to aid human controllers by suggesting feasible takeoff positions.

3.2. LITERATURE ON RELEVANT PACKING PROBLEMS

Because of the clear geometric aspects involved in spatial scheduling, literature addressing certain types of packing problems has a high degree of relevancy. Two types of packing problems are particularly relevant: bin packing problems and strip packing problems.

The classical *bin-packing problem* (BPP) [35] involves determining how to pack n objects into fixed-sized bins of size W in a way that minimizes the total number of bins

required. The simplest form of this problem involves one-dimensional bins and objects. A more complex variation of this problem, the *variable-sized bin-packing problem* (VSBPP) [31] involves determining how to pack n objects into bins where there are m different bin sizes W_i to choose from, each with an associated cost c_i. The objective is to determine a packing that minimizes the total cost. For each of these problem variants, two- and three-dimensional variations exist. For each of these variations of the problem, two- and three-dimensional variants also exist. The two-dimensional variations involve packing rectangular objects into larger rectangular bins. Some varieties require fixed orientations for the objects, while others may also allow objects to be rotated by 90 degrees. Analogously, the three-dimensional variations involve packing three-dimensional box-like objects into larger boxes. These problems may likewise have fixed orientations or permit one of six possible rotations to be selected for each object.

Another type of packing problem, the *strip-packing problem* (SPP) [8], is also very relevant to spatial scheduling. The classical SPP has some similarities to the two-dimensional bin-packing problem. The classical SPP involves determining how to pack two-dimensional rectangular objects into a two-dimensional strip – a container with a fixed width and without a fixed height. The objective is to determine how to pack the set of objects into the strip in a way that minimizes the required strip height. One variation on the classical SPP permits the objects to be rotated by 90 degrees. In addition, there are also three-dimensional variants of the SPP as well [38]. A three-dimensional SPP involves packing a set of three-dimensional boxes into a strip with a fixed length and width in a manner that minimizes the required height. As with three-dimensional BPP's,

some three-dimensional SPP variants also permit the objects to be rotated into one of six possible orientations.

Before continuing on to look at particular approaches to packing problems, it is appropriate to first examine the similarities and differences between packing problems and spatial scheduling problems.

3.2A) Similarities and Differences between Spatial Scheduling and Packing Problems

In Chapter 1 an example spatial scheduling problem was given, and as part of this example a visual representation of the solution was shown in Figure 1. In this depiction the packing aspect is clearly seen: each time a new job is placed inside the area to be processed, its location and orientation within the area must be determined. Furthermore, as jobs continuously enter and leave the processing area, the resulting available space can become very irregular and result in formidable challenges in determining where new jobs should go. Thus, there is a clear two-dimensional packing aspect to spatial scheduling.

Another way to conceive two-dimensional spatial scheduling problems is to understand them as three-dimensional problems, where the dimensions are processing area width (x-dimension), processing area height (y-dimension), and time (z-dimension). A depiction from this perspective is shown below in Figure 2.

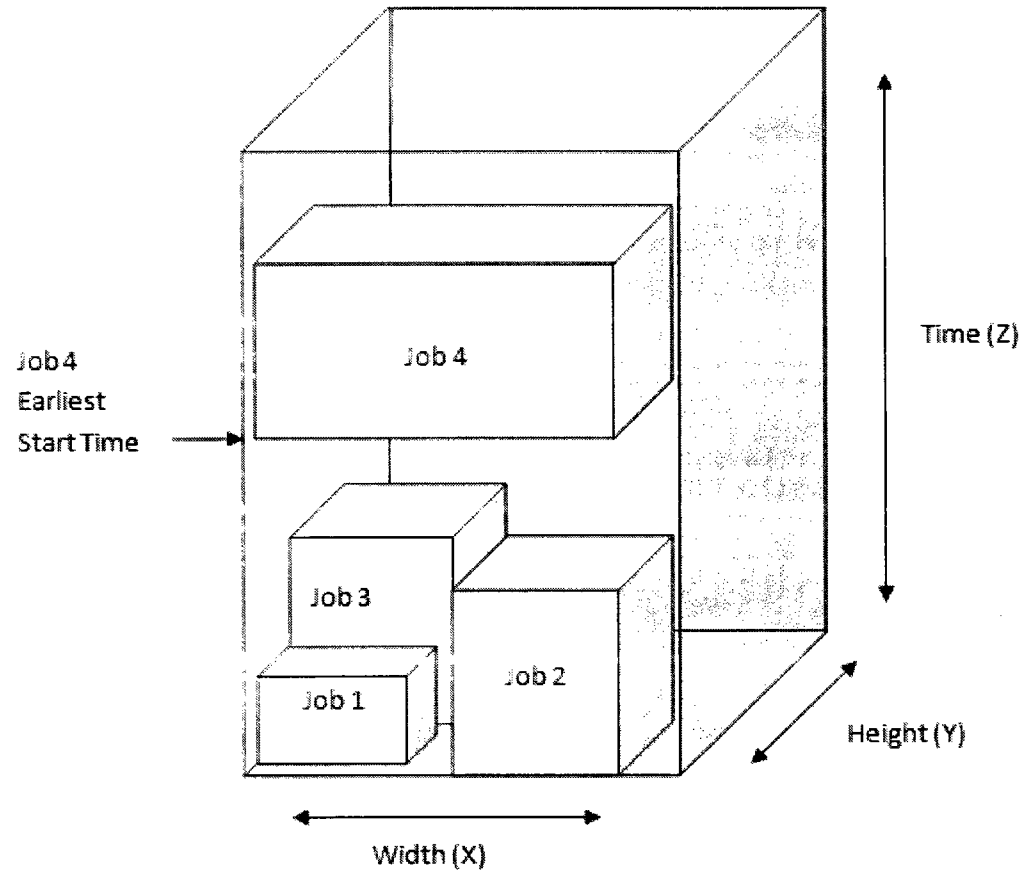

Figure 2: An example of a two-dimensional spatial schedule in three dimensions

It is clear by looking at this example that there are similarities between spatial scheduling and packing problems. Indeed at first glance, this example makes it tempting to think of a spatial scheduling problem as simply a box-packing problem in disguise. This would be a mistake, however, because it ignores some of the fundamental aspects of spatial scheduling: the earliest available start dates and the due dates associated with each job. The earliest start date associated prevent jobs from being placed at *just any* z-coordinate (i.e. start time). Similarly, the job due dates will certainly affect when the jobs should be started, and depending on the objective function may also directly bear on where such jobs should be placed as well. In addition, *when* the jobs are scheduled affects *where* they

can be placed and *how* they can be laid out, and vice versa. So although significant similarities exist, these problems in general cannot be reduced to box-packing problems and, thus, their temporal and spatial components simply cannot be addressed separately. Because of these similarities, however, a number of concepts and problem representations found in the packing literature can be brought to bear on the problem of spatial scheduling.

3.2.B) Packing Solution Methods in the Literature

The one-dimensional classical BPP is known to be NP-hard [34], and because the VSBPP is a generalization of the BPP it follows that it is also NP-hard [31]. As a consequence, heuristic approaches are most commonly utilized in practice. The classical BPP has been widely studied and there is a sizeable body of literature on this problem. One widely-used heuristic for this problem is the first-fit heuristic, a greedy algorithm that successively adds objects to bins. Each object is placed in the first bin it fits in, and new bins are added only when the current object does not fit into any currently utilized bin. Another well-known heuristic, the best-fit heuristic, places objects into the most-full bin in which it fits. As it turns out, these heuristics both perform generally well. Johnson et al. [35] proved that for a given BPP with optimal number of bins L*, both of these algorithms will never do worse than $\frac{17}{10}L* + 2$ bins. Furthermore, in this paper it was also shown that by sorting objects in decreasing order of size, these heuristics will never do worse than $\frac{11}{9}L* + 4$ bins. Such an algorithm is called a first-fit-decreasing (FFD) heuristic. An improved variation on FDD, the MFFD heuristic, was given by Garey and Johnson in [36] and was shown to perform in no worse than $\frac{71}{60}L* + 1$ bins [37].

A number of modern heuristics for the one-dimensional VSBPP are complied and reviewed in Haouari et al. [31]. In particular, six different heuristics for this problem are discussed and results compared. Of the six, four are constructive heuristics that involve iteratively using the subset-sum heuristic, one is a heuristic based on set covering, and one involves the use of genetic algorithms. An important empirical result discovered in this paper is that the set-covering heuristic performed extremely well for the VSBPP, yielding very near-optimal solutions to large problem instances while consuming only a small amount of computational time. This heuristic is based on column-generation and works in two phases: first, it generates a set of "attractive" feasible solutions prior to solving the set-covering model and secondly, it obtains an approximate solution by solving a restriction model. In the paper, the only discussion of this heuristic is in relation to the one-dimensional VSBPP. However, a reference is given to [33], where it is employed to solve a basic version of the two-dimensional bin-packing problem. In this version, all bins were of equal size and rotations on the rectangles were not permitted.

Another approach reviewed in [31], the genetic algorithm approach, is particularly relevant to spatial scheduling because the problem representation may be adapted for certain spatial scheduling variants, in particular those that involve more than one processing area. Multiple bins are inherent to the VSBPP, and the chromosome representation used in the paper suggests a good way to represent multiple-area variants of spatial scheduling problems. In this representation scheme, the chromosomes are represent a list of bins, and items are packed one at a time in the first bin until it is full, then the next bin is packed, and so on. One, two, and three-point crossover schemes are given that can all operate on this representation.

A wide variety of recent approaches for the two-dimensional SPP are reviewed by Riff, et al. in [41]. These approaches may be categorized into three broad classes: exact methods, heuristic methods, and meta-heuristic methods. Two exact approaches are reviewed that are both based on a branch-and-bound strategy. On exact approach by Lesh et al. [42] is reviewed that considers *perfect packing problems*, a very difficult type of strip-packing problem where an optimal solution results in no empty space left inside the strip. This approach employs branch-and-bound and tries to avoid making "holes" – open areas inside the strip surrounded by objects. In this approach, branches are cut when new objects cannot be placed inside the strip without generating a hole. This approach performed well for problems instances with less than 30 objects. Another exact method is that of Martello et al. in [43]. This approach relaxes constraints on the objects' areas to obtain a lower bound. It performed well on the same set of problems as experimented with in [42], and also on other test problems as well. In [41] it was concluded that in general, exact approaches work well for problems with less than 30 objects.

A wide range of heuristic methods have been developed for the SPP. One of the earliest, the *bottom-left* heuristic (BL), was introduced by Baker et al. [8]. BL entails ordering the objects by the amount of area they consume and placing them one at a time into the strip in a bottom-left-first manner. An improved version of this heuristic, developed by Chazelle [44], is called *bottom-left-fit* (BLF). This heuristic works by placing each object at the left-bottom-most location in which it will fit. Hopper and Turton [46] in turn improved the BLF into a new heuristic called BLD. BLD entails ordering the objects according to multiple criteria, such as height, width, and area. A greedy packing is obtained for each of these lists, and the best results are returned. Lesh

et al. developed a probabilistic version of BLD called *bubble-search* (BLD*) [47]. In this heuristic, objects are ordered in decreasing order according to some criteria (e.g. width, height, area, etc.). At each iteration, starting with the first object in the remaining list and working backward, the first object is greedily packed with a probability p. If it is not selected, another is selected in the same manner starting with the next item in the list. This packing process is carried out until all objects have been packed. It is then repeated as many times as can be executed within a specified time limit, and the best solution obtained is returned. Another heuristic, the *best fit decreasing height* (BFDH), was developed by Mumford-Valenzuela et al. [49] for *guillotine-cut* variants of the SPP. A guillotine-cut problem is one where the strip must be packed by cutting into sections using only straight-line "guillotine" cuts. Thus, a packing results in such a "guillotine-cut" division of the strip. BFDH first orders the objects from tallest to shortest. Objects are then packed in a bottom-left-first manner into the area, making "guillotine cuts" to the area and resulting in sub-areas. Objects are then packed left-justified into the resulting sub-area which results in the least amount of space remaining in that area. If an object cannot fit into any existing sub-area, a new guillotine cut is made for the object. Bortfeldt [50] developed an improved version of BFDH (called BFDH*) for use in initializing a genetic algorithm population. BFDH* permits rotations, and so selects the best orientation for each object at a given time. Additionally, before creating new sub-areas BFDH* attempts to first fill holes in existing sub-areas by making guillotine-cut packings to these sub-areas. Finally, Zhang et al. [52] present a recursive heuristic for guillotine-cut packing that involves packing an object, then recursively packing the two sub-areas that result. In this scheme, objects with larger areas are given higher priority

and are thus packed first. They reported very good results for this heuristic, with solutions coming within 1.8% of the optimum on Hopper's difficult benchmark problems.

A number of meta-heuristic approaches are also reviewed by Riff et al. in [41]. Soke et al. [53] present two hybrid meta-heuristic approaches. One involves combining simulated annealing with the BLF heuristic, and the other involves combining genetic algorithms with the BLF heuristic. In each case the meta-heuristic is used to determine the input ordering, and the BLF packing heuristic is then applied to the resulting input ordering. Bortfeldt [50] developed a genetic algorithm approach called the *strip packing genetic algorithm layer* (SPGAL) that according to [41] has produced the best results known in the literature for benchmark strip packing problems with rotations. An initial population is generated with the BFDH* heuristic. This results in solutions that are cut into "layers". A genetic algorithm then searches directly on these layers in such a way as to preserve the guillotine cut constraint. For non-guillotine-cut problems, a post-optimization procedure is used that breaks this layer structure and adds to the solution quality, although it becomes negligible as problems become larger. Finally, Burke et al. [54] developed three hybrid algorithms by combining TABU search, simulated annealing, and genetic algorithms, respectively, with the BF heuristic. In these algorithms, the BF heuristic was used to create the packings and the respective meta-heuristic was used to search through input orderings. It was found that the simulated annealing-BF hybrid gave the best quality solutions.

A few important conclusions are given in [41] regarding the different approaches that were evaluated. The first is that for packing problems of size less than 30, it is

generally feasible and thus appropriate to use an exact method such as branch-and-bound. The second is that the genetic algorithm-based SPGAL approach developed Bortfeldt performed the best of any known approach on benchmark packing problems where the objects may be rotated.

Most of the packing approaches reviewed up to this point have applied to one-or two-dimensional packing problems. Furthermore, it has been shown that two-dimensional spatial scheduling problems are essentially problems involving three dimensions. The differences between a one- or two-dimensional problem compared to a three-dimensional problem are not generally trivial. As a consequence, many solution methods for one-dimensional problems cannot be directly applied to their three-dimensional counterparts. One applied three-dimensional packing problem, *the container loading problem,* is addressed by Chen et al. [40]. This problem involves packing three-dimensional rectangular cartons into rectangular containers in such a way to minimize the total unused space inside all containers. A mixed-integer program is formulated for this problem and validated with a numerical example. Although correct, the computational performance of this approach was not good, requiring about fifteen minutes to obtain an optimal solution for a problem involving only six cartons. Wu et al. [32] modified this model to solve the VSBPP and reported similar computational results for this approach.

Certain meta-heuristics, such as genetic algorithms or simulated annealing, rely on probabilistic search and utilize generic operators for transforming solutions as well as generic objective functions. In order to be applied to a certain type of problem, they require only that problem-specific adaptations of the solution-transforming operators and

objective functions be developed. Thus, if such meta-heuristics have been successfully applied to three-dimensional packing problems, similar operators used to transform packing solutions may be able to be used to transform spatial scheduling solutions as well. These may be able to be used with differing objective functions to solve many different types of spatial scheduling problems. Thus, this represents one viable approach, granted that we can find these types of meta-heuristics applied to packing problems in the literature.

It turns out that such meta-heuristics have indeed been applied to three-dimensional packing problems. One approach used by Wu. et al. [32] applies a genetic algorithm to several variants of the three-dimensional bin-packing problem. In one variant involving only a single bin, the three-dimensional boxes can be rotated into one of six orientations when they are packed. Relying on a result in [39] that shows a greedy first-fit heuristic based on decreasing box volume is a good heuristic for 3D bin packing, a genetic algorithm scheme is devised. This scheme aims to determine an optimal box ordering and selection of orientations to feed into the heuristic so that the best possible packing results. In the problem representation the chromosome contains an ordering of boxes paired with the corresponding orientation of each box (a number from 1 to 6). In the initial population, all solutions are sorted in decreasing volume order and have randomly assigned orientations to each box. Roulette wheel-based selection is used for reproduction, and a one-point crossover scheme is used. In this scheme, two chromosomes are split at a certain point and the new chromosomes get one side from each parent. A repair scheme is also employed because under the crossover mechanism it is possible to have chromosomes that are either missing certain boxes or have duplicates.

Finally, two different types of mutations are used: one that swaps boxes, and one that changes the orientations of boxes.

Thus, this scheme is reminiscent of many hybrid algorithms employed to solve the 2D strip-packing problem reviewed in [41], where a heuristic is employed to pack objects based on an input ordering and a meta-heuristic is used to search through input orderings. This type of problem representation may be readily adapted to spatial scheduling problems. In one sense, the spatial scheduling problems are simpler than this type of bin-packing problem: Spatial scheduling jobs can have at most two possible orientations rather than six, since they cannot be "rotated" in time. The only missing piece is something analogous to the first-fit decreasing-volume heuristic. This warrants further investigation into such heuristics that may be used for spatial scheduling.

3.3. SYNOPSIS OF LITERATURE AND RESEARCH GAP

A cursory glance over this chapter will reveal that there is a significant body of literature that systematically addresses packing problems, but there are only a few miscellaneous papers addressing spatial scheduling by comparison. Although the claim is not made that every existing paper published that directly addresses spatial scheduling has been examined in this literature review, the literature that is presented on this topic appears to represent a nearly comprehensive review of such material at the writing of this dissertation.

Knowledge-based methods dominate the current literature addressing spatial scheduling. Thus, rather than relying primarily on an underlying body of spatial scheduling theory or known algorithms, the current state of the art is best represented by

the expert systems-approach. This approach is characterized by encoding expert domain knowledge of a specific problem into rule-based computer programs as can be seen in, for example, the literature addressing the block assembly or block painting problems in the ship-building industry. Because of their reliance on domain or industry-specific knowledge, these methods are generally brittle and cannot readily be transferred to problems in different domains.

What appears to be missing in the literature is an approach to spatial scheduling that aims to study the mathematical properties underlying such problems in general and that provides more general algorithms that can be readily applied to a broad range of problems. Currently there are no standard models of spatial scheduling problems. Furthermore, this scheduling sub-discipline lacks a firm body of theoretical knowledge suitable for general problem-solving. In light of this gap in knowledge, this dissertation aims to advance the fundamental understanding of spatial scheduling by developing standard models of spatial scheduling problems, analyzing the complexity of such problems, and developing new approximation algorithms capable of providing near-optimal solutions to a wide range of problems within reasonable amounts of computational time.

CHAPTER 4: OPTIMIZATION MODELS FOR SPATIAL SCHEDULING

In this chapter, integer programming models are developed for both single-area and multiple-area variants of spatial scheduling problems. These models are able to provide exact optimal solutions through computer solvers and, perhaps more importantly, provide insight into the essential mathematical properties of such problems. The models will be developed primarily with regard to one objective function in particular, namely to minimize total tardiness. However, the objective function is fairly interchangeable in these models, and this objective function was chosen simply because it is quite common in the literature. As will be shown in these model formulations, there are certain constraints that are inherent to nearly any spatial scheduling problem. Thus these constraints form a reusable core of virtually any spatial scheduling model. As an example of how this core can be readily transferred to other objective functions, a model formulation for a spatial scheduling problem with a different objective function, weighted earliness/tardiness, is given. Finally, several solved problems and computational results are also provided in this chapter.

4.1. SINGLE-AREA MODELS

To define the problem, it is assumed that a set of jobs is given. Each job consumes a certain rectangular space and has a required width, height, earliest start date, due date, and processing time. It is also required that all jobs be processed within a single larger, two-dimensional processing area of specified width and height. The objective is to determine when each job should start and also where inside the area each job should be

placed when it is processed in order to minimize the total tardiness for all jobs. The Tardiness for job j is defined as $T_j = \max(0, C_j\_d_j)$ where C_j is the job's completion time and $d_j$is its due date. Specifically, two variants of the problem are formulated: 1) a fixed-orientation variant, where jobs retain their specified orientation, and 2) a rotation variant, where jobs may be rotated by 90 degrees.

4.1A) Fixed-Orientation Model

As has been previously discussed, a problem of this type actually involves three dimensions: width (x), height (y), and time (z). By considering the problem from this perspective, the most distinguishing constraints become immediately perceivable – no two jobs can consume the same space at the same time. Feasible solutions thus require that for any two jobs i and j, either (I) i and j do not intersect in the width dimension (x-dimension), or (II) i and j do not intersect in the height dimension (y-dimension), or (III) i and j do not intersect in the time dimension (z-dimension).To explain this more precisely, consider the x-dimension. Let x_k denote the x-coordinate assigned to job k, and w_k denote the width of job k. Then (I) above can be stated mathematically as follows: For jobs i and j either $x_i + w_i \leq x_j$ or $x_j + w_j \leq x_i$, which can be rewritten as $x_i - x_j + w_i \leq 0$ or $x_j - x_i + w_j \leq 0$. In a similar manner, if we let h_k and p_k denote the length and required processing time for job k, respectively, we can rewrite I-III above as follows:

I. $x_i - x_j + w_i \leq 0$ or $x_j - x_i + w_j \leq 0$
II. $y_i - y_j + h_i \leq 0$ or $y_j - y_i + h_j \leq 0$
III. $z_i - z_j + p_i \leq 0$ or $z_j - z_i + p_j \leq 0$

The general form of each of these constraints can be reduced to the following constraint: $a \leq 0$ or $b \leq 0$ for decision variables a and b. This general constraint can then be rewritten as an equivalent set of constraints in a form appropriate for a linear programming model as follows: 1) let M be a constant larger than any possible value a or b can take on, and 2) let u_1, u_2 be decision variables. Then we have:

$a \leq 0 \; or \; b \leq 0 \equiv$
(1) $a - Mu_1 \leq 0$
(2) $b - Mu_2 \leq 0$
(3) $u_1 + u_2 \leq 1$
(4) $u_1, u_2 \in \{0, 1\}$

In this set of constraints, if $a \leq 0$ is not true then $u_1 = 1$, and a similar case holds for $b \leq 0$ and u_2. Because of this property, we will refer to u_1 as the violator of $a \leq 0$ and u_2 as the violator of $b \leq 0$. By letting u_1 and u_2 be binary decision variables that sum to at most 1, we assure that at least one of $a \leq 0$ or $b \leq 0$ is true. If we let α_{ij} denote the violator for the constraint $x_i - x_j + w_i \leq 0$, β_{ij} denote the violator for $y_i - y_j + h_i \leq 0$, and γ_{ij} denote the violator for $z_i - z_j + p_i \leq 0$ then we can ensure that at least one of constraints I-III above is true by using the following set constraints:

(6) $x_i - x_j + w_i - M\alpha_{ij} \leq 0$
(7) $x_j - x_i + w_j - M\alpha_{ji} \leq 0$
(8) $y_i - y_j + h_i - M\beta_{ij} \leq 0$
(9) $y_j - y_i + h_j - M\beta_{ji} \leq 0$
(10) $z_i - z_j + p_i - M\gamma_{ij} \leq 0$
(11) $z_j - z_i + p_j - M\gamma_{ji} \leq 0$
(12) $\alpha_{ij} + \alpha_{ji} + \beta_{ij} + \beta_{ji} + \gamma_{ij} + \gamma_{ji} \leq 5$
(13) $\alpha_{ij}, \alpha_{ji}, \beta_{ij}, \beta_{ji}, \gamma_{ij}, \gamma_{ji} \in \{0,1\}$

In the above set of constraints, if (6) or (7) are true then there is no intersection in the x-dimension, if (8) or (9) are true there is no intersection in the y-dimension, and if (10) or (11) are true there is no intersection in the z-dimension. Since there must be no intersection in at least one dimension, at least one of the violators must be 0 - hence constraint (12). Finally, there is constraint (13) so violators are binary decision variables. This set of constraints is required for every pair of jobs i and j in the problem instance. These constitute the most important core constraints common to all single-area problem instances.

In order to formulate the basic non-rotational single-area model, we will utilize the following nomenclature:

Constants	**Meaning**
n	The number of jobs to be processed
w_j	The width (x-dimension) of job j
h_j	The length (y-dimension) of job j
p_j	The processing time (z-dimension) of job j
e_j	The earliest start time of job j
d_j	The due date of job j
W	The processing area width (x-dimensional size)
H	The processing area height (y-dimensional size)
M	A constant larger than any conceivable decision variable value
Decision Variables	
x_j	The x-coordinate of job j
y_j	The y-coordinate of job j
z_j	The z-coordinate (start time) of job j
T_j	The tardiness of job j
α_{ij}	= 0 if $x_i - x_j + w_i \leq 0$ is true, 1 otherwise (explained above)
β_{ij}	= 0 if $y_i - y_j + h_i \leq 0$ is true, 1 otherwise (explained above)
γ_{ij}	= 0 if $z_i - z_j + p_i \leq 0$ is true, 1 otherwise (explained above)

Table 4: Nomenclature for the single-area fixed-orientation model

Model 1: Single-Area Fixed-Orientation Model

$$Minimize\ z = \sum_{j=1}^{n} T_j$$

Subject To:

$x_i - x_j + w_i - M\alpha_{ij} \leq 0 \ \ for\ i,j = 1 \ldots n, i \neq j$ (14)

$y_i - y_j + h_i - M\beta_{ij} \leq 0\ for\ i,j = 1 \ldots n, i \neq j$ (15)

$z_i - z_j + p_i - M\gamma_{ij} \leq 0 \ \ for\ i,j = 1 \ldots n, i \neq j$ (16)

$\alpha_{ij} + \alpha_{ji} + \beta_{ij} + \beta_{ji} + \gamma_{ij} + \gamma_{ji} \leq 5$
$for\ i,j = 1 \ldots n, i \neq j$ (17)

$x_j + w_j \leq W\ for\ j = 1 \ldots n$ (18)

$y_j + h_j \leq H \ \ for\ j = 1 \ldots n$ (19)

$z_j - e_j \geq 0 \ \ for\ j = 1 \ldots n$ (20)

$x_j, y_j, z_j \geq 0 \ \ for\ j = 1 \ldots n$ (21)

$\alpha_{ij}, \beta_{ij}, \gamma_{ji} \in \{0,1\} \ \ for\ i,j = 1 \ldots n, i \neq j$ (22)

$T_j - (z_j + p_j - d_j) \geq 0\ for\ j = 1 \ldots n$ (23)

$T_j \geq 0\ for\ j = 1 \ldots n$ (24)

In this formulation it is assumed that jobs are not rotated; rather they always preserve their specified orientation. Constraints (14)-(17) and (22) together ensure that any two jobs i and j do not intersect in at least one dimension. The rationale behind these constraints was explained above for (6)-(13). Constraints (18) and (19) ensure that no job goes outside the area's width and length, respectively. Constraints in (20) ensure that no job is assigned a starting time before its earliest start time. Constraints in (21) ensure that all coordinates are assigned non-negative values. Constraints (23) and (24) together appropriately set T_j for each job j. To understand these constraints, first consider that for job j with due date d_j, start time z_j, and processing time p_j, the ending time must be $z_j + p_j$. Thus, the tardiness for job j is $\max(0, z_j + p_j - d_j)$ since tardiness cannot be less than 0. Because the objective is to minimize the sum of all T_j, constraint (23) will

ensure that $T_j = z_j + p_j - d_j$ whenever job j is tardy – that is, whenever $z_j + p_j - d_j > 0$. In a similar way, constraints in (24) will ensure that $T_j = 0$ whenever job j is not tardy – that is, whenever $z_j + p_j - d_j \leq 0$.

4.1.B) Incorporating Rotations

Model 1 shown above does not permit jobs to be rotated. In reality jobs may sometimes be rotated in order to make better use of processing space. This model above can be extended to allow for 90-degree rotation of jobs by means of two basic modifications. The first modification is to make each job width (w_j) and height (h_j) into decision variables. In place of the prior constants w_j and h_j we use dimension constants D_{1j} and D_{2j} to arbitrarily designate the two dimensions of each job. The other modification is to introduce a pair of "orientation selectors" R_{1j} and R_{2j} for each job. The orientation selectors are binary decision variables, and each job has a pair of these that sums to one. The resulting model is shown below.

Model 2: Single-Area Rotational Model

$$Minimize\ z = \sum_{j=1}^{n} T_j$$

Subject To:

$$x_i - x_j + w_i - M\alpha_{ij} \leq 0\ for\ i,j = 1 \ldots n, i \neq j \quad (25)$$

$$y_i - y_j + h_i - M\beta_{ij} \leq 0\ for\ i,j = 1 \ldots n, i \neq j \quad (26)$$

$$z_i - z_j + p_i - M\gamma_{ij} \leq 0\ for\ i,j = 1 \ldots n, i \neq j \quad (27)$$

$$\alpha_{ij} + \alpha_{ji} + \beta_{ij} + \beta_{ji} + \gamma_{ij} + \gamma_{ji} \leq 5$$
$$for\ i,j = 1 \ldots n, i \neq j \quad (28)$$

$$x_j + w_j \leq W\ for\ j = 1 \ldots n \quad (29)$$

$$y_j + h_j \leq H\ \ for\ j = 1 \ldots n \quad (30)$$

$$z_j - e_j \geq 0\ \ for\ j = 1 \ldots n \quad (31)$$

$$x_j, y_j, z_j \geq 0\ for\ j = 1 \ldots n \quad (32)$$

$$\alpha_{ij}, \beta_{ij}, \gamma_{ji} \in \{0,1\}\ \ for\ i,j = 1 \ldots n, i \neq j \quad (33)$$

$$T_j - \left(z_j + p_j - d_j\right) \geq 0\ for\ j = 1 \ldots n \quad (34)$$

$$T_j \geq 0\ \ for\ j = 1 \ldots n \quad (35)$$

$$w_j - R_{1j}D_{1j} - R_{2j}D_{2j} = 0\ for\ j = 1 \ldots n \quad (36)$$

$$h_j - R_{1j}D_{2j} - R_{2j}D_{1j} = 0\ \ for\ j = 1 \ldots n \quad (37)$$

$$R_{1j} + R_{2j} = 1\ \ for\ j = 1 \ldots n \quad (38)$$

$$R_{1j}, R_{2j} \in \{0,1\}\ \ for\ j = 1 \ldots n \quad (39)$$

Constraints in (25)-(35) are exactly the same as (14)-(24) in the basic model. Constraint (38) ensures that each job has exactly one orientation selected. Constraints (36)-(37) use the selected orientation for each job *j* to assign one job dimension to the width variable w_j and the other to the height variable h_j. Finally, constraint (39) enforces a binary restriction on each orientation selector variable.

4.1.C) Computational Results

A number of problems were hand-generated in order to computationally test these models. The primary objective of these experiments was to verify the correctness of these models rather than to test the performance of a particular solution method. Excessive computational experiments thus avoided in favor of a small but carefully designed set of test problems. Seven instances for each problem variant (i.e. with and without job rotations) were devised by hand: one instance with 5 jobs, two with 6, two with 10, one with 13, and one with 15. Each problem used a processing area of width 10 and height 8. To verify that optimal solutions were found, all problem instances of size 5-10 jobs were intentionally designed to have an optimal objective function value of 0. In order to further test the models, certain problems were designed so that optimal solutions would require some processing space to be used by different jobs at different times. Additionally, for the variant permitting job rotations two nearly identical versions of the 5-job instance were developed, each containing a job having a dimension of 10 x 3. In this case only one orientation is possible. In order to verify correct orientation selection this job j was set to $D_{1j} = 10, D_{2j} = 3$ in the first version, and $D_{1j} = 3, D_{2j} = 10$ in the second. The models were formulated using ILOG OPL and solved using the ILOG CPLEX solver version 12.1.0 on a dual-core 2.1 GHz PC running Windows with 4 GB of RAM. For problem instances with ten or fewer jobs, optimal solutions were found in less than one second. Two solved 10-job problem instances are shown in Table 5. The non-rotation instance with 13 jobs was solved in 7 minutes and 7 seconds. For the remaining three problem instances, solutions could not be found within fifteen minutes of computational time. This

is consistent with the complexity results in Duin [2] and Paulus [1]. The complexity of such problems will be addressed in greater depth in the next chapter.

Problem with fixed job orientations						**Solution**			**Problem with job rotations**					**Solution**				
Job	w_j	h_j	e_j	p_j	d_j	x_j	y_j	z_j	D_{1j}	D_{2j}	e_j	p_j	d_j	w_j	h_j	x_j	y_j	z_j
1	4	2	0	6	8	6	1	0	4	2	0	6	8	4	2	1	4	0
2	4	5	2	4	8	6	3	4	4	5	2	4	8	5	4	3	0	4
3	2	4	3	5	8	0	0	3	2	4	3	5	8	4	2	1	6	3
4	1	3	1	6	10	5	5	4	1	3	1	6	10	1	3	0	2	4
5	3	1	1	5	6	7	0	1	3	1	0	2	6	1	3	0	5	1
6	10	3	1	3	5	0	4	1	3	10	0	3	5	10	3	0	1	0
7	2	1	1	4	6	5	0	1	2	1	0	4	6	2	1	0	0	0
8	5	4	2	5	9	0	4	4	5	4	0	5	9	5	4	5	4	1
9	3	2	0	9	10	2	2	0	3	2	0	9	13	3	2	0	0	4
10	8	2	3	7	13	2	0	6	8	2	0	7	13	2	8	8	0	6

Table 5: A solved 10-job instance for each problem variant

4.2. MULTIPLE-AREA MODELS

The models shown in the preceding section assume that there is only one processing area available in which jobs may be processed. In reality, there may be two or more processing areas available, each of differing dimensions, in which jobs may be processed in parallel. Multiple-area problems are essentially generalizations of their single-area counterpart previously discussed. The work in this section builds on that of the preceding section by extending models 1 and 2 into their multiple-area counterparts.

In the multiple-area version of the problem, we are given n jobs each having a width w_j, height h_j, processing time p_j, due date d_j, and earliest start date e_j. Each job must be processed inside a processing area, and we are given m available processing

areas each having a width W_k and height H_k. Jobs may be processed in parallel and any job may be assigned to any processing area, provided it can fit inside. The objective is to assign each job a start time, a processing area, and a location inside the assigned processing area in order to minimize total tardy time. As is the case for single-area problem variants, the tardiness for job j is defined as $T_j = \max(0, C_j - d_j)$ where C_j is the job's completion time and d_j is its due date. The objective will then be $\sum_{j=1}^{n} T_j$ where n is the total number of jobs. In this section multiple-area models for both fixed-orientation and rotational problem variants are formulated.

4.2.A. Rotational Model

Model 2 (the single-area rotational model) can be extended into a more general model that incorporates multiple areas. In the multiple-area problem, we assume we are given m processing areas each having a width W_k and height H_k. In order to extend Model 2 into one that accommodates multiple areas we must determine 1) how to assign each job to a processing area, 2) how to ensure that no two jobs *assigned to the same processing area* occupy the same space at the same time, and 3) how ensure that each job remains completely inside the bounds *of its assigned processing area*. In order to assign a job to an area, we employ a binary decision variable A_{jk} where $A_{jk} = 1$ if job j is assigned to area k, and 0 otherwise. To ensure that no two jobs assigned to the same processing area occupy the same space at the same time, we can modify constraints (25)-(28) in Model 2. First, we revise the indexing of violators $\alpha_{ij}, \beta_{ij}, \gamma_{ji}$ to account for multiple areas as follows:

$\alpha_{ijk} = 0$ if $x_i - x_j + w_i \leq 0$ is true inside of area k, 1 otherwise $for \quad i,j = 1 \ldots n, i \neq j, k = 1..m$ (40)

$\beta_{ijk} = 0$ if $y_i - y_j + h_i \leq 0$ is true inside of area k, 1 otherwise $for \quad i,j = 1 \ldots n, i \neq j, k = 1..m$ (41)

$\gamma_{jik} = 0$ if $z_i - z_j + p_i \leq 0$ is true inside of area k, 1 otherwise $for \quad i,j = 1 \ldots n, i \neq j, k = 1..m$ (42)

In Model 2, constraint (28) uses the violators to prevent any two jobs i and j from colliding (i.e. prevents them from intersecting in at least one dimension). For the multiple-area model we must revise this constraint to ensure only that two jobs i and j do not collide if they are assigned to the *same area*. We thus extend constraint (28) as follows:

$\alpha_{ijk} + \alpha_{jik} + \beta_{ijk} + \beta_{jik} + \gamma_{ijk} + \gamma_{jik} + A_{ik} + A_{jk} \leq 7 \quad for \quad i,j = 1 \ldots n, i \neq j, k = 1..m$ (43)

When jobs i and j are assigned to area k, it follows that $A_{ik} = 1$ and $A_{jk} = 1$. When this is the case, (43) simply reduces to (28). Furthermore, if jobs i and j are assigned to different areas, collision between them is impossible. Thus, by revising constraints (25)-(27) in Model 2 to use the violators and indexing shown in (40)-(42), and by replacing constraint (28) with (43), we can ensure that that no two jobs assigned to the same processing area occupy the same space at the same time.

Finally, in order to ensure that each job stays within the bounds of its assigned area, we must revise constraints (29) and (30) of Model 2 to incorporate multiple areas. We first introduce binary decision variables A'_{jk} where $A'_{jk} = 1$ if job j is *not* assigned to area k, and 0 otherwise. Thus, A'_{jk} is the complement to A_{jk} and by definition

$A_{jk} + A'_{jk} = 1$. In order to prevent job j from going outside its assigned area's width, we can revise constraint (29) as follows:

$$x_j + w_j - MA'_{jk} \leq W_k \; for \; j = 1 \ldots n, k = 1..m \quad (44)$$

In this constraint, $x_j + w_j > W_k$ only if $A'_{jk} = 1$. A similar case also holds for the height dimension. We are now able to formulate the multiple-area model permitting rotations as shown below.

Constants	**Meaning**
n	The number of jobs to be processed
m	The number of available processing areas
p_j	The processing time (z-dimension) of job j
e_j	The earliest start time of job j
d_j	The due date of job j
W_k	The width (x-dimensional size) of processing area k
H_k	The height (y-dimensional size) of processing area k
M	A constant larger than any conceivable decision variable value
Decision Variables	
x_j	The x-coordinate of job j
y_j	The y-coordinate of job j
z_j	The z-coordinate (start time) of job j
w_j	The width (x-dimension) of job j
h_j	The length (y-dimension) of job j
T_j	The tardiness of job j
R_{1j}	= 1 if $w_j = D_{1j}$ and $h_j = D_{2j}$, 0 otherwise
R_{2j}	= 1 if $w_j = D_{2j}$ and $h_j = D_{1j}$, 0 otherwise
A_{jk}	= 1 if job j is assigned to area k, 0 otherwise
A'_{jk}	= 1 if job j is *not* assigned to area k, 0 otherwise
α_{ijk}	= 0 if $x_i - x_j + w_i \leq 0$ is true in area k, 1 otherwise
β_{ijk}	= 0 if $y_i - y_j + h_i \leq 0$ is true in area k, 1 otherwise
γ_{ijk}	= 0 if $z_i - z_j + p_i \leq 0$ is true in area k, 1 otherwise

Table 6: Nomenclature for the multiple-area rotational model

Model 3: Multiple-Area Rotational Model

$$Minimize\ z = \sum_{j=1}^{n} T_j$$

$Subject\ To$:

$$x_i - x_j + w_i - M\alpha_{ijk} \leq 0 \quad for\ i,j = 1 \ldots n, i \neq j, k = 1..m \qquad (45)$$

$$y_i - y_j + h_i - M\beta_{ijk} \leq 0 \quad for\ i,j = 1 \ldots n, i \neq j, k = 1..m \qquad (46)$$

$$z_i - z_j + p_i - M\gamma_{ijk} \leq 0 \quad for\ i,j = 1 \ldots n, i \neq j, k = 1..m \qquad (47)$$

$$\alpha_{ijk} + \alpha_{jik} + \beta_{ijk} + \beta_{jik} + \gamma_{ijk} + \gamma_{jik} + A_{ik} + A_{jk} \leq 7 \quad for\ i,j = 1 \ldots n, i \neq j, k = 1..m \qquad (48)$$

$$x_j + w_j - MA'_{jk} \leq W_k \quad for\ j = 1 \ldots n, k = 1..m \qquad (49)$$

$$y_j + h_j - MA'_{jk} \leq H_k \quad for\ j = 1 \ldots n, k = 1..m \qquad (50)$$

$$z_j - e_j \geq 0 \quad for\ j = 1 \ldots n \qquad (51)$$

$$T_j - (z_j + p_j - d_j) \geq 0 \quad for\ j = 1 \ldots n \qquad (52)$$

$$T_j \geq 0 \quad for\ j = 1 \ldots n \qquad (53)$$

$$w_j - R_{1j}D_{1j} - R_{2j}D_{2j} = 0 \quad for\ j = 1 \ldots n \qquad (54)$$

$$h_j - R_{1j}D_{2j} - R_{2j}D_{1j} = 0 \quad for\ j = 1 \ldots n \qquad (55)$$

$$R_{1j} + R_{2j} = 1 \quad for\ j = 1 \ldots n \qquad (56)$$

$$R_{1j}, R_{2j} \in \{0,1\} \quad for\ j = 1 \ldots n \qquad (57)$$

$$\sum_{k=1}^{m} A_{jk} = 1 \quad for\ j = 1..n \qquad (58)$$

$$A_{jk} + A'_{jk} = 1 \quad for\ j = 1 \ldots n, k = 1..m \qquad (59)$$

$$x_j, y_j, z_j \geq 0 \quad for\ j = 1 \ldots n \qquad (60)$$

$$\alpha_{ijk}, \beta_{ijk}, \gamma_{jik} \in \{0,1\} \quad for\ i,j = 1 \ldots n, i \neq j, k = 1..m \qquad (61)$$

$$A_{jk}, A'_{jk} \in \{0,1\} \quad for\ j = 1 \ldots n, k = 1..m \qquad (62)$$

In this model, constraints (45)-(48) reflect the revisions necessary to prevent collisions between jobs assigned to the same area as discussed in (40)-(43). Similarly,

constraints (49)-(50) reflect the revisions necessary to ensure a job is placed within the bounds of its assigned area, as discussed in (44). Constraint (58) ensures that each job is assigned to exactly one area, and constraint (59) ensures that $A_{jk} \neq A'_{jk}$ for each job j and area k. Finally, constraint (62) enforces binary restrictions on the assignment and un-assignment variables A_{jk} and A'_{jk}. The remaining constraints are incorporated from the single-area model (Model 2) without modification.

It may be pointed out that the constraints in this model, excluding (28) and (29), can be transferred to other problem varieties with different objective functions.

4.2.B. Fixed-Orientation Model

The multiple-area model shown above (Model 3) assumes that jobs may be rotated by 90 degrees. In some cases, however, it may be required for jobs to have a fixed orientation. The model shown above may be easily adapted to accommodate this requirement by means of two modifications. First, each job's width w_j and height h_j are made to be pre-specified constants rather than decision variables. In this case D_{1j} and D_{2j} are not needed. Secondly, constraints (54)-(57) are simply removed from the model, as they specify how rotations should be selected. The resulting fixed-orientation model is shown below.

Model 4: Multiple-Area Fixed-Orientation Model

$$Minimize\ z = \sum_{j=1}^{n} T_j$$

Subject To:

$$x_i - x_j + w_i - M\alpha_{ijk} \leq 0 \quad for\ i,j = 1 \ldots n, i \neq j, k = 1..m \qquad (63)$$

$$y_i - y_j + h_i - M\beta_{ijk} \leq 0 \quad for\ i,j = 1 \ldots n, i \neq j, k = 1..m \qquad (64)$$

$$z_i - z_j + p_i - M\gamma_{ijk} \leq 0 \quad for\ i,j = 1 \ldots n, i \neq j, k = 1..m \qquad (65)$$

$$\alpha_{ijk} + \alpha_{jik} + \beta_{ijk} + \beta_{jik} + \gamma_{ijk} + \gamma_{jik} + A_{ik} + A_{jk} \leq 7 \quad for\ i,j = 1 \ldots n, i \neq j, k = 1..m \qquad (66)$$

$$x_j + w_j - MA'_{jk} \leq W_k \quad for\ j = 1 \ldots n, k = 1..m \qquad (67)$$

$$y_j + h_j - MA'_{jk} \leq H_k \quad for\ j = 1 \ldots n, k = 1..m \qquad (68)$$

$$z_j - e_j \geq 0 \quad for\ j = 1 \ldots n \qquad (69)$$

$$T_j - (z_j + p_j - d_j) \geq 0 \quad for\ j = 1 \ldots n \qquad (70)$$

$$T_j \geq 0 \quad for\ j = 1 \ldots n \qquad (71)$$

$$\sum_{k=1}^{m} A_{jk} = 1 \quad for\ j = 1..n \qquad (72)$$

$$A_{jk} + A'_{jk} = 1 \quad for\ j = 1 \ldots n, k = 1..m \qquad (73)$$

$$x_j, y_j, z_j \geq 0 \quad for\ j = 1 \ldots n \qquad (74)$$

$$\alpha_{ijk}, \beta_{ijk}, \gamma_{jik} \in \{0,1\} \quad for\ i,j = 1 \ldots n, i \neq j, k = 1..m \qquad (75)$$

$$A_{jk}, A'_{jk} \in \{0,1\} \quad for\ j = 1 \ldots n, k = 1..m \qquad (76)$$

4.2.C. Computational Results

As in the single-area case, the primary aim of these computational experiments is to verify the correctness of these (multiple-area) models rather than to test the performance of a particular solution method. Excessive experimentation is thus avoided

in favor of a small but carefully designed set of test problems. Five problem instances were developed for both the rotation and non-rotation models: one instance with 2 areas and 8 jobs, two instances with 2 areas and 12 jobs, and two instances with 3 areas and 16 jobs. In each instance, all processing areas were of different sizes and each instance included jobs not able to fit inside all processing areas. This was to verify the correct assignment of jobs to processing areas. In order to verify correct rotation selection, two versions were created of the 2-area, 8-job instance for the model permitting rotations. The larger area in this problem had a width of 8 and height of 5. A job was included in these instances that is able to fit only inside this larger area, having dimensions D1 = 7 and D2 = 4. In the alternate version D1 and D2 were switched so that D1 = 4 and D2 = 7. This enabled the correct orientation selection to be verified.

The models were formulated using ILOG OPL and each problem instance was solved using the ILOG CPLEX solver version 12.1.0 on a dual-core 2.1 GHz PC running Windows with 4 GB of RAM. For both problem variants (rotation and non-rotation) optimal solutions were found for all instances in under 1 minute, excluding one 3-area, 16-job problem instance whose jobs all shared an earliest start date, processing time, and due date. These conditions made the problem difficult to solve, and the solver could not find solutions within a 15-minute time limit for either the rotation or the non-rotation variant. This is consistent with the complexity results in Duin [2] and Paulus [1]. The complexity of spatial scheduling problems is discussed in more depth in the next chapter. One solved problem instance for both problem varieties is given in Table 1. These problems both utilized two processing areas: Area1 with width = 10, height = 5, and

Area2 with width = 5, height = 4. Both cases have an optimal objective function value of 1.

	Fixed-Orientation Problem					**Solution**				**Problem With Rotations**					**Solution**					
Job	w_j	h_j	e_j	p_j	d_j	x_j	y_j	z_j	Area	D_{1j}	D_{2j}	e_j	p_j	d_j	w_j	h_j	x_j	y_j	z_j	A.
1	3	3	2	4	8	0	1	4	1	3	3	2	4	8	3	3	0	2	4	1
2	2	1	0	5	8	0	0	3	2	1	2	0	5	8	1	2	0	2	3	2
3	7	4	2	6	10	3	0	2	1	4	7	2	6	10	7	4	3	1	2	1
4	1	1	0	2	9	0	0	1	2	1	1	0	2	9	1	1	0	2	0	1
5	4	3	4	4	9	0	1	4	2	3	4	4	4	9	4	3	1	1	5	2
6	2	3	1	3	4	0	2	1	1	3	2	1	3	4	3	2	0	0	1	1
7	6	2	6	1	8	0	1	8	1	2	6	6	1	8	6	2	0	0	8	1
8	2	2	5	2	11	0	0	9	2	2	2	5	2	11	2	2	0	0	9	1

Table 7: Two solved multiple-area problem instances: with and without rotations

4.3 ADAPTATION EXAMPLE: A WEIGHTED EARLINESS-TARDINESS PROBLEM

The models developed thus far have all been concerned with the objective of minimizing total tardiness. However, each model contains a core of constraints that can be readily transferred to other types of spatial scheduling problems. In order to demonstrate this more clearly, a model will be formulated for a problem with another widely-known and more complex objective function: the weighted earliness-tardiness objective function. In this objective function, both the earliness and tardiness of each job contribute a certain weighted cost to the objective function. The objective is to minimize total cost. Thus, given the earliness of job j is E_j with cost C_{1j} and tardiness T_j with cost C_{2j}, the objective is to minimize $\sum_{j=1}^{n} C_{1j}E_j + C_{2j}T_j$. The earliness of a job j is defined as $\max(0, d_j - (z_j + p_j))$.

In order to demonstrate the adaptation of core constraints, a model will be formulated for a single-area fixed-orientation problem with a weighted earliness-tardiness objective function.

Constants	**Meaning**
n	The number of jobs to be processed
w_j	The width (x-dimension) of job j
h_j	The length (y-dimension) of job j
p_j	The processing time (z-dimension) of job j
e_j	The earliest start time of job j
d_j	The due date of job j
C_{1j}	The weighted cost associated with the earliness of job j
C_{2j}	The weighted cost associated with the tardiness of job j
W	The processing area width (x-dimensional size)
H	The processing area height (y-dimensional size)
M	A constant larger than any conceivable decision variable value
Decision Variables	
x_j	The x-coordinate of job j
y_j	The y-coordinate of job j
z_j	The z-coordinate (start time) of job j
E_j	The earliness of job j
T_j	The tardiness of job j
α_{ij}	= 0 if $x_i - x_j + w_i \leq 0$ is true, 1 otherwise (explained above)
β_{ij}	= 0 if $y_i - y_j + h_i \leq 0$ is true, 1 otherwise (explained above)
γ_{ij}	= 0 if $z_i - z_j + p_i \leq 0$ is true, 1 otherwise (explained above)

Table 8: Nomenclature for the single-area fixed-orientation model with weighted earliness/tardiness objective function

Model 5: Single-Area Fixed-Orientation Weighted Earliness/Tardiness Model

$$Minimize\ z = \sum_{j=1}^{n} C_{1j}E_j + C_{2j}T_j$$

$Subject\ To$:

$x_i - x_j + w_i - M\alpha_{ij} \leq 0 \ \ for\ i,j = 1 \dots n, i \neq j$ (77)

$y_i - y_j + h_i - M\beta_{ij} \leq 0\ for\ i,j = 1 \dots n, i \neq j$ (78)

$z_i - z_j + p_i - M\gamma_{ij} \leq 0 \ \ for\ i,j = 1 \dots n, i \neq j$ (79)

$\alpha_{ij} + \alpha_{ji} + \beta_{ij} + \beta_{ji} + \gamma_{ij} + \gamma_{ji} \leq 5$
$for\ i,j = 1 \dots n, i \neq j$ (80)

$x_j + w_j \leq W\ for\ j = 1 \dots n$ (81)

$y_j + h_j \leq H \ \ for\ j = 1 \dots n$ (82)

$z_j - e_j \geq 0 \ \ for\ j = 1 \dots n$ (83)

$x_j, y_j, z_j \geq 0 \ \ for\ j = 1 \dots n$ (84)

$\alpha_{ij}, \beta_{ij}, \gamma_{ji} \in \{0,1\} \ \ for\ i,j = 1 \dots n, i \neq j$ (85)

$E_j - (d_j - (z_j + p_j)) \geq 0\ for\ j = 1 \dots n$ (86)

$E_j \geq 0\ for\ j = 1 \dots n$ (87)

$T_j - (z_j + p_j - d_j) \geq 0\ for\ j = 1 \dots n$ (88)

$T_j \geq 0\ for\ j = 1 \dots n$ (89)

In addition to constants found in Model 1, this model also contains C_{1j} and C_{2j}- constants for the weighted earliness and tardiness costs of each job j. Additionally, it contains the decision variables E_j - the earliness of each job j. In this model, constraints (86) and (87) have been derived from constraints (23) and (24) in Model 1, and constraints (88)-(89) are identical to (23)-(24). These are re-stated as follows:

$T_j - (z_j + p_j - d_j) \geq 0\ for\ j = 1 \dots n$ (23)

$T_j \geq 0\ for\ j = 1 \dots n$ (24)

The rationale for these constraints was explained for Model 1. In summary, (23) and (24) together ensure that T_j is the end time minus the due date of job j if job j is tardy, and

$T_j = 0$ if job j is early or on time. This same approach may also be used analogously for properly determining the earliness E_j:

$$E_j - (d_j - (z_j + p_j)) \geq 0 \; for \; j = 1 \ldots n \qquad (86)$$
$$E_j \geq 0 \; for \; j = 1 \ldots n \qquad (87)$$

Thus, apart from the objective function and the addition of two constraints, this model is otherwise identical to Model 1. In particular, (77)-(85) apply to virtually any single-area problem variant with fixed orientations. This example demonstrates how the core constraints developed in this chapter can in general be readily transferred intact to other variants of spatial scheduling problems.

CHAPTER 5: COMPLEXITY OF SPATIAL SCHEDULING

In this chapter, the worst-case complexity of spatial scheduling is examined. Using the properties of the models given in the last chapter, the complexity for branch-and-bound approaches will be examined first. Branch-and-bound is the most commonly used approach for solving integer programs. Next, a proof of NP-hardness by reduction will be given for a class of spatial scheduling problems. Finally, a number of complexity results from the literature will be discussed and consequently, some conclusions will be drawn about the complexity of spatial scheduling in general.

As spatial scheduling problems can vary widely, it is beyond the scope of this dissertation to be able to analyze the complexities of each of these on an individual basis. However, some problems variants are generalizations of others, and by analyzing the worst-case complexity of more specific variants conclusions may be drawn about their generalizations. For the spatial scheduling problems considered in this dissertation, the generalization relationships among problem variants are depicted below in Figure 3.

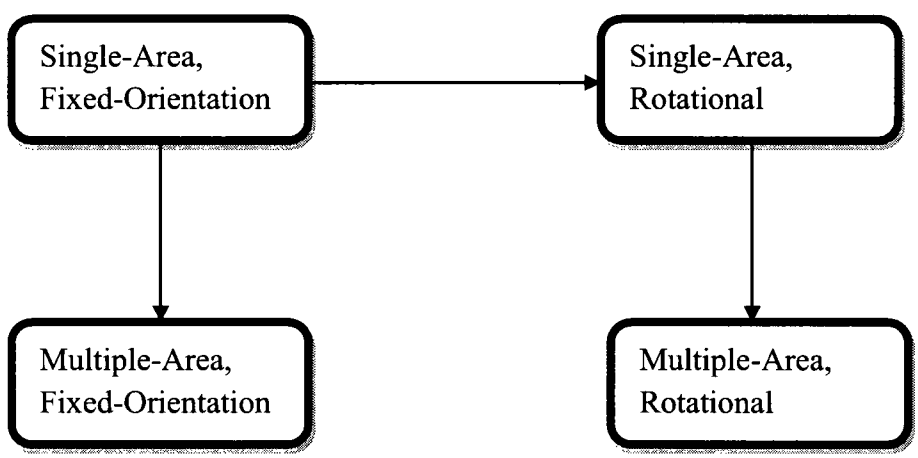

Figure 3: Generalization relationships among problem classes. Arrows go from specific to general.

In Figure 3 above, the most specific problem class is the single-area fixed-orientation. A rotational problem generalizes a fixed-orientation problem by simply removing the fixed orientation restriction. Thus, a solution to a fixed-orientation problem is a feasible solution when the fixed orientation requirement is removed, but not every rotational solution is feasible when a fixed orientation requirement is introduced. In a similar manner, a multiple-area problem also generalizes a single-area problem: a single-area problem is simply a multiple-area problem where the number of areas $m = 1$.

What is apparent from Figure 3 is that every type of spatial scheduling problem addressed by this dissertation is a generalization of the single-area fixed-orientation problem class. Thus, by drawing conclusions about the worst-case complexity of this class, conclusions can be made about the worst-case complexity of the rest of these classes as well. As a consequence, the majority of the analysis in this chapter will focus on this problem class.

5.1. COMPLEXITY FOR BRANCH-AND-BOUND INTEGER PROGRAM SOLUTION

Branch-and-bound is a common algorithm strategy for solving optimization problems, and is the most widely employed method for solving integer programs [27]. In this section the complexity of solving spatial scheduling problems through branch-and-bound will be analyzed.

5.1.A. Model Complexity

As previously discussed, this analysis will focus on the single-area fixed-orientation problem variant (Model 1). For Model 1 it is possible to determine the

number of binary variables, total decision variables, and constraints in terms of the number of jobs $\boldsymbol{n}$. We can define three functions to specify these quantities, respectively, in terms of $\boldsymbol{n}$: $\boldsymbol{B(n), V(n)}$, and $\boldsymbol{K(n)}$. For convenience, Model 1 is reproduced below.

Model 1: Single-Area Fixed-Orientation Model

$$Minimize\ z = \sum_{j=1}^{n} T_j$$

Subject To:

$$x_i - x_j + w_i - M\alpha_{ij} \leq 0 \quad for\ i,j = 1 \dots n, i \neq j \quad (14)$$

$$y_i - y_j + h_i - M\beta_{ij} \leq 0\ for\ i,j = 1 \dots n, i \neq j \quad (15)$$

$$z_i - z_j + p_i - M\gamma_{ij} \leq 0 \quad for\ i,j = 1 \dots n, i \neq j \quad (16)$$

$$\alpha_{ij} + \alpha_{ji} + \beta_{ij} + \beta_{ji} + \gamma_{ij} + \gamma_{ji} \leq 5$$
$$for\ i,j = 1 \dots n, i \neq j \quad (17)$$

$$x_j + w_j \leq W\ for\ j = 1 \dots n \quad (18)$$

$$y_j + h_j \leq H \quad for\ j = 1 \dots n \quad (19)$$

$$z_j - e_j \geq 0 \quad for\ j = 1 \dots n \quad (20)$$

$$x_j, y_j, z_j \geq 0 \quad for\ j = 1 \dots n \quad (21)$$

$$\alpha_{ij}, \beta_{ij}, \gamma_{ji} \in \{0,1\} \quad for\ i,j = 1 \dots n, i \neq j \quad (22)$$

$$T_j - (z_j + p_j - d_j) \geq 0\ for\ j = 1 \dots n \quad (23)$$

$$T_j \geq 0\ for\ j = 1 \dots n \quad (24)$$

1) *Number of binary variables in the model* $B(n)$: By examining Model 1 it can be observed that it contains the following binary variables: α_{ij}, β_{ij}, and γ_{ij}. Each of these variables is indexed over $i,j = 1 \dots n, i \neq j$. A variable having this indexing has $n(n-1) = n^2 - n$ instances. Since there are three such binary variables that have this indexing, it follows that $B(n) = 3n^2 - 3n$.

2) *Number of total variables in the model:* $V(n)$: In addition to the binary variables, the model also has the four additional types of decision variables: x_j, y_j, z_j, and T_j. Each of these variables is indexed over $j = 1 \dots n$, giving $4n$ total instances. Thus, $V(n) = B(n) + 4n = 3n^2 - 3n + 4n = 3n^2 + n$.

3) *Number of constraints in the model:* $K(n)$: As noted above, a variable indexed over $i, j = 1 \dots n, i \neq j$ will have $n^2 - n$ instances and the same holds true for constraints. Looking at the list above, we see that constraints (14)-(17) and (22) are indexed in this way. However, (22) puts three constraints together, giving a total of 7 constraints that have this indexing. This gives a total of $7n^2 - 7n$ constraint instances indexed over $i, j = 1 \dots n, i \neq j$.

Constraints (18)-(21) and (23)-(24) are indexed over $j = 1 \dots n$. It is also seen that (21) puts three constraints together, which gives a total of 8 constraints with this indexing. This gives a total of $8n$ constraint instances indexed over $j = 1 \dots n$. Thus, it follows that $(n) = 7n^2 - 7n + 8n = 7n^2 + n$.

5.1.B. Branch-and-Bound Solution Complexity

The spatial scheduling models developed in the previous chapter are all integer programs with binary decision variables. As a brief review, branch-and-bound solution of binary integer programs may be summarized as follows:

1. Solve a linear programming (LP) relaxation of the integer program (IP). If all decision variables are integers, terminate.
2. For non-integer-valued binary variables $x_1 \dots x_n$, choose a variable $x_{i \in \{1 \dots n\}}$ and solve two sub-problems (branches) A and B where $x_i = 0$ in A and $x_i = 1$ in B.

3. If:
    a. A and B both contain all integer-valued binary decision variables, choose the best solution
    b. A and B are both infeasible, there is no solution
    c. A or B contain non-integer-valued binary variables, recursively repeat step 2 for each sub-problem lacking a full integer solution
4. Always keep track of the best integer-valued solution found. At each branch, if a sub-problem does not give a solution better than the best found so far, the entire branch may be discarded.

Thus, branch-and-bound avoids doing unnecessary work through Step 4 – comparing best-case branch solutions to the best integer solution found up to that point, and ceasing to branch further when it will not lead to an optimal solution. It may be observed, however, that in the worst case a branch will be required on each integer variable. Under these conditions, for n variables 2^n branches would be required. This may be stated in terms of the big-O notation [56] as follows:

For a binary IP having n binary variables, the branch-and-bound solution method is $O(2^n)$. (90)

In the previous section the number of binary decision variables $B(n)$ was calculated for the single-area fixed-orientation model (Model 1). Specifically, $B(n) = 3n^2 - 3n$. Thus, (90) implies that the complexity of solving this spatial scheduling model by branch-and-bound for a given problem size of n jobs is $O(2^{3n^2-3n})$.

The famous *Boolean Satisfiability Problem* (SAT) is one the first problems to be proven NP-complete [57]. This problem involves assigning TRUE or FALSE to a set of Boolean variables in a logical proposition. The objective is to assign Boolean values to each variable in the proposition so as to make the proposition true. Because each variable can take on two values, TRUE or FALSE, in the worst case it requires trying 2^n possibilities for n variables. Thus, the complexity of the SAT problem is $O(2^n)$ - the same complexity as branch-and-bound has in terms of the number of integer variables. As a result, as the number of variables increases linearly, the worst-case number of steps required to solve the problem increases exponentially - rendering this problem intractable in the worst case. The following theorem may now be given:

Theorem 1: Solution of the single-area fixed-rotation integer program (Model 1) through branch-and-bound is at least NP-complete.

Proof: Let P be a single-area fixed-rotation spatial scheduling problem having n jobs, and let S be an SAT problem having n variables, where $n \geq 1$. Then:

$$O(P) = O\big(B(n)\big) = O\big(2^{3n^2-3n}\big) \geq O(2^n) = O(S)$$

Thus, it follows that the complexity of P solved with Model 1 through integer programming branch-and-bound is at least that of S.

5.1.C. A Reduction Proof of NP-Hardness

Strictly speaking, the results of the previous section apply only when the branch-and-bound method is employed to solve the problem formulated as an integer program. This leaves open the question of whether another method may be able to solve the

problem in polynomial time. In this section a proof will be given that a particular class of spatial scheduling problems is NP-hard.

The three-dimensional strip packing problem (3DS) was discussed in the literature review in Chapter 3. This problem involves an open-ended rectangular carton with fixed length and width, and an unbounded height. The objective is to determine a packing of n three-dimensional rectangular boxes into the carton in a way that minimizes the required height. The 3DS is an NP-hard problem [38]. By reducing 3DS to a spatial scheduling problem, the NP-hardness of this spatial scheduling problem can be established.

Theorem 2: Let S be a single-area spatial scheduling problem where 1) all jobs have a common earliest start time of 0, and 2) the objective is to minimize the total time required to complete all jobs. Then S is NP-hard.

Proof: Let G be a 3DS problem with a carton width W and length L, and a set B of boxes to be packed. We assume that each $b \in B$ is a triple having the form $b = (length, width, height)$. Now let S have a processing area of width W and length L, and a set of jobs J where each $j \in J$ is a triple having the form $j = (length, width, processing\ time)$. Then we can assign J as follows: $J = \{(w, h, p) \mid (w', h', l) \in B, w = w', h = h', and\ p = l\}$. Since B and J are identical and since the processing area and the carton have the same length and width, a schedule that minimizes the total processing time T for S is equivalent to a packing that minimizes the height H for G. Thus, G is solved by solving S, and since G is NP-hard it follows that S must also be NP-hard.

This result demonstrates NP-hardness for certain spatial scheduling problems, in particular those with a common earliest start time and an objective to minimize total job completion time. More specifically, this example illustrates the complexity of spatial scheduling through its relation to packing problems. Clearly, it may be concluded that spatial scheduling includes NP-hard problems.

5.1.D) Results from the Literature

The complexity of bin packing and strip packing problems has been widely addressed in the literature and virtually every form of these problems is known to be NP-hard. On the other hand, there is very little literature that addresses spatial scheduling directly. However, within a very small body of literature some important complexity results are found that have far-reaching implications for spatial scheduling.

One important result is found in Duin and Van Sluis [2]. In this paper they describe a problem quite similar to the single-area problems with which this dissertation is concerned, although somewhat simpler. The problem also bears similarities to the two-dimensional strip-packing problem (2DS) already discussed in the literature review. In 2DS a strip is given of a specified width, and n rectangles. The objective is to pack the rectangles within the strip to minimize the strip's height. The problem described in [2] involves jobs that have fixed start times and fixed horizontal positions, and the objective is simply to determine the vertical positions. This problem is proven to be NP-hard.

Some very important results for spatial scheduling are discussed in Paulus and Hurink [1]. In this paper, several elementary machine scheduling problems are considered including, among others, makespan minimization for parallel machines with

unit processing times and flow shop problems. These problems are known to be solvable in polynomial time. The *3-Partition Problem* [58] is a well-known NP-hard problem. This problem involves partitioning a list of numbers into three equally-sized lists. The objective is to create these three partitions in such a way that each list of numbers sums to the same number. Using reductions involving the 3-Partition Problem it is shown that the incorporation of a single spatial resource turns these problems into NP-hard problems. Very importantly, it is pointed out in [1] that similar reductions can be applied to virtually any scheduling problem, and that such reductions render these problems NP-hard with the introduction of any spatial resources. This leads to the conclusion that the problems addressed in this dissertation are NP-hard in general.

CHAPTER 6: A LOWER BOUND FOR THE TOTAL TARDINESS OBJECTIVE

In this chapter, a lower bound is derived for the objective of minimizing total tardiness in a multiple-area spatial scheduling problem. A lower bound provides the ability to objectively judge the quality of a proposed solution to an optimization (minimization) problem. Consequently, a lower bound provides an important theoretical insight into a problem. By definition, a lower bound for a given minimization problem is a value always less than or equal to the optimal objective function value for that problem. From this definition it is clear that the larger the lower bound is, the better the estimate of the optimal solution value it will provide. For many objective functions (including the total tardiness objective function) there is a trivial lower bound of zero. This is implicit in the non-negativity constraints common to many minimization problems. This provides little insight in estimating the optimal objective, however. In this chapter, a method is developed for finding a better lower bound.

In order to begin to derive the lower bound for the total tardiness objective, recall from Chapter 3 that a two-dimensional spatial scheduling problem is essentially a three-dimensional problem where each job has a width, height, and time dimension. With respect to the time dimension, each job has three different times associated with it: a processing time, an earliest start date, and a due date. When a job is processed it requires a certain amount of area (equal to width * height) for a certain amount of time (the specified processing time, to be specific). Thus, for a given job j this results in a three dimensional processing volume in space-time, where

$$\text{Processing volume}(j) = \text{width}(j) * \text{height}(j) * \text{processing time}(j) \qquad (91)$$

Job j requires this volume of space-time in order to be processed. If job j is processed within the interval of [earliest start date(j), due date(j)] – that is, it begins and ends within this interval – then job j will not be tardy. Feasibility constraints prevent job j from beginning before its earliest start date, but it may exceed the due date. Thus, the amount of open time available for job j to be processed without being tardy is

$$\text{Open time}(j) = \text{due date}(j) - \text{earliest start date}(j) \qquad (92)$$

This open time multiplied by the area of job j results in an open volume of space-time in which a job may be processed without being late:

$$\text{Open volume}(j) = \text{width}(j) * \text{height}(j) * [\text{due date}(j) - \text{earliest start date}\ (j)] \qquad (93)$$

At any given time, the absolute maximum amount of space available S_{max} for processing is the sum of area over each of the processing areas:

$$S_{max} = \sum_{a \in Areas} \text{width}(a) * \text{height}(a) \qquad (94)$$

It is normally assumed that the amount of processing time required by a job is not greater than the time interval between the earliest start date and the due date (otherwise it is impossible for the job not to be tardy). Thus, the sum of all jobs' open volume will be greater than or equal to the sum of the jobs' processing volume. However, because of the total area limitation S_{max}, not all of the open volume is usable. As will be shown, the difference between the total processing volume and the total *usable* open volume can be used to determine a lower bound. Accordingly, a critical step in determining this lower

bound lies in determining the amount of total usable volume. In order to do this, a timeline can be constructed from the earliest start dates and the due dates of each job. Events on this timeline occur when either a job's earliest start date occurs or its due date occurs. An event may entail any number of jobs becoming eligible to start or becoming due – this is simply determined if their earliest start dates or due dates coincide. By definition of this timeline, during each time interval between any two consecutive events no jobs become either eligible to start or become due.

Assume there are $n + 1$ events on a timeline, and let each event E_i be numbered where $E_i < E_j$ if $\text{time}(E_i) < \text{time}(E_j)$. This results in n distinct time intervals (referred to as *periods* for the remainder of this chapter) in which no jobs either become eligible to start or become due. The duration of time for period P_i beginning at $\text{time}(E_i)$ is given by:

$$\text{Duration}(i) = \text{time}(E_{i+1}) - \text{time}(E_i) \qquad (95)$$

Now let J_i denote the set of jobs with an earliest start date $\leq \text{time}(E_i)$ and a due date $\geq \text{time}(E_{i+1})$. Thus, each job $j \in J_i$ is eligible to be processed during period i and will not accumulate any tardiness during this period. Moreover, during this period there is a certain amount of total open volume contained:

$$\text{Open volume}(P_i) = \sum_{j \in J_i} \text{width}(j) * \text{height}(j) * \text{Duration}(i) \qquad (96)$$

However, as previously stated, not all of this open volume is necessarily usable. The total amount of available (two-dimensional) processing space S_{max} constrains the amount of usable volume. As a result, the maximum amount of usable volume during period i is $\text{Duration}(i) * S_{max}$. This gives the following usable volume for period i:

$$\text{Usable open volume}(P_i) = \min\left[\text{Open volume}(P_i),\ \text{Duration}(i) * S_{max}\right] \qquad (97)$$

Together with (91), this result enables the total amount of processing volume and usable volume to be computed, respectively:

$$\text{Total processing volume} = \sum_{j \in Jobs} \text{Processing volume}(j) \qquad (98)$$

$$\text{Total usable open volume} = \sum_{i=1}^{n} \text{Usable open volume}(P_i) \qquad (99)$$

The total processing volume is the *required* amount of space-time for job processing, while the total usable open volume is the maximum *available* amount of space-time in which jobs may be processed *without being tardy*. Because the total amount of available (two-dimensional) processing area S_{max} is fixed, it immediately follows that some jobs are guaranteed to be tardy whenever the total processing volume exceeds the total usable open volume.

This concept is illustrated in the figure below:

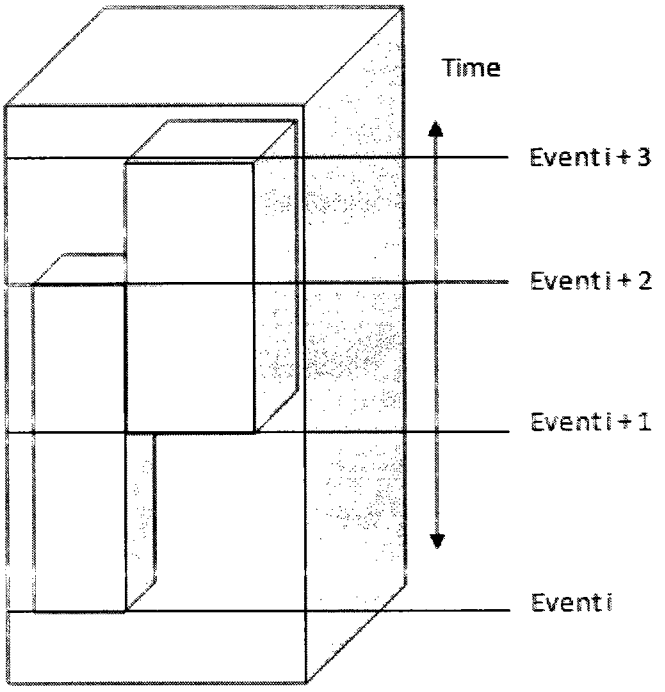

Figure 4: The open volumes of two jobs

The outer box represents the total amount of available processing area over time, and the left and right inner boxes represent the time intervals between two different jobs' earliest start dates and due dates. In other words, the inner boxes represent the two jobs' open volumes. The events are shown, and the periods are the time intervals that occur between the events. It is not difficult to imagine a situation where there are more inner boxes within a specific time interval than can fit inside the processing area. In this case, the maximum amount of usable open volume is the available processing area times the length of the time interval.

In order to determine a lower bound, maximum space utilization efficiency will be assumed. This assumption means that 100% of the available processing space (in other

words, S_{max}) is always utilized whenever jobs are processed. This assumption precludes any unused space within a processing area when jobs are processed. Because the total processing volume is a function of the jobs' width, height, and processing time, it is fixed. Under the maximum space utilization assumption the minimum amount of time T_{min} required to process all jobs must be as follows:

$$T_{min} = \frac{\text{Total processing volume}}{S_{max}} \qquad (100)$$

Under the same assumption, the maximum amount possible of usable processing time T_{uopen} is as follows:

$$T_{uopen} = \frac{\text{Total usable open volume}}{S_{max}} \qquad (101)$$

A lower bound may now be given.

Theorem 3: Let $L = \max(0, T_{min} - T_{uopen})$. Then L is a lower bound.

Proof: By definition, the total tardiness cannot be less than 0. Thus, only the case where $T_{min} > T_{uopen}$ needs to be considered. Consequently, it is assumed that $L = T_{min} - T_{uopen}$. In order to prove L is a lower bound it must be shown that L is always less than or equal to the minimum possible amount of total tardy time Z. Now assume to the contrary that $Z < L$ and let V_p denote the total processing volume as shown in equation (98) above. By the assumed value of L it follows that $T_{min} = L + T_{uopen}$ which gives the following:

$$(Z + T_{uopen})(S_{max}) < (L + T_{uopen})(S_{max}) = (T_{min})(S_{max}) = V_p \qquad (102)$$

This contradicts the definition of V_p, which is the volume of space-time *required* to process all jobs. ■

6.1. AN EXAMPLE DEMONSTRATING AN OPTIMAL LOWER BOUND

Although it is possible for problem instances to have an optimal objective value greater than the lower bound, an example can be quite easily demonstrated where the calculated lower bound is equal to the optimal solution. The existence of such problems demonstrates that this lower bound is *not* trivial and is capable of determining the optimal objective value for some problems. Assume a single processing area having a width = 2 and height = 1, and assume that the following jobs must be processed:

Job	**Width**	**Height**	**Earliest Start**	**Processing Time**	**Due date d_j**
1	1	1	0	2	3
2	1	1	0	2	3
3	2	1	0	3	4

Table 9: Example jobs to be processed

Because there is 1 processing area with dimension 2 x 1, it follows that $S_{max} = (2)(1) = 2$. The total processing volume can be calculated by summing each job's processing volume (width * height * processing time) as follows:

Total processing volume = $(1)(1)(2) + (1)(1)(2) + (2)(1)(3) = 10$

Events occur at times 0, 3, and 4; this totals three events or equivalently, two periods. Period 1 occurs from time 0 to 3, having duration(1) = 3, while period 2 occurs from time 3 to 4 having duration(2) = 1. Jobs 1, 2, and 3 are all eligible to be processed during period 1 without accumulating tardiness during this period, and job 3 is also eligible to be processed in period 2 without accumulating tardiness within the period. Recalling that the open volume for a period i is the sum of width(j) * height(j) * duration(i) over all jobs j eligible to be processed without tardiness in period i. This gives the following open volumes:

Open volume$(P_1) = (1)(1)(3) + (1)(1)(3) + (2)(1)(3) = 12$

Open volume$(P_2) = (2)(1)(1) = 2$

The usable open volume for period i is min [Open volume(P_i), Duration$(i) * S_{max}$], which gives the following:

Usable open volume$(P_1) = \min(12, 3 * 2) = 6$

Usable open volume$(P_2) = \min(2, 1 * 2) = 2$

The total usable open volume is the sum of usable open volumes over each period. Thus:

Total usable open volume = Usable open volume(P_1) + Usable open volume$(P_2) = 6 + 2 = 8$.

Dividing by $S_{max} = 2$, the following are obtained:

$$T_{min} = \frac{\text{Total processing volume}}{S_{max}} = \frac{10}{2} = 5$$

$$T_{uopen} = \frac{\text{Total usable open volume}}{S_{max}} = \frac{8}{2} = 4$$

These results give the lower bound $L = T_{min} - T_{uopen} = 5 - 4 = 1$.

An optimal solution to this problem is shown below:

Job *j*	**Start**	**Completion Time C_j**	**Due Date d_j**	**Tardiness T_j**	**X**	**Y**	**Width**	**Height**
1	0	2	3	0	0	0	1	1
2	0	2	3	0	1	0	1	1
3	2	5	4	1	0	0	2	1

Table 10: Optimal solution to example problem

Thus, the total tardy time is 1, which is equal to the calculated lower bound.

6.2. A POLYNOMIAL-TIME ALGORITHM FOR COMPUTING THE LOWER BOUND

The preceding results may be applied to develop a polynomial-time algorithm for computing the lower bound. This algorithm utilizes two types of data structures:

1. **Job**: This data structure contains the following attributes: width, height, earliest start date, due date, and processing time.
2. **Event**: This data structure contains a time and two sets of jobs: starting and ending. Jobs in the starting set are those that become eligible to begin at the event's time (i.e. the jobs whose earliest start times occur at the event's time). Jobs in the ending set are those that become due at the event's time (i.e. the jobs whose due dates occur at the event's time).

Pseudocode for the lower bound algorithm and corresponding auxiliary procedures is shown below. Comments follow the // symbol.

```
// The main algorithm.
Procedure COMPUTE_LB (Job_List, S_max)  DO:
   // Events is a time-ordered array of events – so Events[1].time < Event[2].time < …
   Events := BUILD_EVENT_LIST(Job_List)
   Open_Usable_Volume := 0
   FOR i = 1…Event_List.Length - 1  DO:
      Duration := Events[i + 1].time - Events[ i ].time
      Current_Open_Volume := ∑_{j∈Job_List} OPEN_VOLUME(j, Events[i], Events[i + 1])
      Open_Usable_Volume := Open_Usable_Volume + min(Current_Open_Volume, S_max)
   END FOR
   T_min := ∑_{j∈Job_List} Processing_Time(j) / S_max
   T_uopen := Open_Usable_Volume / S_max
   RETURN max (0, T_min − T_uopen)
END COMPUTE_LB
```

Table 11: COMPUTE_LB - the main lower bound algorithm

```
// Given a job with a start time and end time of a period, compute the corresponding open
// volume
Procedure OPEN_VOLUME (Job, Start_Time, End_Time)  DO:
    IF Job.Earliest_Start_Date <= Start_Time AND Job.Due_Date >= End_Time DO:
        RETURN (End_Time – Start_Time) * Job.Width * Job.Height
    ELSE DO:
        RETURN 0
    END IF-ELSE
END OPEN_VOLUME
```

Table 12: The OPEN_VOLUME procedure

```
// Generate and order the event list.
Procedure BUILD_EVENT_LIST (Job_List) DO:
    All_Times := {x | x = j.Earliest_Start OR x = j.Due_Date for j ∈ Job_List}.sort
    Events := ∅
    FOR i = 1… All_Times.Length DO:
        Evt := New Event with Evt.time = All_Times[ i ]
        FOR j ∈ Job_List DO:
            Add j to Evt.Starting IF j.Earliest_Start_Date = All_Times[ i ]
            Add j to Evt.Ending IF j.Due_Date = All_Times[ i ]
        END INNER FOR
    END OUTER FOR
    RETURN Events
END BUILD_EVENT_LIST
```

Table 13: The BUILD_EVENT_LIST procedure

It is assumed that the number of jobs is n. Since each job has an earliest start date and a due date, this gives two times for each job, making the number of events no greater than $2n \in O(n)$. The FOR loop inside the *Compute LB* procedure spans over the events, requiring $O(n)$ iterations. Inside this loop, the assignment of the *Current Open Volume* variable sums over each job, making it $O(n)$. Thus, the complexity of this FOR loop is $O(n * n) = O(n^2)$. The assignment of T_{min} sums over each job, requiring $O(n)$ operations. Thus, the total complexity to this point is $(n) + O(n^2) < 2O(n^2) \in O(n^2)$. However, since the *Build Event List* procedure is used at the beginning of the *Compute LB* procedure, the *Build Event List* complexity must also be analyzed and factored into *Compute LB* as well.

In the *Build Event List* procedure, the assignment of the *All Times* variable requires iterating over each job (which requires $O(n)$ operations) and sorting up to $2n$ unique times. Sorting n items is known to require $O(n * \log(n))$ operations [56]. Thus, this assignment requires $O(n) + O(n * \log(n)) < 2O(n * \log(n)) \in O(n * \log(n))$ operations. The outer FOR loop requires at most $2n \in O(n)$ iterations and the inner loop requires $n \in O(n)$ iterations, making the total complexity of the outer FOR loop $O(n * n) = O(n^2)$. Thus, the total complexity for the *Build Event List* procedure is $O(n * \log(n)) + O(n^2) < 2O(n^2) \in O(n^2)$.

Noting that the *Open Volume* procedure contains no loops, attention can be turned back to the *Compute LB* procedure. The complexity of *Compute LB* excluding the operations included in *Build Event List* was shown to be $O(n^2)$. The complexity of *Build*

Event List was also shown to be $O(n^2)$. Thus, the total complexity of *Compute LB* is $O(n^2) + O(n^2) = 2O(n^2) \in O(n^2)$.

CHAPTER 7: HEURISTIC ALGORITHMS

In this dissertation, a number of approximation algorithms will be developed to solve spatial scheduling problems under the total tardiness objective. Although the primary emphasis is on the total tardiness objective, the problem representations and data structures developed are fairly generic and can be readily adapted to problems with other objective functions as well. The approximation algorithm developed in this dissertation may be classified into two broad categories: heuristics and hybrid meta-heuristics. The emphasis of this chapter will be on heuristic algorithms, while hybrid meta-heuristic algorithms will be addressed in a subsequent chapter. This chapter will begin with a brief justification as to why approximate methods are generally necessary for solving spatial scheduling problems in practice. Next, several spatial scheduling heuristic algorithms will be developed. The heuristic algorithms developed all utilize a greedy approach and require no search. These aim to provide a reasonable solution in a very short amount of computational time. Finally, the algorithms are run on a small number of problems in order to verify their correct implementation, and the results are discussed.

7.1. WHY APPROXIMATE METHODS ARE NECESSARY FOR SPATIAL SCHEDULING

In general, it is always preferable to use exact methods wherever possible. The justification for using approximate methods comes when there is reason to believe that many of the problem instances one is interested in solving may not be able to be solved through exact methods. In the case of spatial scheduling problems, there are two principal reasons that indicate exact methods will not suffice for large, realistic problems.

The first reason to believe that exact methods will not generally suffice is because of the worst-case intractability of spatial scheduling problems. In Chapter 5, a proof was given of the NP-hardness for one class of such problems. In addition, results from the literature were presented that render virtually any scheduling problem that involves spatial resources to be NP-hard.

The second reason has to do with the complexity of using integer programming models to solve such problems, in particular the models developed in Chapter 4. In Chapter 5 it was shown that solving these models by branch-and-bound, the usual method of solution implemented in commercial software packages, is at least NP-complete. This indicates that when these problems are solved in practice, one can expect to run into problem instances that cannot be tractably solved. Perhaps most convincing of all, however, are the relatively small problem instances discussed in Chapter 4 that could not be solved. Specifically, there were several tight problem instances of size 13, 15, and 16 jobs that could not be solved. This demonstrates that even for small problems, unsolvable instances are likely be encountered.

Thus, from a theoretical perspective spatial scheduling problems have been shown to be intractable in general, and from a practical perspective a number of very small problem instances could not be solved using ILOG CPLEX, arguably the most powerful solver on the market at the writing of this dissertation. Consequently, approximate methods appear to be justified.

7.2. HEURISTIC ALGORITHMS

In this section a number of heuristic methods are developed for spatial scheduling. The primary intent of these methods is to provide feasible schedules of a modest (not necessarily optimal) quality, in a deterministic fashion.

7.2.A. Bottom-Left Time-Incrementing Heuristic

The bottom-left time-incrementing heuristic (BLTI) is a new heuristic developed as part of this dissertation research for deterministically constructing feasible spatial schedules based on the ordering of an input list of jobs. BLTI is designed to be used in combination with other heuristic and meta-heuristic methods, and the solution quality is entirely determined by the ordering of the input job list. BLTI schedules jobs to a single processing area. However, it can readily be incorporated (as will be shown in a later section) into multiple-area heuristics by first assigning jobs to processing areas, and then applying BLTI separately to each processing area to determine each assigned job's start time and location inside the area. Additionally, BLTI has two sub-variants depending on whether or not jobs are permitted to be rotated by 90 degrees: BLTI-F enforces fixed orientations, while BLTI-R rotates jobs as needed.

BLTI will always generate a feasible spatial schedule. A spatial schedule is feasible if three basic conditions hold:

1. No two jobs ever occupy the same space at the same time inside a processing area
2. Every job's assigned x and y coordinate is such that the job remains entirely within the bounds of its assigned processing area

3. No job is ever assigned a start time before its earliest start time

BLTI is essentially an adaptation of the bottom-left heuristic (BL) found in 2D strip-packing literature combined with a simple time-incrementing scheme. Jobs are sequentially assigned start times and (x, y) coordinates in the order in which they appear in the input list. A "current time" variable for the processing area is kept, which is incremented when jobs enter and exit the processing area. Whenever a new job is assigned its start time and location, the current time is set to the job's earliest start time if the current time was earlier than the job. In this way, no job is ever assigned an infeasible start time, ensuring condition 3 above.

In order to place jobs inside the processing area, BLTI relies on an adaptation of BL. When the next job is to be assigned its location inside the area, BLTI first sets the current time appropriately. BLTI then attempts to pack the job into the area using a bottom-left strategy - that is, to pack the job into the first feasible space found starting at the bottom left corner of the area and moving left-to-right, bottom-to-top. If jobs may be rotated (BLTI-R) and no such fit is found in this manner, BLTI rotates the next job by 90 degrees and attempts to pack this rotated job using the same bottom-left strategy. It may be that the current job may not be able to be packed into the area at the current time, due to other jobs occupying all the available space. When this is the case, BLTI increments the current time to the end time of the next-ending job and removes this job from the processing area. It then attempts to fit the next job into the processing area, and this process of removing jobs currently inside the area continues until the next job can be fit inside.

A job is represented in BLTI as a simple data structure having the following attributes:

1. Job Number
2. Due Date
3. Earliest Start Time
4. Processing Time
5. Width (Initially as specified)
6. Height (Initially as specified)
7. X (Initially 0)
8. Y (Initially 0)
9. Start Time (Initially 0)
10. End Time (Initially 0)

Attributes 1-4 are fixed and never change, while attributes 5-10 are modified by the algorithm. The width and height for each job are specified in the problem statement; however, their values may be swapped if the job is rotated by 90 degrees.

The processing area is represented in BLTI as a data structure having a width and height. The essential function of the area data structure is to keep track of what space inside the area is available and what space is occupied by jobs at the current time. The area data structure supports two operations: PACK and CLEAR. The PACK operation takes a single job as input. Using the bottom-left strategy, it tries to pack the job and assign the job's resulting X and Y coordinate. If it is successful it internally marks the space consumed by that job as occupied and unavailable. If successful it returns SUCCESS and if unsuccessful it returns FAIL. The CLEAR operation takes a single job

as input and uses the job's X, Y, width, and height to determine which space inside the area is occupied by that job. It then internally marks that space as open and available to be used by other jobs.

The following variables are utilized by the BLTI algorithm:

1. *Current_Time:* The current schedule time
    - Continuously incremented by the algorithm
2. *Open*: A list of all jobs currently remaining to be scheduled
    - Initially, all jobs in problem statement
    - Order is pre-determined before BLTI
    - This order is ALWAYS preserved
    - Jobs are removed from this list when they are assigned their processing time and coordinates
3. *In_Processing*: A list of jobs currently inside the area in processing
    - Initially empty
    - Sorted in ascending order by end time (earliest end time first)
    - This order is ALWAYS preserved
4. *Area*: An area data structure
    - Width and height reflect the width and height of the processing area

At a high level, the algorithm involves the alternating between following steps until the *Open* list is empty:

1. Move jobs from *Open* to *In_Processing* one at a time until no jobs can be packed into *Area.* With each move, *Current_Time* is updated to the next job's earliest start time. In between of each these moves, the jobs in *In_Processing* whose end times occur prior to or at the next job's earliest start time are removed to make all possible space available.
2. Remove the next job to be completed out of *In_Processing* and update *Current_Time* to this job's end time.

The BLTI algorithm is described in pseudo-code in the table below. Comments follow the // symbol.

```
Procedure BLTI (Open, Area_Width, Area_Height)  DO:
  In_Processing := Ø
  Current_Time := 0
  Area := Initialize empty area data structure with dimension Area_Width x Area_Height
  WHILE Open IS NOT EMPTY DO:
    Next_Finished_Job := NEXT_ELEMENT(In_Processing)
    // Remove jobs that are finished
     WHILE In_Processing IS NOT EMPTY AND Next_Finished_Job.end_time ≤
Current_Time  DO:
        Area.REMOVE(Next_Finished_Job)
        DELETE Next_Finished_Job FROM In_Processing
        Next_Finished_Job := NEXT_ELEMENT(In_Processing)
    END
    // Add jobs until we run out of space or all open jobs have been added
    Next_Job := NEXT_ELEMENT(Open)
    Current_Time := MAX(Next_Job.earliest_start_time, Current_Time)
    Successful_Pack := PACK(Area, Next_Job)    // Procedure defined in Table below
    WHILE NOT Successful_Pack DO:
        Next_Finished_Job := NEXT_ELEMENT(In_Processing)
        Area.REMOVE(Next_Finished_Job)
        DELETE Next_Finished_Job FROM In_Processing
        Current_Time := MAX(Next_Job.earliest_start_time,
Next_Finished_Job.end_time)
        Successful_Pack := PACK(Area, Next_Job)
    END
    Next_Job.start_time := Current_Time
  END
END
```

Table 14: The BLTI Algorithm

The PACK procedure is called by the BLTI. This procedure delegates to the area data structure's PACK operation to assign a set the X and Y coordinates. The PACK procedure essentially encapsulates the rotation procedure before utilizing the Area's PACK procedure. The PACK procedure is shown below.

```
Procedure PACK (Area, Job) DO:
  Successful_Pack := Area.PACK(Job) // Utilize the Area's PACK procedure
  IF NOT Successful_Pack DO:
    # Change layout – rotate job 90 degrees
    SWAP(Job.Width, Job.Height)
    RETURN Area.PACK(Job)
  ELSE
    RETURN Successful_Pack
  END
END
```

Table 15: The PACK procedure

7.2.B. Combining BLTI with Round-Robin Area Assignment for Multiple Areas

BLTI can readily be adapted to incorporate multiple processing areas. There are potentially a wide variety of schemes that can be used to assign jobs to areas. A very simple approach that can be used for deterministically assigning jobs to areas is a slightly modified version of round-robin. In a purely round-robin approach, the assignment algorithm would cycle sequentially through the list of processing areas when assigning jobs, until no more jobs remain. In this fashion, each area would have at most one job more than any other area. However, it is possible that not every job will fit inside every

processing area. A simple modification can be added to remedy this problem: if the current job to be scheduled will not fit in the current area, simply move on to then next area or until an area is found in which the job can fit. The pseudo-code for his multiple-area version of BLTI is shown below.

```
Procedure BLTI-RR (Jobs[1...n], Areas[1...m]) DO:
    Position := 1
    Assignments := m x n matrix // Assignments[i] is a list of jobs assigned to the ith area
    // First, assign jobs to areas
    FOR j := 1, j ≤ n DO:
        a := Areas[Position]
        IF Jobs[j] fits inside a DO:
            ADD Jobs[j] to Assignments[Position]
        ELSE
            WHILE Jobs[j] does not fit inside a DO:
                INCREMENT Position or reset to 1 if greater than m
                a := Areas[Position]
            END
        END
        INCREMENT j
    END
    // Second, apply BLTI to each area based on the jobs assigned through RR
    FOR k ∈ {1...m} DO:
        BLTI(Assignments[k], Areas[k].Width, Areas[k].Height)
    END
END
```

Table 16: BLTI combined with round-robin area assignment for multiple areas

There are two important properties of BLTI-RR. The first property is that assignment of jobs to areas is determined by input ordering. By looking at the algorithm above, the other important property of this area assignment may be noted: area input order preservation. That is, for any two jobs *a* and *b* in an input list for BLTI-RR where *a* comes before *b*, if *a* and *b* are assigned to the same area then *a* will always come before *b* in that area's assigned input list of jobs. Together these two properties, determination of area assignment by input ordering, along with preservation of original input ordering for each area's input list, enable BLTI-RR to be used or called from procedures in essentially the same way as BLTI. Finally, it should be pointed out that when there is only one processing area, the behavior of BLTI-RR exactly reduces to that of BLTI.

7.2.C. Multiple-Area BLTI with External Assignment of Processing Areas

BLTI-RR provides the ability to assign jobs to processing areas in addition to scheduling the jobs within their assigned areas. In some cases it may be desirable for the assignment of processing areas to be fixed while the heuristic is used for scheduling only. Another multiple area adaptation of BLTI is possible. This variant will be called BLTI-External. It assumes that each job has an attribute called *Assigned_Area* in addition to the usual job attributes. The pseudocode for BLTI-External is shown below.

```
Procedure BLTI-External (Jobs[1...n], Areas[1...m]) DO:
   FOR EACH Area in Areas DO:
      Area_Jobs := ∅
      // Select out ONLY jobs assigned to current area, then perform BLTI on current
area.
      FOR i IN 1...n DO:
         ADD Jobs[i] TO Area_Jobs IF Jobs[i].Assigned_Area = Area
      END FOR
      BLTI (Area_Jobs, Area.Width, Area.Height)
   END FOR
END BLTI-External
```

Table 17: The BLTI-External heuristic

7.2.D. Earliest-Due-Date-First Heuristic

Although the BLTI heuristic (and its multiple-area variants) is guaranteed to provide feasible solutions, it does not aim towards optimizing any particular objective function. Rather, BLTI simply constructs a feasible solution deterministically based on the order of the input job list. Thus, the quality of solutions generated by BLTI is entirely determined by this input ordering.

The earliest-due-date-first (EDD) heuristic is a simple and widely-known heuristic in the scheduling literature [24], and has been applied to a number of objective functions including the total tardiness objective function. EDD can be readily combined with BLTI to produce good solutions to many problem instances for the total tardiness objective function and this heuristic can produce optimal solutions for sufficiently sparse problem instances. Intuitively, a sparse problem instance is one where there is a large amount of space available resulting in little "competition" for space for each job, or

where due dates and earliest start times are such that all or most jobs can have the entire area to themselves. This approach by no means guarantees optimality in general, however, and the solution quality may rapidly deteriorate as problems become tighter. The heuristic involves no search and consequently executes very fast, as was shown in Garcia and Rabadi [60]. The EDD-BLTI heuristic is described in pseudo-code in the table below.

```
Procedure EDD-BLTI (Jobs)  DO:
    SORT Jobs in earliest-due-date-first order
    Apply BLTI to Jobs
END
```

Table 18: The EDD heuristic for spatial scheduling

In the procedure above, the BLTI can be any variant of BLTI, including BLTI-RR for multiple areas.

7.2.E. Computational Results

Extensive computational experiments and solution quality analyses are discussed in a subsequent chapter. However, in order to inform the design of the hybrid algorithms it is necessary to have at least a basic idea of how the heuristics perform, especially in terms of computational speed. For verification of correctness and to gain a preliminary understanding of the heuristic performance, an implementation of EDD combined with BLTI-RR was programmed using the Ruby programming language. A small 20-job

problem for a single processing area with a width 100 and height of 60 is shown below, along with the results generated by this algorithm.

Job	Width	Height	Earliest Start	Processing Time	Due date dj
1	4	5	2	10	20
2	8	2	4	8	18
3	3	7	3	18	32
4	2	2	3	8	16
5	11	3	0	2	14
6	4	4	0	6	12
7	6	4	2	8	22
8	4	6	10	10	38
9	2	6	12	6	40
10	9	4	22	20	66
11	5	14	11	7	33
12	3	3	8	8	30
13	1	8	30	10	60
14	5	12	7	11	80
15	6	8	6	25	72
16	3	7	15	16	50
17	15	4	0	4	12
18	16	2	40	14	82
19	5	15	33	9	52
20	6	23	25	21	77

Table 19: A small 20-job problem

Job	X	Y	Width	Height	Start	End	Area
17	0	0	15	4	0	4	1
6	0	3	4	4	0	6	1
5	0	2	11	3	0	2	1
4	0	1	2	2	3	11	1
2	0	1	8	2	4	12	1
1	0	4	4	5	4	14	1
7	0	3	6	4	4	12	1
12	0	2	3	3	8	16	1
3	0	6	3	7	8	26	1
11	0	13	5	14	11	18	1
8	0	5	4	6	11	21	1
9	0	5	2	6	12	18	1
16	0	6	3	7	15	31	1
19	0	14	5	15	33	42	1
13	0	7	1	8	33	43	1
10	0	3	9	4	33	53	1
15	0	7	6	8	33	58	1
20	0	22	6	23	33	54	1
14	0	11	5	12	33	44	1
18	0	1	16	2	40	54	1
Total Tardiness	0.0						

Table 20: EDD-BLTI-RR algorithm-generated solution to the small 20-job problem instance. Jobs are listed in earliest-start-time-first order.

The solution generated by the algorithm is optimal, as may be seen by the total tardy time. It may be noted that the Y-coordinate for each job assignment was 0, and that at no point in time was the processing area near capacity. This problem may be said to be sparse, and in general the EDD-BLTI heuristic will always generate optimal solutions for sufficiently sparse problems under this objective function.

Each of the remaining problems discussed in this chapter were used in Garcia and Rabadi [60]. Each problem instance was generated using Algorithm 1 found in Appendix

1 (the problem generation algorithms utilized in this dissertation are specified in Appendix 1). In-depth computational experiments and solution quality analysis are discussed in a later chapter; the results shown here simply give a brief overview of the general computational speed and quality ranges. A single processing area with width of 10 and height of 7 was used in each problem instance. Problem sizes ranged from 100 to 10,000 jobs. The problem-generation Algorithm 1 in Appendix 1 uses a tightness factor ranging from 1.0 to 10.0, and the problems generated use tightness factors ranging from 3.3 to 9.9. The results of these computational experiments are shown in the table below.

Problem Instance	**Number of Jobs**	**Tightness**	**Total Tardiness (Objective Function)**	**Computational Time (seconds)**
E-100	100	3.3	0	0.125
H-100	100	9.9	668	0.156
E-500	500	3.3	4	0.624
H-500	500	9.9	94462	0.717
E-1000	1000	3.3	0	1.4
M-1000	1000	8.5	2328	1.26
H-1000	1000	9.9	147,225	1.26
E-10000	10,000	3.3	0	13.76
M-10000	10,000	8.9	256,152	14.69
H-10000	10,000	9.9	3,142,787	14.18

Table 21: Results for experimental problems

A number of features may be pointed out about these solutions. Perhaps most observable is the speed at which this algorithm can produce solutions. 100-job instances required far less than one second of computational time, while the longest processing time of any problem instance took less than 15 seconds to process 10,000 jobs. It may also be observed that this algorithm was able to obtain optimal solutions for all sparse (tightness = 3.3) problem instances, regardless of size. Thus, EDD-BLTI executes very

fast and can arrive at good and even optimal solutions for many problem instances, particularly sparse instances.

CHAPTER 8: HYBRID META-HEURISTIC ALGORITHMS

In this chapter, a number of hybrid meta-heuristic algorithms for spatial scheduling are developed. In each case, a heuristic algorithm (specifically, forms of the BLTI heuristics developed in the previous chapter) is utilized in conjunction with a meta-heuristic algorithm. The heuristic is utilized to rapidly construct feasible solutions, while the meta-heuristic is utilized to search through the space of inputs into the heuristic. This chapter begins by introducing a general framework for hybrid meta-heuristic spatial scheduling algorithms. While this framework forms the basis of all subsequent algorithms developed, it is not restricted to these algorithms and may also be used to combine many different types of heuristics and meta-heuristics for solving spatial scheduling problems. Several hybrid heuristic algorithms based on different varieties of local search and simulated annealing are developed. In-depth computational experiments and solution quality analyses will be discussed in a later chapter.

8.1. A FRAMEWORK FOR HYBRID SPATIAL SCHEDULING APPROXIMATION ALGORITHMS

Hybrid approximations algorithms have been used very successfully in the bin- and strip-packing literature (for example, Wu [32] or Soke [53]). This literature was covered in significant detail in the literature review (Chapter 3). In general, these hybrid algorithms combine a greedy heuristic together with a meta-heuristic to obtain near-optimal solutions. In this approach a heuristic is used to construct a packing in a greedy and most often deterministic manner, while a meta-heuristic is used to modify and search through the space of inputs fed into the greedy heuristic. In most of the literature that uses

this approach, the meta-heuristic operates on input orderings while the greedy heuristic is dependent upon its input orderings.

Because of the success on packing problems, similarity in structure of spatial scheduling problems to packing problems, and the existence of greedy heuristics to construct feasible schedules (e.g. BLTI), a similar hybrid approach may be used for spatial scheduling problems. Moreover, a general framework based upon this principle can provide an infrastructure for the implementation and testing of many approximation algorithms for spatial scheduling. Such a framework is outlined below and forms the basis of the remaining approximation algorithms developed in this dissertation.

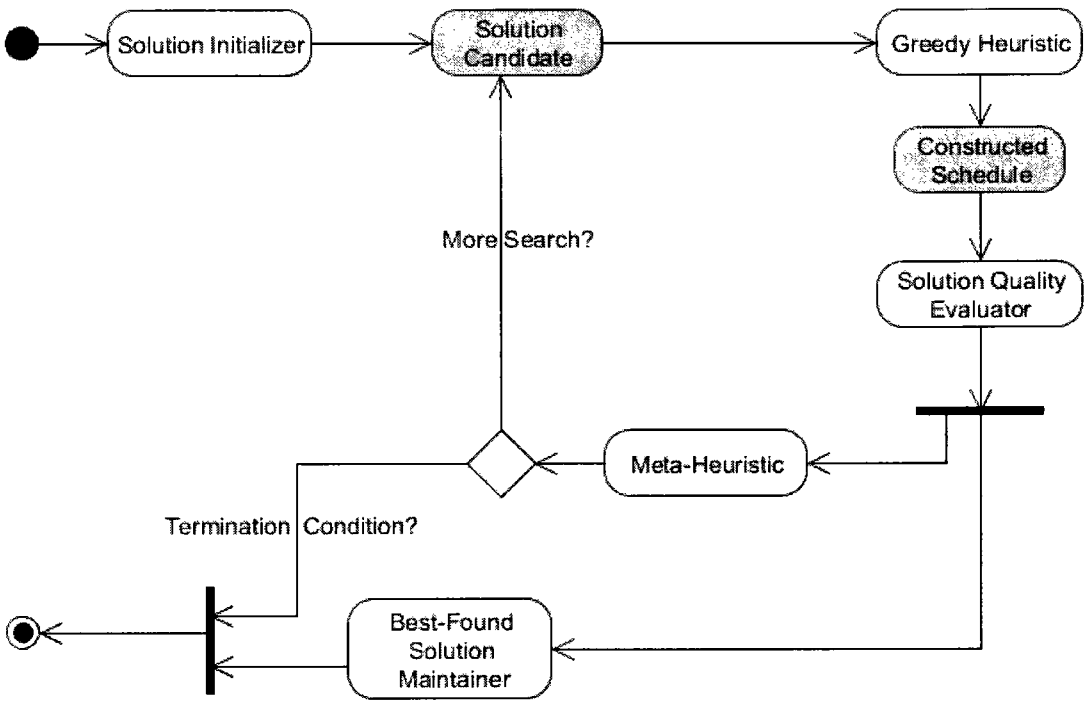

Figure 5: A framework for hybrid spatial scheduling algorithms

In this framework, the white blocks represent components of the algorithm. The shaded blocks represent outputs of certain algorithm components that are passed as input to other components. The shaded "Candidate Solution" block is a candidate solution in the *appropriate input format* required by the heuristic algorithm. The shaded "Constructed Schedule" block is a full schedule constructed by the greedy heuristic. Candidate solutions are represented as inputs into the greedy heuristic, and the greedy heuristic constructs feasible schedules from inputs it receives from the meta-heuristic or initializer. Schedules are evaluated by the solution quality evaluator. Based on feedback by the solution quality evaluator, the meta-heuristic searches for optimal solutions by improving these greedy heuristic inputs until a termination condition occurs.

The individual components are described in more detail as follows:

- **The Solution Initializer:** This component begins the algorithm process by constructing the initial candidate solution. The output of this component is an initial solution in the *input format required* by the Greedy Algorithm.
- **The Greedy Heuristic:** This component implements some kind of greedy method for constructing a feasible schedule based on the jobs and possibly other input parameters. Although a wide variety of approaches are possible, this component should construct solutions without search. Its output is a feasible schedule that goes to the Solution Quality Evaluator.
- **The Solution Quality Evaluator:** This component evaluates schedules from the Greedy Algorithm with respect to a particular objective function. It has two outputs: 1) the objective function value of its input schedule,

and 2) its input schedule. Its objective function evaluation is input for both the Meta-Heuristic and the Best-Found Solution Maintainer, and its input schedule is also input for the Best-Found Solution Evaluator.

- **The Meta-Heuristic:** This component searches through the space of *Greedy Heuristic Inputs* to find the most optimal solution possible. In order to guide its search, it takes as input evaluations of the solution candidates it generates from the Solution Quality Evaluator component. The Meta-Heuristic component also makes the decision after each schedule evaluation (i.e. each algorithm iteration) as to whether or not the algorithm should terminate. If the algorithm should not terminate, the Meta-Heuristic generates as output a new input or set of inputs for the Greedy Heuristic component.
- **The Best-Found Solution Maintainer:** The objective is always to find the best possible solution. Many meta-heuristics operate on a probabilistic basis and may not necessarily terminate on the absolute best solution encountered. The Best-Found Solution Maintainer component simply ensures that the best solution is always kept. It takes as input schedule combined with an evaluation of that schedule, both from the Solution Quality Evaluator. If the schedule is better than the schedule its best-so-far schedule, it simply replaces its best-so-far with this schedule.

This framework can provide a common structure for many types of spatial scheduling approximation algorithms. It is designed to make research into such problems more effective by enabling the interchangeability and interoperability of different meta-

heuristics, greedy algorithms, and objective functions. This allows for easier experimentation and incremental progress by focusing being able to focus on single components at a time.

8.1.A. Naming Convention

The modular nature of this framework lends itself to a convenient naming convention based on three core components: 1) the initial solution generator, 2) the meta-heuristic, and 3) the greedy heuristic. Consequently, the following naming convention will be used:

<name of initial solution generator> / <name of meta-heuristic> / <name of greedy heuristic>

In this notation, for example, an algorithm that initializes with BLTI-RR, uses a local search as a meta-heuristic, and creates schedules with BLTI would be named EDD-BLTI-RR/Local Search/BLTI.

8.1.B. Solution Representations and Formats

This framework pairs a constructive heuristic algorithm with a meta-heuristic. The heuristic is used to construct feasible schedules deterministically from an input list of jobs. The quality of the schedule is thus a function of the properties of the input list of jobs. The meta-heuristic is used to modify the input list of jobs and search for the input configuration that, when fed into the constructive heuristic, will produce the best quality schedule. Thus, there are essentially two basic solution formats that must be represented: 1) the heuristic input and 2) the constructed schedule. This is depicted in the figure below:

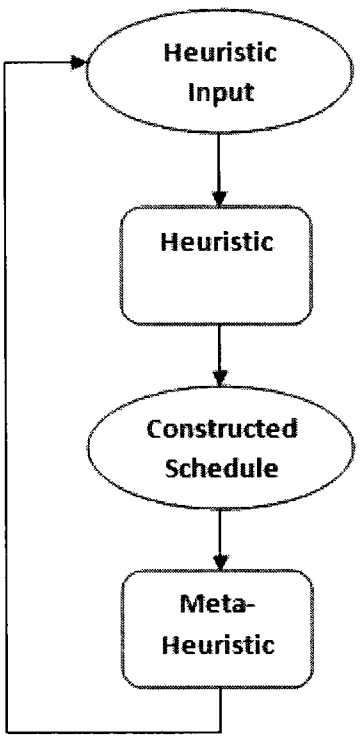

Figure 6: The two types of problem representations needed and their relationship to the hybrid algorithm components

There are many possible ways to represent the heuristic inputs and constructed schedules that can be incorporated into this framework. Based on the heuristics employed (BLTI-based heuristics) a single basic representation is used for all of the hybrid meta-heuristic algorithms developed in this dissertation.

1) Heuristic Input Representation: The heuristic input consists of an ordered list of elements, where each element contains the following attributes:

- Job: a job containing the following information:

- A unique job name (or number)
- Width
- Height
- Earliest Start Date
- Due Date
- Processing Time

- Assigned Area: The processing area to which the job is currently assigned

2) Constructed Schedule Representation: The constructed schedule contains a feasible schedule generated by the heuristic. Specifically, it provides the following information for each job:

- Name: The unique name or number of the corresponding job
- Start Time
- End Time
- Assigned Area
- X Coordinate
- Y Coordinate

8.1.C. Meta-Heuristic Mutator Operations and Associated Parameters

Meta-heuristic algorithms work by intelligently transforming one solution or set of solutions into another and intelligently searching for better-quality solutions until some termination condition occurs. In order to transform one solution into another, meta-heuristics rely on specific mutator operations for transforming solutions encoded in the chosen solution representation. Because a single solution representation scheme is used for all of the hybrid meta-heuristic algorithms developed, a single set of mutation operations will also be used.

In this hybrid meta-heuristic framework, the meta-heuristic algorithm searches through the space of heuristic inputs and seeks to find the input configuration producing the best schedule when fed into the heuristic algorithm. Thus, the meta-heuristic must transform and search through *heuristic inputs* as opposed to constructed schedules. As discussed above, a heuristic input is essentially an ordered list of jobs along with each job's assigned area. Consequently, two mutator operations have been designed to facilitate appropriate transformation: SWAP_JOBS and SWAP_AREAS.

1) SWAP_JOBS: The SWAP_JOBS operator works by randomly swapping the positions of two jobs. This mutator is thus used to transform one job ordering into another. In order to accomplish this, the mutator randomly chooses two jobs in an ordered job list and swaps their positions. However, swapping a job located near the front of the list and has a very early due date with a job near the end of the list with a late due date will not generally be beneficial. A better option is to swap jobs near each other so that the solution quality is not drastically changed in one mutation operation. In order to facilitate

this, a parameter called the ***SWAP RANGE*** is employed. The swap range is the maximum distance allowed for a single swap operation between two jobs in an ordered list. For example, if the swap range is 3, a job in position 1 could be swapped with jobs in positions 2, 3, or 4, but not 5 or higher.

2) SWAP_AREAS: The SWAP_AREAS operator works by randomly changing the assigned area of each job according to a specified probability. This probability takes the form of a user-specified parameter called ***AREA CHANGE PROBABILITY***. This operator is most easily describe in pseudocode, and is shown below.

```
Procedure SWAP_AREAS (Heuristic_Input_Elements)  DO:
    FOR EACH Element IN Heuristic_Input_Elements DO:
        R := Random Number in [0, 1]
        IF R <= AREA_CHANGE_PROBABILITY DO:
            New_Area := Different area than Element.Job is assigned to, and that
                                Element.Job fits within
            Element.Assigned_Job := New_Area IF New_Area exists
        END
    END FOR
END SWAP_AREAS
```

Table 22: The SWAP_AREAS operator

Thus, SWAP_JOBS and SWAP_AREAS perform the two types of transformations used to turn one solution into another in these hybrid algorithms. At each search step, a meta-

heuristic must mutate the candidate solution into another, but it need not have any knowledge of the inner workings of the mutator. Thus, for generality SWAP_JOBS and SWAP_AREAS are combined into a single generic MUTATE operation as shown below.

```
Procedure MUTATE (Heuristic_Input_Elements) DO:
    SWAP_JOBS(Heuristic_Input_Elements)
    SWAP_AREAS(Heuristic_Input_Elements)
END MUTATE
```

Table 23: The MUTATE operation

By combining the mutator operations together, the specific mutator operations need not be tightly tied to the meta-heuristics, making it possible to easily replace meta-heuristic components and obtain an entirely new algorithm. It also enables new mutator operations or even entirely new definitions of MUTATE to easily be incorporated in future research.

8.2. HYBRID ALGORITHM 1: EDD-BLTI-RR/LOCAL SEARCH/BLTI-EXTERNAL

The first hybrid algorithm obtains the initial solution using EDD-BLTI-RR and then combines a local search meta-heuristic with BLTI-External for schedule construction. Also, because the objective is minimizing total tardiness the solution quality evaluator will evaluate for this objective. The design for the EDD-BLTI-RR/Local Search/BLTI-External algorithm is shown in the figure below.

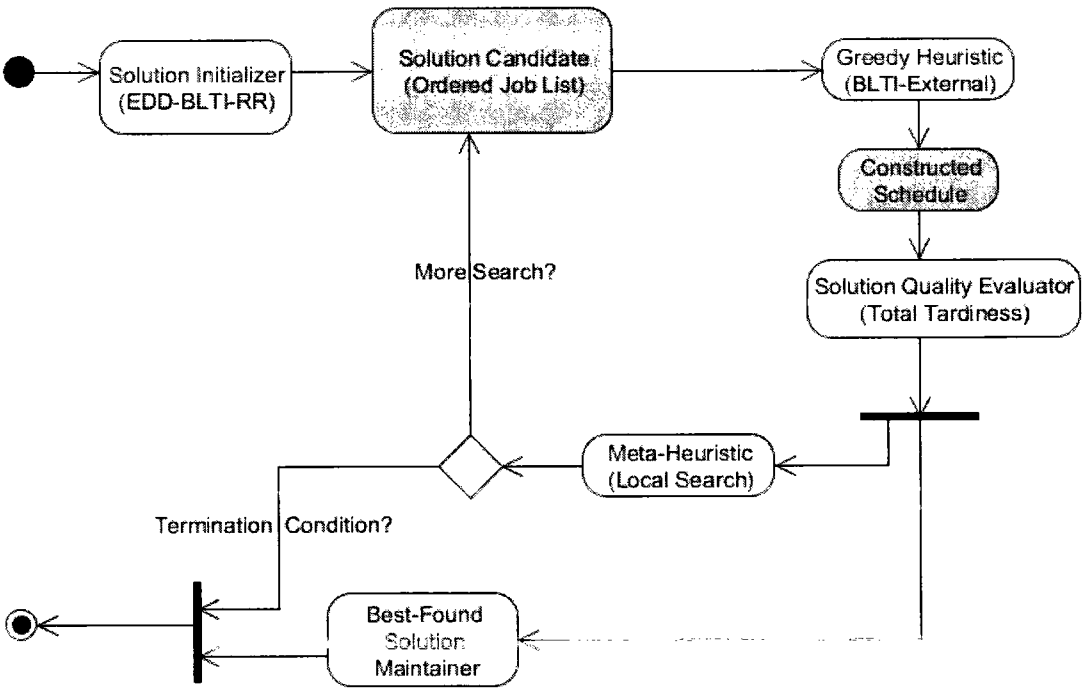

Figure 7: The EDD-BLTI-RR/Local Search/BLTI-External Algorithm Design

Local search is perhaps the simplest of all meta-heuristics and works by mutating candidate solutions, evaluating them, and accepting only changes that result in better quality solutions. The algorithm terminates once termination certain conditions have been reached. Three common termination conditions are 1) terminate when the maximum specified number of iterations are reached, 2) terminate when a maximum period of time has passed, or 3) terminate if a particular cost lower bound is reached. These are the three conditions utilized in this algorithm. Accordingly, there are three user-specified parameters associated with this meta-heuristic (in addition to the *swap range* and *area change probability* parameters needed for the mutator): ***MAX ITERATIONS***, ***MAX TIME,*** and ***LOWER BOUND***. The max time parameter is in seconds. The general local search algorithm is given below.

```
Procedure LOCAL_SEARCH (Initial_Solution)  DO:
    Current_Solution := Initial_Solution
   Iterations := 0
    WHILE Iterations < MAX_ITERATIONS AND MAX_TIME NOT EXCEEDED AND
     LOWER_BOUND NOT REACHED DO:
       Next_Solution := MUTATE(Current_Solution)
       Current_Solution := Next_Solution IF Next_Solution is better than
                                Current_Solution
       INCREMENT Iterations BY 1
    END WHILE
    RETURN Current_Solution
END LOCAL_SEARCH
```

Table 24: The LOCAL_SEARCH algorithm

The Solution Quality Evaluator component provides the objective function value for each candidate solution, and the current solution's objective value is compared to the next solution's objective value. If the next is better it simply replaces the current solution, and the process continues until termination.

This meta-heuristic can in some cases perform very well, but it is prone to converging on a local, rather than global, optima. The next algorithm design is intended to overcome this pitfall.

8.3. HYBRID ALGORITHM 2: EDD-BLTI-RR/SIMULATED ANNEALING/BLTI-EXTERNAL

Simulated annealing (SA) is a widely-used meta-heuristic originally developed by Kirkpatrick et al. [18] that simulates a molten metal being cooled in a controlled manner. At higher temperatures metal particles are in constant transition, and as a metal is cooled in a controlled manner its particles gradually settle into a state of low energy. When metals are quickly cooled they have the tendency to freeze in a high-energy state, with the resulting crystal structure containing many cracks and flaws. This is generally an undesirable property. In contrast, when cooled in a slow and controlled manner the particles settle into a regular crystal structure with few flaws, resulting in maximum strength. In SA, candidate solutions represent "metal states", and a simulated temperature is used to determine the probability of accepting a "higher energy" solution than the current solution at any given time. In SA, a good solution is analogous to a "low-energy" state, while a bad solution is analogous to a "high-energy" state. One state (i.e. solution) is transformed into another using a meaningful mutator operation. SA utilizes a "current temperature" variable that is gradually decreased. Lower-energy states are always accepted when encountered, but at any given time there is also a certain probability of accepting a higher-energy state. Specifically, this probability is a function of temperature. Thus, as the temperature is gradually decreased so is the probability of transitioning into a higher-energy state, until a stable temperature is reached. This allowing of higher-energy states to be occasionally accepted based a temperature-determined probability enables SA to avoid becoming trapped in local optima.

In the bin- and strip-packing literature SA has been very successful as the meta-heuristic component of hybrid heuristic/meta-heuristic algorithms similar to those prescribed by the framework above. Burke et al. [54] tried this approach with SA, genetic algorithms, and TABU search all combined with the bottom-left heuristic. Their best results were obtained using SA, providing good evidence that it will be an effective meta-heuristic for spatial scheduling given the clear similarities between spatial scheduling and multi-dimensional bin-packing. Additionally, this meta-heuristic is well-suited to spatial scheduling because in contrast to evolutionary methods, which require generating and evaluating *large populations* of solutions at a time, SA only needs to generate and evaluate a single solution at a time. The computational results for BLTI showed that in large problem instances it could take nearly 15 seconds to generate and evaluate a single solution. While it would take a prohibitively long time to iterate through a large number of populations, it is not prohibitive to do this for a large number of individual solutions. In light of these results a hybrid spatial scheduling algorithm combining SA with BLTI has been developed as part of this dissertation. This algorithm is an implementation of the framework developed in the previous section. The algorithm design is shown below as an adaptation of the general framework.

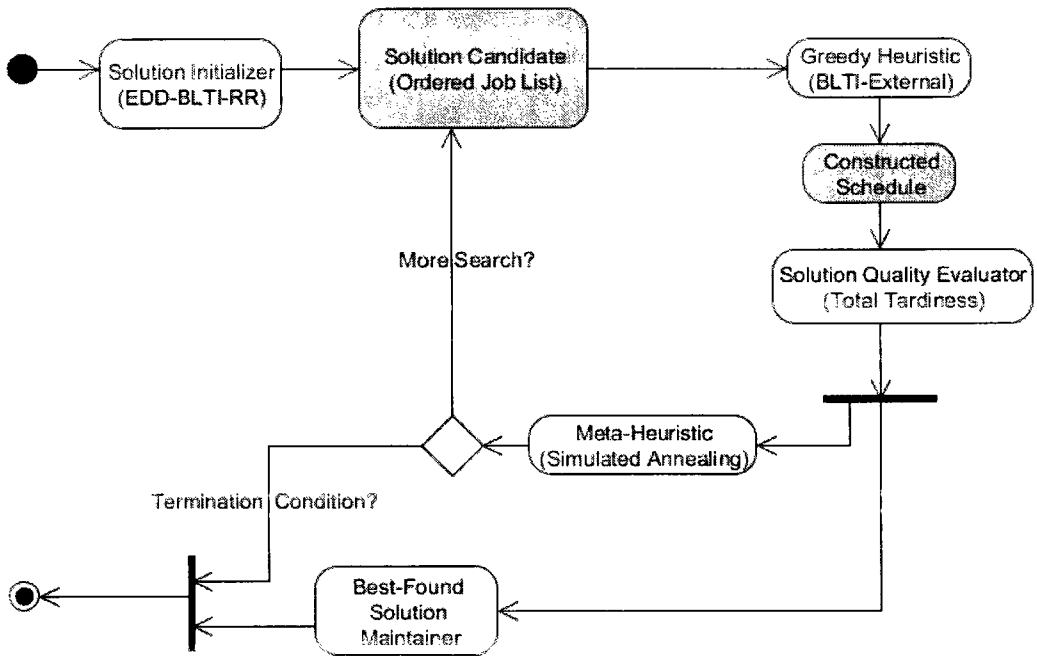

Figure 8: The EDD-BLTI-RR/Simulated Annealing/BLTI-External Algorithm Design

8.3.A. Basic Simulated Annealing

SA always accepts a better solution when it is encountered. However, SA will also accept a worse solution with a certain probability P, which is based on the current temperature T. SA relates P to T using the following equation, based on the Boltzmann distribution:

$$P = e^{\frac{-c_2 - c_1}{T}} \qquad (103)$$

In (103) c_1 designates the cost of the current solution (in this case, the total tardiness) and c_2 designates the cost of the potential next solution to transition into. The basic SA has two main loops, an outer loop and an inner loop. The outer loop performs temperature cooling, while the inner loop performs a set number of mutations and potential transitions

at each temperature. At the end of each execution of the outer loop the temperature is decreased exponentially by a cool rate multiplier $C \in (0,1)$ such that in the i^{th} iteration, $T = T(C^i)$.

In the basic SA implementation used in this research a number of parameters are required, some of which are related to the algorithm termination conditions while others are related to the mutator operations or algorithm iterations. These parameters are user-specified and are as follows:

Parameter	**Purpose**
Swap Range	The mutator swap range (discussed in preceding section)
Area Change Probability	The probability of a mutator switching the assigned processing area of a job (discussed in preceding section)
Max Time	The maximum number of seconds the algorithm is allowed to run before terminating
Max Iterations	The maximum amount of outer iterations permitted before terminating
Inner Iterations	The number of inner iterations to run within each outer iteration
Cool Rate	A number between 0 and 1 - the rate at which the temperature should be cooled at each outer iteration i. In the i^{th} iteration $T = T(C^i)$.
Initial Acceptance Probability	The initial probability (at the starting temperature) that a worse solution will be accepted.
Max Improvement Time	The maximum number of seconds that can pass without improvement before the algorithm terminates
Lower Bound	A lower bound on solution cost that if reached, will cause the algorithm to terminate.
Min Temperature	The stopping temperature – the algorithm terminates once this temperature becomes the current temperature.

Table 25: User-specified parameters required by the basic simulated annealing algorithm

The basic simulated annealing algorithm (adapted from [65]) may be described as follows:

```
Procedure BASIC_SA_OPTIMIZE (Initial_Solution)  DO:
    T := INITIAL_TEMP(Initial_Solution, INNER_ITERATIONS,
INITIAL_ACCEPTANCE_PROBABILITY)
    Current_Solution := Initial_Solution
    Best_Solution := Current_Solution
    Outer_Iterations := 0
    WHILE Termination Conditions Not Reached DO:
        FOR i in 1... INNER_ITERATIONS DO:
            Next_Solution := MUTATE(Current_Solution)
            IF COST(Next_Solution) < COST(Current_Solution) DO:
                Current_Solution := Next_Solution
                Best_Solution := Current_Solution IF COST(Current_Solution) <
                                        COST(Best_Solution)
            ELSE
                P := e^( -COST(Next_Solution) – COST(Current_Solution)) / T
                R := Uniform Random in [0, 1]
                Current_Solution := Next_Solution IF R <= P
            END IF-ELSE
        END FOR
        T := T * COOL_RATE
        Outer_Iterations := Outer_Iterations + 1
    END WHILE
    RETURN Best_Solution
END BASIC_SA_OPTIMIZE
```

Table 26: The basic simulated annealing algorithm

1) Initial Temperature Determination: Although there are a number of possible ways to determine the initial temperature, the approach used in this algorithm is to set the initial temperature in such a way that it corresponds to the user-specified initial acceptance rate of worse solutions. This is accomplished as follows. Let c_1 denote the cost of the initial solution S and let c_2 denote the cost of some neighboring solution MUTATE(S). The definition of a logarithm is as follows:

$$x = b^y \equiv \mathrm{y} = \log_b(x) \qquad (104)$$

Based on this definition, the corresponding starting temperature T can then be obtained mathematically starting with (103) as follows:

$$P = e^{\frac{-c_2 - c_1}{T}} \equiv \frac{-c_2 - c_1}{T} = \log_e(P) \equiv T = \frac{-c_2 - c_1}{\log_e(P)} \qquad (105)$$

Using (105) it is possible to compute a starting temperature given the initial acceptance probability by comparing the quality of a neighbor. However, the starting temperature can vary based on the cost of the neighbor. Thus, the average over the number of inner iterations is used in order to obtain a more certain initial temperature. The INITIAL_TEMP procedure called by the SA algorithm above is defined as follows:

```
Procedure INITIAL_TEMP (Initial_Solution, Iterations, P)  DO:
    T := 0
    FOR i in 1... Iterations DO:
        Neighbor := MUTATE(Initial_Solution)
        T := T + [( -COST(Neighbor) – COST(Initial_Solution) / ln(P)]
    END FOR
    RETURN T / Iterations
END INITIAL_TEMP
```

Table 27: The INITIAL_TEMP procedure

2) Termination Conditions: A number of user-specified parameters relate to termination conditions. Accordingly, the SA algorithm terminates when any of the following conditions occur:

1. The number of outer iterations exceed MAX ITERATIONS
2. The amount of time since the algorithm started exceeds MAX TIME
3. The cost of the current solution ≤ LOWER BOUND
4. The amount of time that has passed since an improvement was found exceeds MAX IMPROVEMENT TIME
5. Current temperature ≤ MIN TEMPERATURE

8.3.B. Simulated Annealing with Reheating and Restarting

The simulated annealing algorithm utilizes a temperature-based probability to occasionally permit a lower-quality solution to be accepted. This mechanism reduces the

likelihood that the algorithm settles into local optima. In the basic SA algorithm described above, the temperature is cooled exponentially in each outer iteration, resulting in a strictly decreasing temperature as time moves forward. However, more sophisticated cooling schedules are possible that may further reduce the likelihood of settling into a sub-optimal solution ([65], [66], [67]). One insight by Abramson et al. [65] is the notion of "phase changes" analogous to physical phase changes that occur in matter. They observed that at certain temperatures large solution improvements are more likely to be encountered. This research uses their insight as the basis for another, more sophisticated SA design.

SA always keeps track of the best solution found, regardless of the current solution. It is also quite easy to keep track of the temperature at which the best solution was found. By keeping track of these items it becomes possible to retain a memory not only of the best solution, but also the conditions under which it was produced. By having this stored in memory it becomes possible to return to this state if needed. This forms the basis for a variation on the basic SA: 1) keep track of both best solution and best temperature, 2) if the difference between current solution and best found exceeds a certain threshold, then 3) revert the current solution and temperature back to the best solution and temperature, respectively. Thus, this variation incorporates instantaneous reheating and restarting. In order to determine when to revert, this algorithm variation utilizes a user-specified parameter called ***MAX SOLUTION DEVIATION*** in addition to the other SA parameters described above. The max solution deviation is a value that represents the threshold as a proportional difference from the best solution. For example, if $\text{max solution deviation} = 1.4$ this means that if the current solution exceeds 140% of

the best solution then the algorithm should reheat to the best temperature and restart with the best solution. This modified version of simulated annealing is shown in Table 28. The same termination conditions used in the basic version of SA are also used in this algorithm as well. Moreover, the *MAX SOLUTION DEVIATION* parameter enables significant flexibility in the way this algorithm behaves. If *MAX SOLUTION DEVIATION* is set to be very large – such that the cost difference between the current solution and the best could never conceivably exceed this deviation – then reheating/restarting will never occur and this algorithm will behave exactly as the basic SA above. On the other hand, if *MAX SOLUTION DEVIATION* is set to 1 then only improvements are accepted. This behavior is equivalent to local search. As a result, this is the simulated annealing algorithm implemented for computational experimentation, and the basic SA as well as local search are employed by simply setting the *MAX SOLUTION DEVIATION* parameter accordingly.

```
Procedure REHEAT_BEST_SA_OPTIMIZE (Initial_Solution)  DO:
    T := INITIAL_TEMP(Initial_Solution, INNER_ITERATIONS,
                          INITIAL_ACCEPTANCE_PROBABILITY)
    Current_Solution := Initial_Solution
    Best_Solution := Current_Solution
    Best_Temp := T
    Outer_Iterations := 0
    WHILE Termination Conditions Not Reached DO:
        FOR i in 1... INNER_ITERATIONS DO:
            Next_Solution := MUTATE(Current_Solution)
            IF COST(Next_Solution) < COST(Current_Solution) DO:
                Current_Solution := Next_Solution
                IF COST(Current_Solution) < COST(Best_Solution) DO:
                    Best_Solution := Current_Solution
                    Best_Temp := T
                END IF
            ELSE
                P := e^( -COST(Next_Solution) – COST(Current_Solution)) / T
                R := Uniform Random in [0, 1]
                Current_Solution := Next_Solution IF R <= P
                IF Current_Solution ≥ MAX_SOLUTION_DEVIATION * Best_Solution
                    Current_Solution := Best_Solution
                    T := Best_Temp
                    i := INNER_ITERATIONS + 1 // Force inner loop to terminate
                END IF
            END IF-ELSE
        END FOR
        T := T * COOL_RATE
        Outer_Iterations := Outer_Iterations + 1
    END WHILE
    RETURN Best_Solution
END REHEAT_BEST_SA_OPTIMIZE
```

Table 28: The Reheat-Best simulated annealing algorithm

CHAPTER 9: COMPUTATIONAL EXPERIMENTS AND RESULTS

In order to test the performance of the algorithms developed in this dissertation, a large number of test problems with differing characteristics were generated. Specifically, three distinct problem generation algorithms were developed in order to facilitate the computational experimentation. These problem-generating algorithms will be discussed later in this chapter, and they are described in detail in Appendix 1. This chapter provides a detailed discussion of the computational experiments and the results. This entails the experiment design, the experimental results, and conclusions about the algorithm performances.

The algorithms developed in the preceding two chapters were implemented in the Ruby programming language [59] and were used in this experimentation. Specifically, all algorithms in Chapter 7 (heuristic algorithms) were implemented, and the EDD-BLTI-RR/Simulated Annealing/BLTI-External hybrid meta-heuristic algorithm from Chapter 8 was also implemented. The meta-heuristic component used in this algorithm was the Reheat-Best simulated annealing. This variant was chosen for implementation because it can behave as a local search only, as basic simulated annealing, and also as simulated annealing with reheating/restarting depending on how the parameters are set. This was discussed in Chapter 8. Additionally, the lower bound-calculating algorithm discussed in Chapter 6 was also implemented and included in the experiments.

9.1. EXPERIMENT DESIGN

The computational experiments were designed to test the performance of the algorithms developed in this dissertation on a wide range of problems across many

dimensions including 1) problem size, 2) degree of space limitation (tightness), and 3) qualitative differences. Because the hybrid meta-heuristic algorithms are parameter-driven, a number of different algorithm behaviors are possible depending on how certain parameters are set. As a result, different parameter settings may be viewed as treatments, and the combination of different treatments results in differently-behaving algorithms.

The experiment design is composed of five distinct parts: 1) Treatments and Combinations, 2) Problem Sets, and 3) Performance Measures, 4) Analytical Methodology, and 5) Computational Resources. The "Treatments and Combinations" section describes the different algorithms and parameter settings used. The "Problem Sets" section describes the different problem sets on which the algorithms were tested. The "Performance Measures" section describes the dimensions and measures of performance used to evaluate the algorithms. The "Analytical Methodology" section describes the approaches used for the data analysis and for drawing conclusions about algorithm performance. Finally, the "Computational Resources" section provides a specification of the computer on which the experiments were carried out.

9.1.A. Treatments and Combinations

A number of solution methods and algorithms were utilized in these experiments. Apart from the hybrid meta-heuristic algorithms, these include the following:

- Model 3 solved with CPLEX (CPLEX)
- Earliest-Due-Date Heuristic (EDD)
- Lower Bound (LB)

The implemented EDD-BLTI-RR/Simulated Annealing/BLTI-External hybrid meta-heuristic algorithm utilizes a number of user-specified parameters to determine its behavior. Because different parameter settings determine algorithm behavior (including behaving as local search, basic SA, or SA with reheating/restarting), certain parameter settings are treated as treatments. Consequently, individual parameter settings may be viewed as a treatment and each solution method may be viewed as a treatment combination. In this manner, one complete parameter set is designated as a control, while the other treatments are applied by modifying a particular parameter from the control to another value. The specific treatments used in these experiments are as follows:

- Local Search (Control)
- Basic Simulated Annealing
- High Area Swap Probability
- Reheat/Restart Simulated Annealing with Low Solution Deviance
    - This permits the search to accept modestly worse solutions but forces the algorithm to reheat and restart if the current solution wanders too far from the best found solution. Thus, this behaves like a local search with an additional (but limited) neighborhood exploration capability.
- Reheat/Restart Simulated Annealing with High Solution Deviance
    - This permits the search to deviate farther away from the best found solution before forcing to reheat/restart. This provides more freedom for neighborhood exploration, but without limitless ability to wander away from the best found solution.

Treatment	Parameter	Value	Comments
Local Search	Swap Range	3	
	Area Change Probability	0.05	
	Max Iterations	600	
	Max Time	-	Depends on problem size
	Min Temp	0.1	
	Lower Bound	0	
	Cool Rate	0.96	
	Inner Iterations	1	
	Max Solution Deviation	1	Accept only improvements
	Max Improvement Time	-	Depends on # Areas
	Initial Acceptance Prob.	0.85	
Basic SA	Max Solution Deviation	500	Make impossibly large so that no solution would ever cause a reheat/restart
High Area Swap	Area Change Probability	0.2	
SA-Low Deviation Reheat	Max Solution Deviation	1.4	
SA-High Deviation Reheat	Max Solution Deviation	2.6	

Table 29: Hybrid meta-heuristic algorithm parameter settings (treatments)

In the table above, Local Search is regarded as the control treatment, while other treatments modify a specific parameter from Local Search. For example, the Basic SA modifies the Max Solution Deviation parameter. It is clear that a number of these

treatments may be used in combination (for example, High Area Swap with Basic SA). Based on this principle, a number of treatment combinations were chosen to create the specific hybrid meta-heuristic algorithm variants used in the experiments. These are shown in the table below.

Combination Abbreviation	**Treatment 1**	**Treatment 2**
LS	Local Search	-
BSA	Basic SA	-
RSAL	SA-Low Deviation Reheat	-
RSAH	SA-High Deviation Reheat	-
LSS	High Area Swap Probability	Local Search
BSAS	High Area Swap Probability	Basic SA
RSALS	High Area Swap Probability	SA-Low Deviation Reheat
RSAHS	High Area Swap Probability	SA-High Deviation Reheat
CPLEX	Model 3 with CPLEX	-
EDD	Earliest-Due-Date-First /BLTI	-
LB	Lower Bound Algorithm	-

Table 30: Treatment combinations used in experiments

In addition to the combinations related to the hybrid meta-heuristic algorithms, the last three items in the table are included because they are used for certain problems as well. Thus, this table represents a comprehensive list of all methods used in experimentation.

1) Replications: In Table 30 above, it should be noted that the hybrid meta-heuristic algorithm treatments are stochastic in nature, while the CPLEX, EDD, and LB algorithms are deterministic. As a result, repeated trials are necessary for the hybrid meta-heuristic algorithm variants, but not for the deterministic methods. In order to test the consistency of each stochastic algorithm, ten runs per algorithm were used in each application of a stochastic algorithm. For example, for all test problems that BSA was applied to, it was run ten times on each problem. This replication allows for testing the consistency.

9.1.B. Problem Sets

Three sets of benchmarks problems were developed for this experimentation, each with distinct characteristics. Three different problem-generation algorithms (detailed in Appendix 1) were developed and used to create each of the three different problem sets. As a result, the sets are called A2, A3, and A4 correspond to the algorithm that generated them. Each of these problem generation algorithms utilizes a tightness factor ranging from 0 to 10 to generate a problem of a specified number of jobs. The tightness factor controls how much of a general processing space limitation there is – and consequently, how difficult the problem is. At higher tightness levels there is much less space available at any given time and as a result it is more difficult to find the optimal locations and times for each job. Conversely, at very low tightness levels each job is likely to be able to have the processing space to itself, with little competition for processing space resources.

The problem generation algorithms all create qualitatively different problems. Problems generated by the Algorithm 2 have their size slightly correlated with the tightness level. As tightness increases there is a limit on how little space a job can

consume. As a result, this algorithm creates bulkier jobs at higher tightness levels. Problems generated by the Algorithm 3 create jobs with evenly distributed sizes, and there is no correlation of job size to tightness. Neither Algorithm 2 nor Algorithm 3 correlates job size to job processing time. Algorithm 4 does precisely this. Algorithm 4 generates jobs with a perfect correlation of job size to job processing time – so the larger the job the more processing time it requires. There is no correlation between job size and job tightness in A4, however.

1. Problem Set Naming Convention: There are three major problem sets (A2, A3, and A4), and within these three sets there are problems with a number of characteristics. These include the number of processing areas used, the number of jobs, and the tightness level. In order to classify the problems, the following naming convention is used:

<Generating Algorithm> / <Number Processing Areas> / <Number Jobs> / <Tightness>

For example, A3/10/500/6 denotes a problem generated by Algorithm 3 having ten processing areas, 500 jobs, and a tightness level of 6.

2. Processing Area Configurations Used: A number of different processing area configurations were used for to generate the test problems. The number and dimensions of each configuration are shown below, along with the problem sets in which each configuration was included.

Configuration	Used in Sets	Area Widths	Area Heights
1	A2	15	8
2	A3, A4	12	9
3	A2	15 20 10	8 12 6
4	A3, A4	12 15 10 12 5 9 7 5 14 12	9 5 5 6 6 11 13 5 4 8

Table 31: Area configurations used in each problem set

3. Problem Set Combinations: Within each major set, test problems span over different numbers of processing areas, number of jobs, and tightness levels. Thus, each problem type is a specified by specific combination of these attributes. The problem sets are summarized in the table below.

Generation Algorithm	Numbers of Processing Areas	Numbers of Jobs	Tightness Levels	Instances of Each Combination
A2	1, 3	10, 50, 100	2, 4, 6	10
A3	1, 10	500	4, 6, 8	10
A4	1, 10	500	4, 6, 8	10

Table 32: Summary of test problems used in experiments

In this table, for instance, there are two different processing area configurations used for A2 (1 area and 3 areas) with each configuration having three different numbers of jobs used (10, 50, and 100), each of which has three different tightness levels (2, 4, and 6), with 10 problem instances for each of these combinations. Thus, the total number of problems in the A2 set is (2)(3)(3)(10) = 180. Similarly, the total number of problems in the A3 set is (2)(1)(3)(10) = 60, and the total number for the A4 set is also (2)(1)(3)(10) = 60.

In the problems sets, A2 consists of small and medium-sized problems (ranging from 10 to 100 jobs) at three different tightness levels (low=2, medium=4, and high=6), while A3 and A4 consist of large problems (500 jobs) at three different tightness levels (medium=4, high=6, very high=8). A2 preceded A3 and A4 in time, and was used to compare the algorithms to solutions obtained with Model 3 run in CPLEX. Only these smaller problems can be solved with CPLEX, consequently CPLEX is only applied to problems in A2. In many instances, the problems were not able to be solved with CPLEX

within the time limit of 10 minutes. In these cases, the incumbent solution found at 10 minutes was used in lieu of the optimal solution.

When the CPLEX and the hybrid meta-heuristic algorithm variants were applied to the problems, certain time limits were imposed based on problem size. Because the EDD and LB methods were deterministic and run in polynomial-time, no time limit needed to be imposed (these ran very quickly). For the hybrid meta-heuristic algorithm variants, two parameters control timing: MAX TIME, which limits the total amount of time permitted, and MAX IMPROVEMENT TIME, which terminates the algorithm if a certain time passes without finding an improvement. The time limits are shown below based on problem size and method.

Method	**Problem Size (Jobs)**	**Total Time Limit (MAX TIME)**	**MAX IMPROVEMENT TIME**
CPLEX	≤ 100	600 seconds	N/A
CPLEX	500	N/A	N/A
Hybrid Algorithms	≤ 100	180 seconds	40 seconds
Hybrid Algorithms	500	480 seconds	120 seconds

Table 33: Time limits for experimental methods based on problem size

9.1.C. Performance Measures

The algorithms developed in this dissertation are intended to provide solutions for spatial scheduling problems under the total tardiness objective. All solution algorithms

are approximation algorithms, and with the exception of the EDD heuristic all algorithms are stochastic as well. A good approximation algorithm is one that will find high-quality solutions quickly and consistently. As a result, three criteria are used to assess algorithm performance: 1) solution quality, required solution time, and algorithm variance.

These experiments involve a fairly large number of algorithm and problem set combinations, and each hybrid algorithm is run ten times on each individual problem. As is shown in Table 32 above, each problem combination contains ten problems. Because of the large number of problem combinations, this analysis compares the performance of each algorithm for each problem combination. Thus, the measures of performance used aggregated the results for all problems within a problem combination over all runs for each algorithm applied. For example, the A3/10/500/4 problem set contains ten problems, and each of the BSA, BSAS, LS, LSS, RSAL, RSALS, RSAH, and RSAHS algorithms were run ten times on each of these problems. The performance measures will compare each algorithm based on aggregate performances of over all runs on all problems within the A3/10/500/4 set.

The three performance measures are as follows:

1) Quality: The quality metric for a particular algorithm on a particular problem combination will be its average objective function over all runs over all problems in the combination set. For example, if BSA is used on A3/1/500/4, the quality rating is the average objective on all ten runs over all ten problems.

2) Time: The time metric used is similar to the quality, but instead of objective function values the average computational time is used.

3) Variance: Because each different test problem may have a different optimal value, the standard statistical variance over all runs over all problems within a combination set is not a meaningful measure of algorithm variance. In order to meaningfully measure algorithm variance, a measure called the Min-Max Delta Average was developed and is used. This method involves first taking the difference of the minimum and maximum algorithm's objective function values over all ten runs of a particular problem (called the min-max delta). Then, the min-max deltas for all problems in a combination set are simply averaged. For example, suppose BSA is run on the A3/10/500/6 problem combination set. Given that there are ten problems in this set, the Min-Max Delta Average, let $P_i = \{O_1, O_2, \ldots, O_{10}\}$ denote the objective function values obtained on BSA's ten runs on problem i for $i = 1 \ldots 10$. Then the Min-Max Delta Average is computed as follows:

$$MinMaxDeltaAvg(BSA) = \frac{1}{n}\sum_{i=1}^{n} \max(P_i) - \min(P_i) \qquad (106)$$

Here, $n = 10$ and denotes the number of problems in the combination set.

This metric provides a meaningful measure of variance because as an algorithm becomes more consistent on a given problem set, the Min-Max Delta Average will decrease. In the case where all solutions obtained have exactly the same objective function values, it is clear that the Min-Max Delta Average will be zero.

9.1.D. Analytical Methodology

The analytical data in these experiments are centered on problem combination sets, where each combination set includes the generating algorithm, the number of

processing areas, the number of jobs, and the tightness level. For each problem combination, all applicable solution algorithms are utilized and their performance is assessed in terms of the three performance measures described in the preceding section: Quality, Time, and Variance. Based on the results over all problem combinations, two types of analysis will be carried out: 1) Best Solution Quality Analysis, and 2) Comparative Algorithm Performance Analysis. The Best Solution Quality Analysis will examine the quality of solutions obtained by the best-performing hybrid meta-heuristic algorithm for each problem combination. These will be compared to the CPLEX results (where applicable) and the EDD results. These comparisons will be summarized to draw an overall conclusion about how well the hybrid meta-heuristics generally perform in terms of solution quality.

The Comparative Algorithm Performance Analysis will score each hybrid meta-heuristic algorithm over several different problem characteristics. Scores will be obtained by counting the number of times an algorithm won or placed second and was within 5% of the winner for problems having the specified characteristic. Three dimensions of comparison are used for each category: solution quality, variance (i.e. algorithm consistency), and solution time required. By tallying the scores, conclusions will be drawn about which algorithms perform best and the pertinent circumstances for this performance.

9.1.E. Computational Resources

The computer used to perform the experiments on was a Dell computer running Windows 7 with a quad-core 2.5 GHz processor and 6 GB of RAM.

9.2. EXPERIMENTAL RESULTS

In the results that follow, a cells marked with (W) indicates that the corresponding hybrid algorithm (row) won in terms of the particular performance metric (column). A cell marked with (S) indicates that the corresponding hybrid algorithm came in second place *and* was within 5% of the winner. EDD, CPLEX, and LB are not counted in terms of winning but are shown for comparison. The * next to CPLEX indicates when some problem instances in the set could not finish within the ten-minute time limit, and in such cases the incumbent found at ten minutes was taken in place of the optimal. Cases where all hybrid algorithms reached the optimal solution were not marked with winners.

9.2.A. A2 Problem Set Results

Algorithm	Quality	Variance	Time
BSA	0	0	0
EDD	0	0	0.0204
LB	0	0	0.0016
LS	0	0	0
RSAH	0	0	0
RSAL	0	0	0
CPLEX	0	0	1

Table 34: Results for problem set A2/1/10/2

Algorithm	Quality	Variance	Time
BSA	0	0	0
EDD	0	0	0.0234
LB	0	0	0
LS	0	0	0
RSAH	0	0	0
RSAL	0	0	0
CPLEX	0	0	1

Table 35: Results for problem set A2/1/10/4

Algorithm	Quality	Variance	Time
BSA	110.36	51.9	1.08544
EDD	147.9	0	0.0172
LB	2.084167	0	0.0015
LS	95.19	16.2	(W) 0.38128
RSAH	101.89	44.7	1.49049
RSAL	(W) 90.28	(W) 14.1	2.57835
CPLEX	78.5	0	10.2

Table 36: Results for problem set A2/1/10/6

Algorithm	Quality	Variance	Time
BSA	0	0	0
EDD	0	0	0.1138
LB	0	0	0.0155
LS	0	0	0
RSAH	0	0	0
RSAL	0	0	0
CPLEX	0	0	1

Table 37: Results for problem set A2/1/50/2

Algorithm	Quality	Variance	Time
BSA	42.32	36	(W) 7.48739
EDD	55.9	0	0.1123
LB	0	0	0.0142
LS	(W) 18.32	(W) 16.3	14.3221
RSAH	27.47	30	19.44077
RSAL	(W) 18.32	(S) 16.4	18.19153
CPLEX	9.4	0	33.1

Table 38: Results for problem set A2/1/50/4

Algorithm	Quality	Variance	Time
BSA	3964.27	1259.6	(S) 28.7415
EDD	5947.5	0	0.1027
LB	10.365	0	0.0154
LS	(W) 3445	(W) 947.6	49.33272
RSAH	4000	1325.4	(W) 28.53357
RSAL	3953.61	1306.6	29.55301
CPLEX *	3074	0	>600

Table 39: Results for problem set A2/1/50/6

Algorithm	Quality	Variance	Time
BSA	5.05	(S) 4.1	(W) 2.05233
EDD	6.1	0	0.2246
LB	0	0	0.0593
LS	(W) 2.77	(W) 4	7.76911
RSAH	4.25	4.4	5.19138
RSAL	3.16	4.8	7.40691
CPLEX	0.1	0	16.4

Table 40: Results for problem set A2/1/100/2

Algorithm	Quality	Variance	Time
BSA	332.59	218.2	(W) 24.80731
EDD	564.1	0	0.2122
LB	0	0	0.0593
LS	(W) 164.04	(W) 96.6	61.7666
RSAH	313.51	231.3	31.46904
RSAL	238.03	156.5	55.13094
CPLEX *	96.3	0	>600

Table 41: Results for problem set A2/1/100/4

Algorithm	Quality	Variance	Time
BSA	17251.97	4115.7	69.98187
EDD	26291.4	0	0.2229
LB	11.51833	0	0.0484
LS	(W) 15728.53	(W) 3291.6	109.7983
RSAH	17307.78	4436.7	(W) 68.64963
RSAL	17168.13	3968.3	(S) 69.46582
CPLEX *	17071.1	0	>600

Table 42: Results for problem set A2/1/100/6

Algorithm	Quality	Variance	Time
BSA	0	0	0
BSAS	0	0	0
EDD	0	0	0.0139
LB	0	0	0
LS	0	0	0
LSS	0	0	0
RSAH	0	0	0
RSAHS	0	0	0
RSAL	0	0	0
RSALS	0	0	0
CPLEX	0	0	1

Table 43: Results for problem set A2/3/10/2

Algorithm	Quality	Variance	Time
BSA	0	0	0.07646
BSAS	0	0	0.01122
EDD	2	0	0.0188
LB	0	0	0.0016
LS	0	0	0.04365
LSS	0	0	0.00796
RSAH	0	0	0.04226
RSAHS	0	0	0.01233
RSAL	0	0	0.02965
RSALS	0	0	0.01217
CPLEX	0	0	1

Table 44: Results for problem set A2/3/10/4

Algorithm	Quality	Variance	Time
BSA	2.87	8.4	1.22944
BSAS	1.91	5.3	(W) 0.86826
EDD	29.6	0	0.0171
LB	0	0	0.0016
LS	0.95	1.3	1.8668
LSS	0.71	1.7	1.21991
RSAH	1.16	3.2	2.19516
RSAHS	(W) 0.58	(W) 0.6	1.17122
RSAL	0.93	1.6	2.21848
RSALS	0.85	1.7	1.36578
CPLEX	0.5	0	1

Table 45: Results for problem set A2/3/10/6

Algorithm	Quality	Variance	Time
BSA	0	0	0.26816
BSAS	0	0	0.09578
EDD	1.7	0	0.145
LB	0	0	0.0139
LS	0	0	0.40501
LSS	0	0	0.0922
RSAH	0	0	0.26285
RSAHS	0	0	0.08768
RSAL	0	0	0.38685
RSALS	0	0	0.10327
CPLEX	0	0	10.6

Table 46: Results for problem set A2/3/50/2

Algorithm	Quality	Variance	Time
BSA	12.57	18.5	7.0239
BSAS	8.5	13.1	(W) 4.6752
EDD	104	0	0.1717
LB	0	0	0.0156
LS	5.91	5.1	13.60511
LSS	(S) 5.2	(W) 4	8.13945
RSAH	8.44	14.2	14.94082
RSAHS	5.57	5.3	8.55212
RSAL	7.15	9.6	11.26042
RSALS	(W) 4.95	5.2	6.67088
CPLEX *	2.9	0	208.6

Table 47: Results for problem set A2/3/50/4

Algorithm	Quality	Variance	Time
BSA	774.48	347.1	20.73854
BSAS	740.17	260.1	(W) 18.63512
EDD	1370.4	0	0.1622
LB	2.085	0	0.0141
LS	692.16	268	28.5627
LSS	(S) 683.52	(W) 231.6	31.02349
RSAH	749.59	343.3	24.85256
RSAHS	706.31	271.4	25.47106
RSAL	708.26	256.4	28.4284
RSALS	(W) 676.7	250.2	28.5191
CPLEX *	284.5	0	>600

Table 48: Results for problem set A2/3/50/6

Algorithm	Quality	Variance	Time
BSA	(W) 0	(W) 0	4.47515
BSAS	0.05	0.5	0.57325
EDD	8.5	0	0.3433
LB	0	0	0.0578
LS	(W) 0	(W) 0	3.32798
LSS	(W) 0	(W) 0	(S) 0.8549
RSAH	(W) 0	(W) 0	3.63693
RSAHS	(W) 0	(W) 0	0.84456
RSAL	0.01	0.1	2.22051
RSALS	(W) 0	(W) 0	(W) 0.83509
CPLEX	0	0	168.2

Table 49: Results for problem set A2/3/100/2

Algorithm	Quality	Variance	Time
BSA	61.58	73.7	(S) 22.96493
BSAS	48.12	51.4	(W) 21.81195
EDD	536.6	0	0.3464
LB	0	0	0.0546
LS	45.77	69.5	30.77508
LSS	(S) 37.08	(W) 36.1	29.4806
RSAH	52.29	84.3	26.94921
RSAHS	38.64	(S) 36.8	33.19781
RSAL	48.24	66.5	31.49741
RSALS	(W) 36.58	(S) 36.8	29.14601
CPLEX *	11.11	0	>600

Table 50: Results for problem set A2/3/100/4

Algorithm	Quality	Variance	Time
BSA	4797.9	1438.9	(W) 26.89427
BSAS	(W) 4639.86	1099.8	31.00228
EDD	6866.9	0	0.3526
LB	2.395	0	0.0484
LS	4770.58	1206.2	30.29597
LSS	(S) 4668.49	(S) 993.3	32.02694
RSAH	4845.33	1106.5	30.85922
RSAHS	4706.21	1030	29.59528
RSAL	4774.12	1264.9	29.83365
RSALS	4669.82	(W) 953.2	29.80651
CPLEX *	** N/A	**N/A	**N/A

Table 51: Results for problem set A2/3/100/6

** Indicates no feasible solution could be found for any problems in the set within the ten-minute time limit.

9.2.B. A3 Problem Set Results

Algorithm	Quality	Variance	Time
BSA	433.25	(S) 196.3	(W) 36.29984
EDD	489.2	0	0.9001
LB	0	0	1.5794
LS	(W) 292.94	260.4	221.9161
RSAH	416.08	(W) 195.6	(S) 39.50159
RSAL	407.59	223.3	84.71218

Table 52: Results for problem set A3/1/500/4

Algorithm	Quality	Variance	Time
BSA	35248.56	21252.2	373.4832
EDD	97752.9	0	0.911
LB	0	0	1.6974
LS	(W) 27703.46	(W) 16075.3	470.754
RSAH	36333.23	27433.7	(S) 369.4155
RSAL	36134.54	30322.7	(W) 363.0544

Table 53: Results for problem set A3/1/500/6

Algorithm	Quality	Variance	Time
BSA	776830.3	73017.5	(W) 405.3713
EDD	884517.2	0	0.8958
LB	1540.967	0	1.5644
LS	(W) 763812.9	(W) 41443.6	472.9361
RSAH	777539.1	61043.2	(S) 409.9439
RSAL	(S) 775331.2	51497.1	418.1327

Table 54: Results for problem set A3/1/500/8

Algorithm	Quality	Variance	Time
BSA	302.23	281.8	86.62901
BSAS	271.58	195.8	(W) 77.95707
EDD	1895.9	0	0.5491
LB	0	0	1.2394
LS	292.24	320.7	87.21973
LSS	(W) 247.39	221.2	(S) 83.31195
RSAH	296.06	274.9	98.3677
RSAHS	250.72	(W) 186.9	85.08191
RSAL	264.05	216	98.20824
RSALS	(S) 248.07	(S) 190.9	99.1589

Table 55: Results for problem set A3/10/500/4

Algorithm	Quality	Variance	Time
BSA	14386.51	3217.1	84.79285
BSAS	(S) 13918.87	3840.3	(W) 73.95548
EDD	24211.8	0	0.5477
LB	0	0	0.9283
LS	14215.2	3647.1	96.45603
LSS	13931.26	(W) 3067.7	91.37448
RSAH	14581.91	4305.9	78.75917
RSAHS	13947.35	3136.6	79.28835
RSAL	14358.85	4002.4	90.26367
RSALS	(W) 13865.68	3115.1	95.581

Table 56: Results for problem set A3/10/500/6

Algorithm	Quality	Variance	Time
BSA	88475.37	4831.9	94.5521
BSAS	87752.61	4675.4	91.10116
EDD	100324.1	0	0.5522
LB	150.2083	0	0.5158
LS	88107.82	4421.7	107.3271
LSS	(W) 87630.1	4552.7	98.41894
RSAH	88387.85	4636.8	95.47611
RSAHS	(S) 87636.26	4801.8	(W) 87.55033
RSAL	88167.35	4504.6	92.04264
RSALS	87785.83	(W) 4169.4	100.1691

Table 57: Results for problem set A3/10/500/8

9.2.C. A4 Problem Set Results

Algorithm	Quality	Variance	Time
BSA	849033.7	87691.9	(S) 413.4883
EDD	959508.9	0	0.916
LB	907.3019	0	1.7157
LS	(W) 829120.8	(W) 55277.3	475.06
RSAH	850463	75546.6	414.1899
RSAL	(S) 844013.6	84874.4	(W) 410.7197

Table 58: Results for problem set A4/1/500/4

Algorithm	Quality	Variance	Time
BSA	(S) 1578900	84688.4	433.6951
EDD	1705902	0	0.9251
LB	3202.327	0	1.6851
LS	(W) 1563916	(W) 53989.1	475.5586
RSAH	1583208	87459.3	(S) 429.4015
RSAL	1581623	73352.7	(W) 417.515

Table 59: Results for problem set A4/1/500/6

Algorithm	Quality	Variance	Time
BSA	(S) 2248969	71877.6	458.2324
EDD	2389030	0	0.9297
LB	5895.187	0	1.5928
LS	(W) 2240755	(W) 61347.1	475.7197
RSAH	2259811	101370.6	(W) 435.5452
RSAL	2254627	90554.9	(S) 446.6047

Table 60: Results for problem set A4/1/500/8

Algorithm	Quality	Variance	Time
BSA	49929.28	7288.1	85.48364
BSAS	48922.86	6792.2	107.4652
EDD	66892.9	0	0.568
LB	0	0	1.2384
LS	49819.81	7141.2	100.6848
LSS	48620.4	6199.4	88.03727
RSAH	49644.03	7862.7	87.45657
RSAHS	(S) 48464.61	(W) 5642.6	89.52469
RSAL	50221.84	6868.6	88.79947
RSALS	(W) 48402.51	6032.8	(W) 77.85881

Table 61: Results for problem set A4/10/500/4

Algorithm	Quality	Variance	Time
BSA	87304.18	7201.8	95.17158
BSAS	86433.04	6071.8	91.93623
EDD	102245.8	0	0.5724
LB	105.4749	0	0.9982
LS	87893.88	7461.1	103.659
LSS	(W) 86343.02	(W) 5777.4	90.72987
RSAH	87891.39	7728.1	87.24683
RSAHS	(S) 86380.18	7185.8	87.13509
RSAL	88162.12	7893.9	(S) 83.75141
RSALS	86677.39	6223.1	(W) 82.83173

Table 62: Results for problem set A4/10/500/6

Algorithm	Quality	Variance	Time
BSA	115288.3	6078.6	(S) 91.1078
BSAS	(S) 114412.9	(S) 5271.8	107.1338
EDD	128202.3	0	0.5881
LB	370.6453	0	0.5519
LS	115232.7	6514.3	99.22911
LSS	114583.4	4938.2	99.61771
RSAH	115609.8	6852.5	92.57541
RSAHS	114523.7	(W) 5023.9	83.44613
RSAL	115701.8	6137.3	93.11543
RSALS	(W) 114347.8	5702.5	(W) 88.75499

Table 63: Results for problem set A4/10/500/8

9.3. ANALYSIS OF HYBRID ALGORITHM PERFORMANCE

In this section, the overall performance of the hybrid meta-heuristic algorithms is analyzed. The first branch of analysis is focused on the how the best meta-heuristic hybrids performed in terms of quality compared to CPLEX, the calculated lower bound, and the EDD solution. The second part examines the performances of the different hybrid algorithm variants to one another over quality, variance, and time. The performances are compared with regard to different problem characteristics and recommendations are made accordingly.

9.3A. Best Solution Quality Analysis

In this section, the winning solutions of each problem set are compared to the lower bound, the solutions obtained the EDD heuristic, and the solutions obtained by CPLEX (where applicable).

The following acronyms are used:

- BMH = winning hybrid meta-heuristic solution quality
- LBCP = lower bound
- CPLEX = obtained solution quality
- MCP = BMH / CPLEX
- MEP = BMH / EDD solution quality
- CEP = CPLEX solution quality / EDD solution quality.

Set	LB	CPLEX	BMH	EDD	LBCP	MCP	MEP	CEP
A2-1-10-2	0	0	0	0	N/A	N/A	N/A	N/A
A2-1-10-4	0	0	0	0	N/A	N/A	N/A	N/A
A2-1-10-6	2.084	78.5	90.28	147.9	0.026548	1.150064	0.610412	0.530764
A2-1-50-2	0	0	0	0	N/A	N/A	N/A	N/A
A2-1-50-4	0	9.4	18.32	55.9	0	1.948936	0.327728	0.168157
A2-1-50-6	10.366	3074	3445	5947.5	0.003372	1.12069	0.579235	0.516856
A2-1-100-2	0	0.1	2.77	6.1	0	27.7	0.454098	0.016393
A2-1-100-4	0	96.3	164.04	564.1	0	1.703427	0.2908	0.170714
A2-1-100-6	11.52	17071.1	15728.53	26291.4	0.000675	0.921354	0.598239	0.649304
A2-3-10-2	0	0	0	0	N/A	N/A	N/A	N/A
A2-3-10-4	0	0	0	0	N/A	N/A	N/A	N/A
A2-3-10-6	0	0.5	0.58	29.6	0	1.16	0.019595	0.016892
A2-3-50-2	0	0	0	1.7	N/A	N/A	0	0
A2-3-50-4	0	2.9	4.95	104	0	1.706897	0.047596	0.027885
A2-3-50-6	2.085	284.5	676.7	1370.4	0.007329	2.378559	0.493797	0.207604
A2-3-100-2	0	0	0	8.5	N/A	N/A	0	0
A2-3-100-4	0	11.11	36.58	536.6	0	3.292529	0.06817	0.020704
A2-3-100-6	2.395	N/A	4639.86	6866.9	N/A	N/A	0.675685	0

Table 64: A2 problem set comparisons

Set	LB	BMH	EDD	MEP
A3-1-500-4	0	292.94	489.2	0.598814
A3-1-500-6	0	27703.64	97752.9	0.283405
A3-1-500-8	1540.97	763812.9	884517.2	0.863537
A3-10-500-4	0	247.39	1895.9	0.130487
A3-10-500-6	0	13865.68	24211.8	0.572683
A3-10-500-8	150.21	87630.1	100324.1	0.87347

Table 65: A3 problem set comparisons

Set	LB	BMH	EDD	MEP
A4-1-500-4	907.3	829120.8	959508.9	0.86411
A4-1-500-6	3202.37	1563916	1705902	0.916768
A4-1-500-8	5895.19	2240755	2389030	0.937935
A4-10-500-4	0	48402.51	66829.1	0.724273
A4-10-500-6	105.47	86343.01	102245.8	0.844465
A4-10-500-8	370.65	114347.8	128202.3	0.891933

Table 66: A4 problem set comparisons

Within the A2 problems, values of N/A occur for 1) algorithms with a high area swap probability in relation to single-area problems (this makes no difference in such cases), 2) when CPLEX solutions could not be obtained, and 3) where a division by zero occurs in calculation of values. CPLEX solutions were available for most problem sets. In many of these problem sets the optimal solutions were found for all problems within

the set. It may be observed that the calculated lower bound (LB) is significantly smaller than the CPLEX solutions in all of the non-zero objective function value cases. As a result, it is not expected that the best hybrid meta-heuristic solutions (BMH) come near the calculated LB (either within the A2 set or others). As can be seen, the BMH / CPLEX ratios (the MCP value) show that the hybrid algorithms perform well compared to CPLEX, most often coming within 1.7 times the optimal/CPLEX solution objectives. In some problem sets it is observed that the hybrid algorithms beat CPLEX as well. When compared to the EDD heuristic, it is observed that in all cases the hybrid algorithms far outperformed the EDD heuristic and in most cases gave objective function values only a small fraction of the EDD. The MEP and CEP values represent the ratios of the best hybrid solutions to EDD solutions and CPLEX solutions to EDD solutions, respectively. It is may be observed that these ratios are very close together in almost all cases, and that the solutions obtained by the hybrid algorithms are much closer to the CPLEX solutions than they are to the EDD heuristic (and in some cases the hybrid algorithms even beat CPLEX). It thus appears the hybrid meta-heuristic algorithms performed well in this problem set.

The A3 problems were all much larger than any A2 problems and were thus too large to be applied to for CPLEX. These problems were much larger and also somewhat different from the A2 jobs in that the job sizes were more uniformly distributed at higher tightness levels. Additionally, this set included problems of extremely high tightness levels (level 8). At tightness levels 4 and 6, the MEP values ranged from 0.13 to 0.59, indicating that the hybrid algorithms far outperformed the EDD heuristic at these tightness levels. At tightness = 8 level, the MEP went up to the 0.86-0.87 range.

The A4 problems were identical to the A3 problems in size and tightness levels. However, the A4 problems correlated job processing time to job size. As can be observed by the relative EDD performances, these problems tended to be significantly tighter than their A3 counterparts at any given tightness level (in other words, an A4 problem with a tightness of 6 is tighter than an A3 problem of tightness 6). This is particularly indicated by the LB, which is far higher in these problems than in all the other problem sets. The MEP ranges from 0.72 to 0.92 in the A4 problems, indicating that the hybrid algorithms outperformed the EDD, but not as significantly as they did for the A2 and A3 problems. Because the calculated lower bounds are so much higher in these problems, it is probable that the optimal solutions are far closer to the EDD solutions in these problems than in others. Under these circumstances there would be no reason to imagine that the hybrid meta-heuristic algorithms perform generally worse on this type of problem than on others.

9.3.B. Comparative Algorithm Performance Analysis

In this section the different hybrid meta-heuristic algorithm variants are compared to one another with respect to performance in three dimensions: solution quality, variance, and time. Within each dimension, the algorithms are compared across differing problem characteristics including 1) single processing area versus multiple processing areas, 2) different tightness levels, and 3) problem type (i.e. A2, A3, and A4). With respect to each criterion/problem characteristic an algorithm's score is the number of times that algorithm either won or came in second place within 5% of the winner for that

criterion/problem characteristic. For instance, if BSA won twice and came in second once (within 5% of winner) for the single area criteria, its score would be 3 for this criterion. In the tables below, the top scores in each category are marked with (W) and the second-place scores are marked with (S). However, this denotation is used only where two or fewer algorithms win or come in second.

In each table it should be noted that in single area problems the algorithm varieties that employ a high area swap probability (e.g. BSAS, LSS, RSAHS, and RSALS) were not applied, since this treatment will not make any difference in single-area cases.

Criteria	BSA	BSAS	LS	LSS	RSAH	RSAHS	RSAL	RSALS
Single Area	2	0	(W) 11	0	0	0	(S) 4	0
Multiple Areas	1	2	1	(W) 8	1	(S) 5	0	(W) 8
Tightness = 2	1	0	(W) 2	1	1	1	0	1
Tightness = 4	0	0	(W) 4	(S) 3	0	1	(S) 3	(W) 4
Tightness = 6	1	1	(W) 4	(S) 3	0	2	0	2
Tightness = 8	1	1	(W) 2	1	0	1	1	1
A2	1	1	(W) 6	(S) 5	1	2	2	4
A3	0	0	(W) 3	(S) 2	0	1	1	(W) 2
A4	2	1	(W) 3	1	0	2	1	2

Table 67: Quality score for each algorithm over each criterion/problem characteristic

The superior performance of algorithm variants that use local search as the meta-heuristic is clearly seen in the table above (LS and LSS). Perhaps the most important dimension of difference in problem varieties is the distinction between problems that have only one processing area versus those that have multiple processing areas. In the latter case the decision must be made as to which processing area each job should be assigned to. Interestingly, the local search-based algorithms won in both problem categories along this dimension. For the multiple-area problems, the algorithm varieties with the high area swap probabilities (i.e. those ending with S, especially LSS, RSALS, and RSAHS) far outperformed those that did not have this feature. In addition to the local search varieties, algorithms that used simulated annealing with reheating also performed well, and matched the local search in many instances. In particular, the algorithms that used simulated annealing with reheating either matched local search or came in second on both sides of the single area-multiple area problem dimension.

Criteria	BSA	BSAS	LS	LSS	RSAH	RSAHS	RSAL	RSALS
Single Area	(S) 2	0	(W) 10	0	1	0	(S) 2	0
Multiple Areas	1	0	1	(W) 7	1	(S) 6	0	5
Tightness = 2	(W) 2	0	(W) 2	1	1	1	0	1
Tightness = 4	1	0	(W) 3	(S) 2	1	(W) 3	1	(S) 2
Tightness = 6	0	0	(W) 4	(W) 4	0	1	1	1
Tightness = 8	0	0	(W) 2	0	0	1	0	1
A2	2	0	(W) 6	(S) 5	1	3	2	3
A3	1	0	(W) 2	1	1	1	0	(S) 2
A4	0	0	(W) 3	1	0	(S) 2	0	0

Table 68: Variance score for each algorithm over each criterion/problem characteristic

The local search algorithm variants (LS and LSS) clearly show superior performance with regard to variance as well, as can be observed in the table above. Interestingly enough, both the single-area and multiple-area categories were won by local search-based algorithms. Furthermore, algorithm varieties that used the high area swap probability also far outperformed those that did not on the multiple-area problems. Although not as good as a local search, simulated annealing with reheating appears to perform fairly well in terms of consistency. Interestingly, the variance results appear to

follow a pattern quite similar to solution quality results and appear to have quite a strong correlation.

Criteria	BSA	BSAS	LS	LSS	RSAH	RSAHS	RSAL	RSALS
Single Area	6	0	1	0	(W) 7	0	(S) 5	0
Multiple Areas	3	(W) 6	0	2	0	1	1	(S) 4
Tightness = 2	1	0	0	1	0	0	0	1
Tightness = 4	(W) 5	(S) 3	0	1	1	0	1	1
Tightness = 6	1	(S) 3	1	0	(W) 4	0	(W) 4	1
Tightness = 8	(W) 2	0	0	0	(W) 2	1	1	1
A2	(W) 5	(S) 4	1	1	2	0	1	1
A3	(S) 2	(S) 2	0	1	(W) 3	1	1	0
A4	2	0	0	0	2	0	(W) 4	(S) 3

Table 69: Time score for each algorithm over each criterion/problem characteristic

In contrast to the solution quality and variance, it appears that the basic simulated annealing variants performed the best in terms of computational time. Furthermore, simulated annealing with reheating came in second place while local search did not even win once. Thus, the computational time appears to be inversely correlated with quality

and variance. However, the high area swap probability was a part of both the winner and second place solution in the multiple-area category.

9.4. SUMMARY OF RESULTS

In this chapter, a computational experimentation was carried out on a large number of problems to investigate the performance of several varieties of hybrid meta-heuristic spatial scheduling algorithms. It appears that in general the hybrid meta-heuristic algorithms are effective in finding near-optimal solutions to spatial scheduling problems over a wide variety of differing characteristics. Where optimal solutions could be found, it was seen that these algorithms tend to find solutions that are far closer to the optimal solution than to the solutions found by the EDD heuristic algorithm. Hybrid algorithms that used local search as the meta-heuristic (LS and LSS) were found to be the most effective and provided the best solutions in nearly all problem categories. Furthermore, in problems involving multiple processing areas the high area swap probability led to superior performance in terms of quality, consistency (variance), and time. Algorithms that utilized simulated annealing with reheating also generally performed well and matched the local search-based algorithm performance on many problem sets.

In summary, the LS algorithm won in nearly all single-area problems, while the LSS (local search with high area swap probability) won in nearly all multiple-area varieties both in terms of solution quality and in terms of minimal variance. Consequently, because the LSS will behave as the LS whenever there is only a single processing area, this algorithm appears to give the best overall performance.

CHAPTER 10: CONCLUSIONS

In this dissertation a number of important contributions were made to field of spatial scheduling. From the literature review it was seen that there is relatively little research done in spatial scheduling compared to other types of scheduling. Furthermore, most of the research was conducted from an Expert Systems point of view as opposed to an Operations Research perspective. In this dissertation, spatial scheduling as a problem class was addressed mathematically and systematically from an Operations Research perspective. General optimization models for several spatial scheduling problem classes were developed and it was shown that these models contain a set of core constraints that apply (and can thus be transferred) to almost any spatial scheduling problem regardless of objective function. The complexity of the spatial scheduling models was addressed and it was proven that the models are at least NP-complete. Furthermore, spatial scheduling problems were shown to be NP-hard in general. In addition to complexity, a method for computing a lower bound for the total tardiness objective was developed. Furthermore, this lower bound was shown to be non-trivial, and a polynomial-time algorithm for computing this lower bound was also given.

In addition to theoretical aspects of spatial scheduling, a number of approximation algorithms for solving spatial scheduling problems were developed in this dissertation. Broadly speaking, these algorithms fell into two categories: greedy heuristic algorithms and hybrid meta-heuristic algorithms. The greedy heuristic algorithms were designed to quickly provide solutions with guaranteed feasibility. The hybrid meta-heuristic algorithms combine greedy heuristics with a meta-heuristic to find near-optimal solutions

for larger, realistic problems that cannot be solved using exact methods such as integer programming. Furthermore, a general, modular algorithm framework was given for the hybrid meta-heuristic algorithms that can incorporate many different greedy heuristics and meta-heuristics. Although the algorithms were designed to optimize the total tardiness objective, the representations, data structures, and framework developed are sufficiently general to be adapted for use on other objectives as well.

In order to test the performance of the algorithms developed, extensive computational experiments were carried out on a wide variety of problems. Four different problem generation algorithms were developed (detailed in Appendix 1) to provide problems of different sizes, tightness levels, numbers of processing areas, and qualitative characteristics. The computational experiments were extensive enough to require nearly a full month of continuous computing time. These experiments tested the earliest due date heuristic algorithm, CPLEX (for small and medium-sized problems) and several varieties of hybrid meta-heuristic algorithms. The algorithms were assessed with respect to solution quality (objective function value), variance (consistency), and computational time. The hybrid meta-heuristic algorithms were effective in providing near-optimal solutions. Where optimal solutions could be obtained it was found that the hybrid meta-heuristic algorithms tended to give solutions much closer to the optimal solution than to the EDD heuristic solution. Furthermore, in some cases it was found that the hybrid algorithm solutions obtained within three minutes beat the CPLEX incumbent solutions at the CPLEX 10-minute time limit (in terms of objective function). Of the hybrid meta-heuristic algorithm varieties examined it was found that the local search meta-heuristic had superior performance in both solution quality and algorithm consistency.

Furthermore, for problems involving multiple processing areas it was found that the high area swap probability yielded superior performance in solution quality, consistency, and time.

This dissertation has contributed a number of advances to the discipline of spatial scheduling. However, a number of future research areas are indicated. One such area is to investigate the utility of the lower-bound developed in Chapter 6 for finding solutions via integer programming. By incorporating the calculated lower bound into the mathematical models developed in Chapter 4 a large number of candidate solutions may be able to be pruned away in the branch-and-bound search, resulting in the speed of solution being greatly enhanced as well as the ability to solve significantly larger problems with integer programming. Another research area involves the development of better area-assignment heuristics. Chapter 7 developed several heuristics for quickly constructing schedules for several problem variants. However, only one scheme was introduced for area assignment to jobs: Round Robin. This is a very naïve approach and from the results of the meta-heuristic algorithms applied to multiple-area problems it is clear that the area assignment has a very important role in finding good solutions. One area of future research is to develop more intelligent area assignment schemes to be incorporated into the BLRT heuristic.

In this dissertation the solution methods focused on the total tardiness objective. However, the models developed contain reusable core sets of constraints that apply to many other problems as well. Additionally, the hybrid meta-heuristic framework and associated representations and data structures may be applied to problems with differing

objectives as well. Thus, other future research areas include adapting the methods developed in this dissertation for use on problems with different objective functions, and also experimenting with different hybrid algorithm components including alternative greedy heuristics and meta-heuristics.

BIBLIOGRAPHY

[1] J.J. Paulus and J. Hurink, "Adjacent Resource Scheduling: Why Spatial Resources are so Hard to Incorporate," *Electronic Notes on Discrete Mathematics,* vol. 25, pp. 113-116. 2006.

[2] C.W. Duin and E. Van Sluis, "On the Complexity of Adjacent Resource Scheduling," *Journal of Scheduling*, vol. 9, pp. 49-62, 2006.

[3] R. L. Graham, "Bounds for Certain Multiprocessing Anomolies" *Bell System Technical Journal*, vol. 45, pp. 1563-1581, 1996.

[4] L. A. Hall, "Approximability of Flow Shop Scheduling," *Proceedings of the 36th IEEE Annual Symposium on Foundations of Computer Science*, pp. 82-91, 1995.

[5] M. Sviridenko et al., "Makespan Minimization in Job Shops: A Polynomial Time Approximation Scheme," *Proceedings of the 31st Annual ACM Symposium on Theory of Computing,* pp. 394-399, 1999.

[6] R. L. Grahm et al., "Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey," *Annals of Discrete Mathematics*, vol. 5, pp. 287-386, 1979..

[7] Y. Bartal et al., "New Algorithms for an Ancient Scheduling Problem," *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pp. 51-58, 1992.

[8] B. S. Baker, et al., "Orthogonal Packings in Two Dimensions," *SIAM Journal on Computing*, vol. 9, pp. 846-855, 1980.

[9] F.K. Miyazawa and Y. Yakabayashi, "Three-dimensional packings with rotations", *Computers and Operations Research,* vol. 36, pp. 2801-2815, 2009.

[10] F.K. Miyazawa and Y. Yakabayashi, "Two- and three-dimensional parametric packing," *Computers and Operations Research,* Vol. 34, pp. 2949-2603, 2009

[11] Francois Clautiaux et al., "A new exact method for the two-dimensional bin-packing problem," *European Journal of Operational Research,* vol. 183, pp. 1196-1211, 2007.

[12] Kyoung Jun Lee and Jae Kyu Lee, "A Spatial Scheduling System and its Application to Shipbuilding: DAS-Curve," *Expert Systems with Applications*, vol. 10, pp. 311-324, 1996.

[13] Kyungchil Park et al., "Modeling and Solving the Spatial Block Scheduling Problem in a Shipbuilding Company," *Computers & Industrial Engineering,* vol. 30, pp. 357-364.

[14] K.K. Cho et al., "A Spatial Scheduling System for Block Painting Process in Shipbuilding," *CIRP Annals - Manufacturing Technology,* Vol. 50, pp. 339-342, 2001.

[15] R.K. Ahuja et al., "A Survey of Very Large-Scale Neighborhood Search Techniques," *Discrete Applied Mathematics*, vol. 123, pp. 75-102, 2002.

[16] D. E. Joslin and D. P. Clements, "'Squeaky Wheel' Optimization," *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, pp. 340-346, 1998.

[17] J. Frank and E. Kürklü, "Mixed Discrete and Continuous Algorithms for Scheduling Airborne Astronomy Observations," *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems.* Berlin/Heidelberg:Springer, pp. 183-200, 2005.

[18] S. Kirkpatrick et al., "Optimization by Simulated Annealing," *Science*, vol. 220, pp. 671-680, 1983.

[19] G. Rabadi et al., "A Heuristic Algorithm for The Just In-Time Single Machine Scheduling Problem With Setups: A Comparison With A Simulated Annealing," *The International Journal of Advanced Manufacturing Technology*, vol. 32, no. 3-4, pp. 326 – 335, 2007.

[20] Rabadi, G. et al., "A Simulated Annealing Algorithm for a Scheduling Problem with Setup Times," *Proceeding of Industrial Engineering Research Conference*, Orlando, Florida, 2002.

[21] G.C. Anagnostopoulos and G. Rabadi, "A Simulated Annealing Algorithm for the Unrelated Parallel Machine Scheduling Problem," *Proceeding of the World Automation Congress (WAC)*, Orlando, Florida, 2002.

[22] D.C. Montgomery, D.C., *Design and Analysis of Experiments*, 7th ed. NY: John Wiley & Sons, 2008.

[23] G. Rabadi et al., "A Branch-and-Bound Algorithm for the Early/Tardy Machine Scheduling Problem with a Common Due-Date and Sequence-Dependent Setup Time," *Computers & Operations Research Journal,* vol. 31, no. 10, pp. 1727-1751, 2004.

[24] M. Pinedo, *Scheduling Theory, Algorithms and Systems*, Second edition, Prentice Hall, 2008.

[25] T. Cheng and C. Sin, "A State-of-the-Art Review of Parallel-Machine Scheduling Research," *European Journal of Operational Research*, vol. 47, pp. 271–292, 1990.

[26] H. Lee and M. Pinedo, "A heuristic to minimize the total weighted tardiness with sequence-dependent setups," *European Journal of Operational Research*, vol. 100, pp. 464–474, 1997.

[27] W. Winston, *Operations Research: Applications and Algorithms,* 4th ed. Belmont, CA: Thomson, 2004

[28] A. Imai et al., "Berth allocation in a container port: Using a continuous location space approach," *Transportation Research Part B 39* , 199-221, 2005.

[29] V. Chvátal, *Linear Programming*. New York, NY: W.H. Freeman, 1983.

[30] R. Varghese and D.Y. Yoon, "Spatial block arrangement in shipbuilding industry using genetic algorithm for obtaining solution for anticipated bottleneck," Proceedings of the Fifteenth International Offshore and Polar engineering Conference vol. 4, pp. 774-780, 2005.

[31] M. Haouari and M. Serariri, "Heuristics for the variable sized bin-packing problem," *Computers and Operations Research*, vol. 36, pp. 2877-2884, 2009.

[32] Y. Wu et al., "Three-dimensional bin-packing problem with variable bin height," *European Journal of Operational Research*, vol. 202, pp. 347-355, 2010.

[33] M. Monaci and P. Toth, "A set-covering-based heuristic for bin packing problems," *INFORMS Journal on Computing*, vol. 18, pp.71-85, 2006.

[34] V.V. Vaziriani, *Approximation Algorithms*, Berlin: Springer, 2003.

[35] D.S. Johnson et al., "Worst-Case Performance Bounds for Simple One-Dimensional Packing Algorithms," SICOMP, vol 3, no. 4, 1974.

[36] M.R. Garey and D.S. Johnson, "A 71/60 theorem for bin packing," *Journal of Complexity*, vol. 1, pp. 65–106, 1985.

[37] M. Yue and L. Zhang, "A simple proof of the inequality MFFD(L)≤71/60 OPT(L) + 1,L for the MFFD bin-packing algorithm," *Acta Mathematicae Applicatae Sinica*, vol. 11, pp. 318–330, 1995.

[38] A. Bortfeldt and D. Mack, "A heuristic for the three-dimensional strip packing problem," *European Journal of Operational Research*, vol. 183, no. 3, pp. 1267-1279, 2007.

[39] G. Dosa, "The tight bound of the first-fit decreasing bin packing algorithm is FFD(I) ≤ 11/9OPT(I) + 6/9," *Combinatorics, Algorithms, Probabilistic, and Experimental Methodologies*, LNCS Vol. 4614, Springer-Berlin, pp. 1-11, 2007.

[40] C.S. Chen et al., "An analytical model for the container loading problem," European Journal of Operational Research, vol. 80, pp. 68-76, 1995.

[41] M.C. Riff et al., "A revision of recent approaches for two-dimensional strip-packing problems," Engineering Applications of Artificial Intelligence, vol. 22, pp. 823-827, 2009.

[42] N. Lesh et al., "Exhaustive approaches for 2D rectangular perfect packings," Information Processing Letters, vol. 99, pp. 161-169, 2004.

[43] S. Martello et al., "An exact approach to the strip-packing problem," INFORMS Journal of Computing, vol. 15, pp. 310-319, 2003.

[44] B. Chazelle, "The bottom-left bin packing heuristic: An efficient implementation," IEEE Transactions on Computers, vol. 32, pp. 697-707, 1984.

[45] E. Coffmann Jr. et al., "Performance bounds for level oriented two-dimensional packing algorithms," SIAM Journal on Computing, vol. 9 no. 1, pp. 808–826., 1980.

[46] E. Hopper and B.C.H. Torton, "An empirical investigation on metaheuristic and heuristic algorithms for a 2D packing problem," European Journal of Operational Research, vol. 128, pp. 34-57, 2001.

[47] N. Lesh and M. Mitzenmacher, "Bubble search: a simple heuristic for improving priority-based greedy algorithms," Information Processing Letters, vol. 97, pp. 161-169, 2006.

[48] N. Lesh et al., "New heuristic and interactive approaches to 2D rectangular strip packing," ACM Journal of Experimental Algorithmics, vol. 10, pp. 1-18, 2004.

[49] C. Mumford-Valenzuela et al., "Heuristics for Large Strip Packing Problems with Guillotine Patterns: An Empirical Study," Kluwer Academic Publishers, Dordrecht, pp. 501–522, 2003.

[50] A. Bortfeldt, "A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces," European Journal of Operational Research, vol. 172, pp. 814–837, 2006.

[51] A. Bortfeldt and H. Gehring, "New large benchmarks for the two-dimensional strip-packing problem with rectangular pieces," In: IEEE Proceedings of the Thirty Ninth Hawaii International Conference on Systems Sciences, p. 302., 2006

[52] D. Zhang et al., "A new heuristic recursive algorithm for the strip rectangular packing problem." Computers and Operations Research, vol. 33, pp. 2209–2217, 2006.

[53] A. Soke and Z. Bingul, "Hybrid genetic algorithm and simulated annealing for two-dimensional non-guillotine rectangular packing problems," Engineering Applications of Artificial Intelligence, No. 19, 557–567, 2006.

[54] E. Burke et al., "Metaheuristic enhancements of the best- fit heuristic for the orthogonal stock-cutting problem," Technical Report, University of Nottingham, NOTTCS-TR-2006-3, 2006.

[55] S. Boyd and J. Mattingly, "Branch and bound methods," Lecture notes from Stanford University retrieved from http://www.stanford.edu/class/ee364b/notes/bb_notes.pdf, 2007.

[56] T.H. Cormen et al., *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001.

[57] S.A. Cook, "The Complexity of Theorem Proving Procedures," Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, pp. 151–158, 1971.

[58] M.R. Garey. and D.S. Johnson, "Complexity results for multiprocessor scheduling under resource constraints," *SIAM Journal on Computing* , vol. 4, pp. 397–411, 1975.

[59] The Ruby Programming Language. [Online.] Available: http://www.ruby-lang.org/en.

[60] C. Garcia and G. Rabadi, "An Optimization Model for Scheduling Problems with Two-Dimensional Spatial Resource Constraints," Proceedings of Modeling and Simulation (MODSIM) World 2009 Conference, October 14-16 2009, Virginia Beach, VA, 2009.

[61] T. Segaran, *Programming Collective Intelligence*, Sebastopol, CA: O'Reilly Publishers, 2007

[62] R.J. Howarth and E.P.K Tsang, "Spatio-temporal Conflict Detection and Resolution", *Constraints*, vol. 3, no. 4, pp. 343-361, 1998.

[63] Yuanbin Song and D. K. H. Chua, "Detection of Spatio-temporal Conflicts in Temporal 3D Space System," *Advances in Engineering Software*, vol. 36, no.11-12, pp. 814-826, 2005.

[64] D. Bertsimas. and J. Tsitsiklis, "Simulated Annealing", *Statistical Science*, vol. 8, no. 1, pp. 10-15, 1993.

[65] D.A. Abramson et al., "Cooling Schedules for Simulated Annealing Based Scheduling Algorithms," Proceeding of the 17th Australian Computer Science Conference, pp 541 – 550, 1994..

[66] D. Connolly, "An improved annealing scheme for the QAP," European Journal of Operational Research, vol. 46, pp. 93-100, 1990..

[67] E.H.L. Aarts et al., "A Computational Study of Local Search Algorithms for Job Shop Scheduling," ORSA Journal on Computing, vol. 6, no. 2, pp. 118-125, 1994.

[68] M. Randall et al., "A Simulated Annealing Approach to Communication Network Design,", *Journal of Combinatorial Optimization*, vol. 6, pp. 55-65, 2002.

APPENDIX 1: PROBLEM GENERATION ALGORITHMS

In order to facilitate testing of the models and algorithms developed in this dissertation, four separate problem generation algorithms were developed. This dissertation research spans over several years and the four algorithms were developed in sequence. Each algorithm produces problems with differing characteristics including tightness levels ranging from 1 to 10. A tighter problem is one with more jobs needing to be processed at a any given time on average, resulting in more competition for spatial resources.

Algorithm 1 was the first problem generation algorithm to be developed. In this appendix it is taken directly from Garcia and Rabadi [60] and is reproduced almost verbatim (with a few modifications for contextual fit). Problems generated by Algorithm 1 were used to verify the correctness of the greedy heuristic algorithms in Chapter 7 and to test the computational speed of these algorithms. However, when Model 3 solved with CPLEX was applied to these problems, it was found that they were not typically tight enough to produce a non-zero optimal objective value. Consequently, it was not deemed suitable for investigating solution algorithm performance. As a result newer, better problem generation algorithms were then developed.

Algorithm 2 was developed to enable problems of a higher tightness to be produced. Algorithm 2 utilizes a fundamentally different approach from Algorithm 1 and enables a much greater range of problem tightness levels to be generated, both at the lower end as well as the higher end. Algorithm 2 also correlates job sizes to problem tightness levels to a moderate degree. In particular, it limits how small a job can be based

on a tightness level. Thus, higher tightness levels will tend to produce fewer small jobs and as a result, tighter problems tend to have more bulky jobs.

Algorithm 3 was developed using a similar approach to Algorithm 2. However, Algorithm 3 was designed to remove the correlation between job size and problem tightness. Algorithm 3 is also capable of producing a wide range of tightness levels but without a correlation between problem tightness and job size.

Algorithm 4 was also developed using a similar approach to Algorithm 2 and Algorithm 3. However, Algorithm 4 was designed specifically to correlate job size with job processing time. This is a very realistic assumption that may be encountered in practice. For instance, a machine may need to move over more space or perform more operations on a larger job, resulting in a longer processing time. Algorithm 4 uses a simple mechanism to generate job duration: the job processing time is equal to the job area (width multiplied by height). This results in a perfect correlation between size and processing time. Algorithm 4 has does not correlate problem tightness with job size.

A.I. ALGORITHM 1 (PROBLEMS FOR HEURISTIC PERFORMANCE TESTING)

Algorithm 1 was developed to generate random spatial scheduling problem instances of varying tightness levels, based on a given maximum area width and height. Algorithm 1 was implemented in Ruby [59] and used primarily to generate problems for testing the performance of the greedy heuristic algorithms developed in Chapter 7. In this algorithm, each job's width and height are generated using a uniform distribution over [1, width or height of area]. Durations (i.e. processing times) are generated using a uniform distribution over [5, 25]. Earliest allowable start times and deadlines are generated using

an incremented *current time* and a *tightness* factor ranging from 1 to 10, with 10 generating the most tightly-packed problems. Earliest allowable start times were generated by

$E = current\ time - r$

where r is a random number uniformly distributed over [0, *(current time) / tightness*]. Deadlines are generated by

$D = current\ time + current\ job\ duration + (r * tightness)$

where r is a random number uniformly distributed over [0, *current job duration*]. Finally, *current time* is initialized to 0 prior to the generation of any job and subsequently incremented after each job generation by

$Increment = 10 * r / tightness$

where r is a random number uniformly distributed over [0, *current job duration*].

A.2 ALGORITHM 2: (CORRELATION OF PROBLEM TIGHTNESS WITH JOB SIZE)

Algorithm 2 produces problems with a modest correlation between problem tightness and job size. This algorithm consists of two procedures: a procedure to generate a problem for a single area and another that uses this single-area procedure to generate a problem for multiple areas. In the single-area procedure the input consists of 1) the desired number of jobs to be generated n, 2) a processing area width and height, and 3) a *tightness* factor ranging from 0 to 10, with 10 generating the most tightly-packed problems. Intuitively, a "tighter" problem is one that requires more jobs to be scheduled

within a shorter period of time, thus requiring a tighter packing of jobs inside the area to reach an optimal solution. This tightness factor determines 1) the minimum width and height each job can have relative to the area dimensions (tighter = larger minimum dimensions), 2) the amount of "slack" time – the amount of time in addition to processing time – a job may have before it is tardy (tighter = less slack time), and 3) the variance in earliest start dates, processing times, and due dates (tighter = less variance). The problem generation algorithm is centered on the concept of a *current time* that is incremented each time a job is created. The higher the tightness level, the less *current time* is incremented during each iteration. The procedure for generating a single-area problem is shown below.

```
Procedure GENERATE_SINGLE_AREA_PROBLEM (Width, Height, Tightness, n)
DO:
    Jobs := Ø
    Current_Time := 0
    Min_Dim_Percent := 0.3
     // Minimum percentage of area dimension a job must have at max tightness
    Ratio := Tightness / 10          // The tightness ratio – always assumed to be in [0, 1]
    Max_Dur := 25                    // Maximum duration
    Min_Dur := 5
    Max_Time_Pad := 50
      // Maximum amount of slack time until next job becomes eligible
   Max_Time_Increment := 50
     // Maximum amount current time may be incremented in a single iteration
    FOR j := 1, j ≤ n DO:
        W := uniform random value in [min(1, Ratio * Min_Dim_Percent * Width),
              Width]
        H := uniform random value in [min(1, Ratio * Min_Dim_Percent * Height),
             Height]
        Processing_Time := uniform random value in [Min_Dur + ((Max_Dur –
                                Min_Dur) * Ratio), Max_Dur]
        Earliest_Start := Current_Time
        Due_Date := Earliest_Start + Processing_Time + (1-Ratio) * Max_Time_Pad
        Current_Time := Current_Time + uniform random value in [(1-Ratio) *
                              Max_Time_Increment]
      ADD job = {W, H, Earliest_Start, Processing_Time, Due_Date} TO Jobs
     END
     RETURN Jobs
END GENERATE_SINGLE_AREA_PROBLEM
```

Table 70: The GENERATE_SINGLE_AREA_PROBLEM procedure for Algorithm 2

In order to generate multiple-area problems, the above procedure is utilized for each area. The input for generating multiple-area problems consists of 1) width and height values for areas 1...*m*, 2) a tightness factor ranging from 1.0-10, and 3) the desired number of jobs to be generated *n*. The multiple-area procedure is shown below.

```
Procedure GENERATE_MULTIPLE_AREA_PROBLEM (Width[1...m], Height[1...m],
Tightness, n)  DO:
    Jobs := Ø
    FOR i := 1,  i ≤ m DO:
        IF i < m  THEN K := ⌊n / m⌋
        ELSE K :=⌈n / m⌉
        Next_Set := GENERATE_SINGLE_AREA_PROBLEM (Width[i], Height[i],
                    Tightness, K)
        Jobs := Jobs ∪Next_Set
    END
    RETURN Jobs
END GENERATE_MULTIPLE_AREA_PROBLEM
```

Table 71: The GENERATE_MULTIPLE_AREA_PROBLEM procedure for Algorithm 2

A.3. ALGORITHM 3 (NO CORRELATION OF PROBLEM TIGHTNESS WITH JOB SIZE)

Algorithm 3 provides a similar range of tightness to Algorithm 2 in the problems it is capable of generating. However, whereas Algorithm 2 has a limitation on how small a job can be based on tightness, there is no such limitation in Algorithm 3. As a consequence, job sizes are uniformly distributed regardless of tightness. Algorithm 3 utilizes the same basic logic as Algorithm 2, with the exception of the manner in which job widths and heights are generated. Structurally, Algorithm 3 uses the same two procedures as Algorithm 2: GENERATE_SINGLE_AREA_PROBLEM and GENERATE MULTIPLE_AREA_PROBLEM. The GENERATE MULTIPLE_AREA_PROBLEM procedure is exactly the same, while the GENERATE_SINGLE_AREA_PROBLEM is modified accordingly. The version of GENERATE_SINGLE_AREA_PROBLEM used in Algorithm 3 is shown below.

```
Procedure GENERATE_SINGLE_AREA_PROBLEM (Width, Height, Tightness, n)
DO:
    Jobs := Ø
    Current_Time := 0
    Min_Dim_Percent := 0.3
    // Minimum percentage of area dimension a job must have at max tightness
    Ratio := Tightness / 10          // The tightness ratio – always assumed to be in [0, 1]
    Max_Dur := 25                    // Maximum duration
    Min_Dur := 5
    Max_Time_Pad := 50
      // Maximum amount of slack time until next job becomes eligible
    Max_Time_Increment := 50
      // Maximum amount current time may be incremented in a single iteration
    FOR j := 1, j ≤ n DO:
        W := uniform random value in [1, Width]
        H := uniform random value in [1, Height]
        Processing_Time := uniform random value in [Min_Dur + ((Max_Dur –
                                Min_Dur) * Ratio), Max_Dur]
        Earliest_Start := Current_Time
        Due_Date := Earliest_Start + Processing_Time + (1-Ratio) * Max_Time_Pad
        Current_Time := Current_Time + uniform random value in [(1-Ratio) *
                                Max_Time_Increment]
      ADD job = {W, H, Earliest_Start, Processing_Time, Due_Date} TO Jobs
    END
    RETURN Jobs
END GENERATE_SINGLE_AREA_PROBLEM
```

Table 72: The GENERATE_SINGLE_AREA_PROBLEM procedure for Algorithm 3

A.4. ALGORITHM 4: (CORRELATION OF JOB SIZE TO PROCESSING TIME)

Algorithm 4 is designed to generate jobs where there is a direct correlation between job size and job processing time. This is a very realistic assumption that may be readily expected in industrial problems, such as when machines need to cover more space to process jobs (resulting in longer processing times). The basic structure of Algorithm 4 is the same as with Algorithm 2 and 3, consisting of the two procedures GENERATE_SINGLE_AREA_PROBLEM and GENERATE_MULTIPLE_AREA_PROBLEM. In Algorithm 4, the GENERATE_MULTIPLE_AREA_PROBLEM procedure is exactly the same as is found in Algorithm 2 and Algorithm 3. However, the GENERATE_SINGLE_AREA_PROBLEM procedure is modified to correlate the job processing time to job size. In this problem, the processing time is set to be equal to the total job area (width by height). Additionally, Algorithm 4 does not correlate job size to problem tightness; in this respect it is like Algorithm 3 rather than Algorithm 2. The GENERATE_SINGLE_AREA_PROBLEM for Algorithm 4 is shown below.

```
Procedure GENERATE_SINGLE_AREA_PROBLEM (Width, Height, Tightness, n)
DO:
    Jobs := Ø
    Current_Time := 0
    Min_Dim_Percent := 0.3
      // Minimum percentage of area dimension a job must have at max tightness
    Ratio := Tightness / 10          // The tightness ratio – always assumed to be in [0, 1]
    Max_Dur := 25                    // Maximum duration
    Min_Dur := 5
    Max_Time_Pad := 50
      // Maximum amount of slack time until next job becomes eligible
    Max_Time_Increment := 50
      // Maximum amount current time may be incremented in a single iteration
    FOR j := 1, j ≤ n DO:
        W := uniform random value in [1, Width]
        H := uniform random value in [1, Height]
        Processing_Time := Width * Height
        Earliest_Start := Current_Time
        Due_Date := Earliest_Start + Processing_Time + (1-Ratio) * Max_Time_Pad
        Current_Time := Current_Time + uniform random value in [(1-Ratio) *
                                Max_Time_Increment]
      ADD job = {W, H, Earliest_Start, Processing_Time, Due_Date} TO Jobs
    END
    RETURN Jobs
END GENERATE_SINGLE_AREA_PROBLEM
```

Table 73: The GENERATE_SINGLE_AREA_PROBLEM procedure for Algorithm 4

VITA

Christopher J. Garcia

Old Dominion University
Department of Engineering Management and Systems Engineering
241 Kaufman Hall
Norfolk, VA 23529

EDUCATION

M.S., Florida Institute of Technology, 2008
Major: Operations Research

M.S., Nova Southeastern University, 2004
Major: Computer Science

B.S. Old Dominion University, 2001
Major: Computer Science

BIOGRAPHY

Christopher Garcia was raised in Fredericksburg, Virginia and attended Chancellor High School. After graduating he moved to Norfolk to attend Old Dominion University, earning a bachelor's degree in computer science. He subsequently earned a master's degree in computer science from Nova Southeastern University and a master's degree in operations research from Florida Institute of Technology prior to beginning his Ph.D. studies at Old Dominion University. Christopher has held positions as Operations Research Analyst, Consultant, and Senior Software Engineer within the defense, transportation, and healthcare technology industries. He has also authored publications in the areas of scheduling, optimal component selection, health care policy, and modeling & simulation.