View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by VCU Scholars Compass

Virginia Commonwealth University
VCU Scholars Compass

Theses and Dissertations

Graduate School

2014

Spatial Scheduling Algorithms for Production Planning Problems

Sudharshana Srinivasan
Virginia Commonwealth University

Follow this and additional works at: http://scholarscompass.vcu.edu/etd

Part of the Applied Mathematics Commons

© The Author

Downloaded from

http://scholarscompass.vcu.edu/etd/3406

This Dissertation is brought to you for free and open access by the Graduate School at VCU Scholars Compass. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of VCU Scholars Compass. For more information, please contact libcompass@vcu.edu.

©Sudharshana Srinivasan, May 2014

All Rights Reserved.

SPATIAL SCHEDULING ALGORITHMS FOR PRODUCTION PLANNING PROBLEMS

A Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

by

SUDHARSHANA SRINIVASAN

Master of Science in Applied Mathematics

Virginia Commonwealth University, May 2011

Director: Dr. J. Paul Brooks,

Associate Professor, Department of Statistics and Operations Research

Virginia Commonwewalth University

Richmond, Virginia

May, 2014

In loving memory of Amma. You are, and will always remain, my hero.

Acknowledgements

I embarked upon an educational journey five years ago, without the faintest idea on how to arrive at my destination. I was fortunate to meet the right people at the right time in the right places, who gave me the right direction. This work would not have been possible without the support and assistance provided by these wonderful individuals and I would like to take this opportunity to express my sincerest gratitude to them. Listing every person that showed me the way would fill volumes. Hence I will mention a chosen few and leave the rest in a special place in my heart.

First and foremost, I want to thank my advisors, who have been instrumental in the successful completion of my defense and dissertation. Dr. Jill Wilson, has been a role model for me ever since I started graduate school. She got me interested in operations research and specifically in the exciting area of scheduling. She has always been supportive not only academically, but also emotionally through the rough road to finish this thesis. Dr. Paul Brooks helped me find my dissertation topic and guided me as I inched toward the finish line. I consider myself extremely lucky to have been counseled by these two very remarkable professors. Their expert and valuable insights have made me pursue approaches that I might not have otherwise considered and their careful reading of this document helped me refine my work and its presentation.

I would like to thank the members of my doctoral committee, Dr. David Edwards, Dr. Jose Dula, and Dr. Yongjia Song, for taking an interest in evaluating my work and for their time and contribution in helping me finish this last part of my dissertation. I also want to extend my gratitude to Dr. Jason Merrick for having faith in my ability to undertake and complete the doctoral program and for being a constant source of encouragement throughout these years in graduate school. I wish to express my appreciation to Dr. D'Arcy Mays for being the most approachable and cheerful administrator I have known.

The support and advice from peers and friends made taking classes and tests less of an ordeal. I owe my gratitude to my fellow students, Toni, Ben, Phil, and Derrick, for making this ride memorable. To my ‘lab mates’, Shaun, Robert, Jacob, Eric, and Chris, thank you guys, you officially made friday mornings more enjoyable and productive.

Words are insufficient to convey the emotions I experience when I think of my families. I am forever obliged to the Sorrels, Toni, Calvin, and little Ben, for adopting me and for being my home away from home. I cannot imagine having survived thus far without them. My heartfelt thanks to Aai and Baba, your unconditional love and support provided much needed motivation. I am truly blessed to have become a part of such a lovely family. Finally, a great big hug and thank you to the two most important men in my life - Appa and Advait. Appa you have always believed in me and have given me the freedom to write my own story. Thank you for tolerating my teenage years, for teaching me to be independent, and for being the coolest dad on the planet. Advait, thanks for being amazing, for being there for me when it matters, and for your infinite patience. Your love and encouragement has helped me sail through all the good times and the bouts of frustration, for that I will be eternally indebted.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract

SPATIAL SCHEDULING ALGORITHMS FOR PRODUCTION PLANNING PROBLEMS

By Sudharshana Srinivasan

A Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

Virginia Commonwealth University, 2014.

Director: Dr. J. Paul Brooks,

Associate Professor, Department of Statistics and Operations Research

Spatial resource allocation is an important consideration in shipbuilding and large-scale manufacturing industries. Spatial scheduling problems (SSP) involve the non-overlapping arrangement of jobs within a limited physical workspace such that some scheduling objective is optimized. Since jobs are heavy and occupy large areas, they cannot be moved once set up, requiring that the same contiguous units of space be assigned throughout the duration of their processing time. This adds an additional level of complexity to the general scheduling problem, due to which solving large instances of the problem becomes computationally intractable. The aim of this study is to gain a deeper understanding of the relationship between the spatial and temporal components of the problem. We exploit these acquired insights on problem characteristics to aid in devising solution procedures that perform well in practice. Much of the literature on SSP focuses on the objective of minimizing the makespan of the schedule. We concentrate our efforts towards the minimum sum of completion times objective and state several interesting results encountered in the pursuit of developing fast and reliable solution methods for this problem. Specifically, we develop mixed-integer programming models that identify groups of jobs (batches) that can be scheduled simultaneously. We identify scenarios where batching is useful and ones where batching jobs provides a solution with a worse objective function value. We present computational analysis on large instances and prove an approximation factor on the performance of this method, under certain conditions. We also provide greedy and list-scheduling heuristics for the problem and compare their objectives with the optimal solution. Based on the instances we tested for both batching and list-scheduling approaches, our assessment is that scheduling jobs similar in processing times within the same space yields good solutions. If processing times are sufficiently different, then grouping jobs together, although seemingly makes a more effective use of the space, does not necessarily result in a lower sum of completion times.

CHAPTER 1

INTRODUCTION

1.1 Motivation

Everyday we are confronted by the logistical challenges in planning and allocation of resources (human, material, or a physical quantity) to tasks; examples include the scheduling of processors to jobs in a computing environment, effectively allocating personnel to a project, generating timetables for a school, creating airline schedules, and reserving operating rooms for surgeries. Traditional scheduling problems create schedules that efficiently utilize resources to complete tasks while minimizing some objective criterion. The objective functions encountered in scheduling literature can be broadly classified into min-sum or min-max objectives. For example, minimizing the makespan and minimizing maximum lateness are min-max objectives, while minimizing the (weighted) sum of completion or start times of jobs is a min-sum objective. The problems are classified using the three-field classification scheme introduced by Graham et al. 1979, which specifies the machine environment, job characteristics, and optimality criterion for the problem. Further variations to the problems are created by including or excluding, one or more, of the following characteristics:

1. Online scheduling: Job characteristics are known as and when jobs become available.
2. Precedence of jobs: A job cannot be processed until certain other jobs have completed.
3. Pre-emption: Jobs can be interrupted and resumed at any given time.

4. Due-dates and release-dates: Jobs can be processed only after they are released and have to be processed prior to their due dates. Sometimes tardy jobs are permitted, but penalized.

5. Additional resources: Renewable resources are limited within each time period and replenished at the end of the time period. Equipment and labor are some of the classic examples for renewable resources. Nonrenewable resources, such as capital budget, are limited over the overall planning horizon, but not restricted within each time period.

There are many results on the complexity of these problems under different conditions (Garey, Johnson, and Sethi 1976; Lenstra, Rinnooy Kan, and Brucker 1977). The complexity in solving these problems and diversity in their applications have given rise to varied solution methodologies (Baker 1974; Coffman Jr 1976; Pinedo 2008; Lawler et al. 1993; Brucker 2001). While some approaches are exact, others are designed to provide fast, cheap, and reliable solutions. Various general integer programming formulations have been proposed for discrete scheduling problems (Queyranne and Schulz 1994; Dyer and Wolsey 1990). One of the notable exact methods used to solve these mathematical models is the branch and bound method (Land and Doig 1960). Essentially, branch and bound algorithms use lower and upper bounds on the optimal objective value to control the enumeration of its solutions. Often such procedures become computationally intractable with an increase in problem size. Hence, there is a need to develop alternative methods that produce near-optimal solutions in polynomial time. Approximation algorithms (AA) deliver solutions with provable quality that are bounded in runtime. Heuristics, in contrast, generate quick solutions without any provable performance guarantees.

In the past 50 years, there has been a great deal of work on developing AA for NP hard optimization problems, specifically scheduling problems (Hochbaum 1997; Vazirani 2001). Encouraged by this successful application of AA to scheduling problems (Lee and Chen 2009; Savelsbergh, Uma, and Wein 1998; Hall et al. 1997), this thesis aims to construct approximation algorithms for the spatial scheduling problem with the objective to minimize the sum of completion times. Spatial scheduling problems involve the assignment of a spatial resource along with conventional temporal attributes. Therefore, the solutions should not only specify starting times for each job, but also locations for the placement of jobs on a physical workspace. The motivation for the choice of our objective is two-fold. First, within the realm of spatial scheduling problems, much energy is devoted to studying methods for minimizing the 'makespan' or total project duration (Perng, Lai, and Ho 2009; Zheng et al. 2011; Koh et al. 2011; Zhang and Chen 2012; Caprace et al. 2013). Results for other objectives are scarce (Garcia and Rabadi 2011; Kolisch 2000) and to the best of our knowledge this is the first study to consider the sum of completion times objective. Second, when jobs are independent and competing for the same resource, the cost associated with individual completion times becomes more relevant and natural (Leung, Kelly, and Anderson 2004). For example, inventory systems may require that certain products with higher overheads per time be processed ahead of others.

The process of designing AA is similar to that of exact algorithms. In order to design reliable algorithms with good performance guarantees, we need to gain an understanding of the structure of the problem. Therefore we concentrate our efforts in examining the simplest form of the problem by considering a workspace area of fixed dimensions. We disallow precedence constraints, due-dates, rotation of jobs, and set-up times. By doing so, we are able to focus on the relationship between the spatial and temporal components in the problem and gain a better understanding of the problem characteristics. The insights acquired by studying the simple problem can enable us to devise algorithms that perform well in practice. The remainder of this section briefly describes the spatial scheduling problem, provides relevant background, and reviews the fundamental concepts used in subsequent chapters of this dissertation.

1.2 Problem Description

In large-scale production and manufacturing industries, the assembly line involves scheduling jobs that each require a certain amount of two-dimensional space within a limited physical workspace. This requires that the schedule not only assign starting times to each job, but also locations and orientations. It is also important to ensure that jobs are assigned the same contiguous units of space throughout the duration of their processing. Thus, any solution to this problem attempts to answers the following questions:

(a) When to schedule the jobs? (Temporal component)

(b) Where to schedule the jobs? (Spatial component)

Mathematically, the spatial scheduling problem is described as follows: Given a set J of jobs with fixed processing times p_j, height h_j, and width w_j for each $j \in J$ and a workspace B of height H and width W; does there exist a schedule of the jobs that minimizes the given objective criterion such that the spatial and temporal restrictions in each time period are satisfied? While the spatial restrictions ensure that jobs collectively fit within the physical workspace, the temporal constraints involve conditions (if present) on the starting times of the jobs. Common objectives include minimizing total project duration (min-max objective) or minimizing the average time to complete individual jobs (min-sum objective). The number of tasks, nature of the resources (renewable or non-renewable), and constraints (precedence of tasks, due-dates, or limitations on availability) add to the complexity of the scheduling problem.

Throughout the remainder of this thesis we study the spatial scheduling problem for the objective of minimizing the sum of completion times (denoted by Z). We disallow precedence constraints, due-dates, rotation of jobs, and set-up times. We will hereafter refer to this variation of the problem as SSP.

1.3 Fundamental Concepts

In this section we present relevant background information on the concepts used in the subsequent chapters of this dissertation. The information presented here can be found in Garey and Johnson 1979, Nemhauser and Wolsey 1988, Vazirani 2001, and Williamson and Shmoys 2010. In what follows, we denote linear programming or linear program as LP, integer programming or integer program as IP, and mixed-integer programming or mixed-integer program as MIP.

1.3.1 Integer programming

Mathematical programming is a modeling technique that assists in making decisions to address real-world problems. The decisions are modeled as unknown variables and the objective, a function of the decision variables, represents the best possible solution for the model. The restrictions on decisions being made express the constraints or mathematical inequalities of the problem. Given a vector $c \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$, and a matrix $A \in \mathbb{R}^{mxn}$, we want to find a vector $x \in \mathbb{Z}^n$ that solves the following integer linear program:

$$\min \mathbf{c^T x}$$

$$s.t. A\mathbf{x} \geq \mathbf{b}$$

$$\mathbf{x} \in \mathbb{Z}^{\mathbf{n}}$$

Relaxing the integrality constraints produces an LP. MIPs use a combination of integer and real-valued variables. The integrality constraints help capture the discrete nature of certain decisions. Both linear and integer programs seek to find an optimal assignment of values to the variables that satisfy the constraints while minimizing or maximizing a linear objective function. LPs are solved using the simplex method (Dantzig 1965) or barrier methods (Karmarkar 1984) and the approach to solving (mixed) integer programs is with the use of branch and bound algorithms, first introduced by Land and Doig 1960. The algorithm performs controlled enumeration of the solution space with the use of lower and upper bounds on the optimal solution. The first step is to remove the integrality constraints in the original problem, resulting in an LP relaxation of the problem. Upon solving the relaxation, if we obtain integral solutions, without explicitly imposing them, we have obtained an optimal solution. If not, as is usually the case, we partition the set of integer solutions into two disjoint sets. This is referred to as a variable dichotomy branching strategy. For example, if an integer variable, say x, has a value of 5.6 in the relaxation, we partition the set of variables into two sets: one with $x \geq 6$ and $x \leq 5$. The two new problems created by branching on variable x are solved and the optimal solution for the problem is the better solution among the two subproblems. We continue the process of enumeration as we build a search tree by selecting different branching variables. The algorithm terminates when all the subproblems are solved or fathomed. Although the subproblems we are attempting to solve are LPs, using the branch and bound algorithm can take long due to the sheer number of subproblems being solved. One of the ways to optimize the procedure and speed up the process is by using upper bounding (for minimization problems) heuristics or approximation algorithms.

1.3.2 Computational Complexity

Computational complexity theory classifies decision problems based on the inherent difficulty in solving them computationally. Class P contains problems with instances that can be solved and verified in polynomial time. In other words, the worst-case running-time for these algorithms is polynomial in size of the input. Sometimes we may not have an efficient (polynomial time) algorithm for a problem, however if given a solution we may be able to verify its accuracy in polynomial time. These problems are defined as non-deterministic polynomial-time (NP) problems. A problem p is said to be NP-complete if is it in NP and if every other problem in NP is reducible to p. Many combinatorial optimization problems have been shown to be NP-complete by reductions from known NP-complete problems (Garey and Johnson 1979). Approximation and parameterization techniques are commonly used to find reasonable solutions for such problems.

1.3.3 Approximation Algorithms

Although Johnson 1974 coined the term "approximation algorithm", earlier papers had proved the performance guarantees of heuristics (Graham 1966). Approximation algorithms deliver solutions with provable quality that are bounded in runtime. Suppose we wish to solve an NP-hard minimization problem consisting of instances in $\mathcal{I}$. Let $z(I)$=min$\{c_I x : x \in S_I\}$ $\forall I \in \mathcal{I}$. Let $\mathcal{A}$ be an algorithm that operates on instances in $\mathcal{I}$, and let $\mathcal{A}(I)$ be the objective value resulting from the application of $\mathcal{A}$ to I. Let $\rho \geq 1$.

Definition 1 *$\mathcal{A}$ is a ρ-approximation algorithm for $\mathcal{I}$ if for each $I \in \mathcal{I}$, $\mathcal{A}$ runs in time polynomial in the size of I, and $\mathcal{A}(I) \leq \rho z(I)$. $\mathcal{A}$ is said to have a factor ρ, also referred to as the performance guarantee of $\mathcal{A}$.*

Observe that we compare the objective value obtained by the application of the algorithm to instance I with the optimal objective value $z(I)$ for that instance. In practice, however, this is not possible, because if $z(I)$ is known then there would be no need to approximate it. To overcome this issue and calculate ρ, we compare $\mathcal{A}(I)$ with a lower bound for $z(I)$, say $L(I)$. Lower bounds can be obtained using LP or combinatorial relaxations. Since $L(I) \leq z(I)$ we have,

$\mathcal{A}(I) \leq \rho L(I) \implies \mathcal{A}(I) \leq \rho z(I)$.

CHAPTER 2

SPATIAL SCHEDULING PROBLEM

In this chapter we describe the spatial scheduling problem and present mathematical formulations for SSP. We also survey the literature and present our approaches in developing solution methodologies for the problem.

2.1 Introduction

Spatial scheduling is often an important consideration in large-scale manufacturing and production industries. Assembly units are often heavy, occupy large areas, and cannot be relocated easily. Therefore they require a fixed quantity of physical space throughout the duration of processing time. Since workspace area available at such facilities is limited, assembly lines need to consider efficient utilization of spatial resources along with temporal considerations common to traditional scheduling problems. For example, if we have a workspace of 25 square units, and three jobs with characteristics as specified in Table 1, we would like to find starting times and a layout of the jobs within the given space that minimizes the sum of the completion times objective.

Table 1. Sample instance of the spatial scheduling problem

Job	Width	Height	Processing Time
1	2	1	2
2	2	2	2
3	4	3	2

2.2 Background

Spatial scheduling problems have been studied in recent years, mostly in the context of shipbuilding (Lee et al. 1997; Park et al. 1996; Cho et al. 2001; Raj and Srivastava 2007). Modern day ship building practices use prefabricated sections or 'blocks'. These super-structures that occupy large spaces are built elsewhere in the yard and then transported to the building dock for assembly. This is known as 'block construction'. The shipbuilding facilities usually have a limitation of the space they can provide. Hence they require an optimal method of spacing the assemblies and the equipment required. There are also temporal restrictions such as deadlines and due-dates. Thus, this problem is ideal for spatial scheduling. Figure 1 shows the layout of jobs before and after applying spatial scheduling solution procedures. We can see that initially the space is not utilized effectively and some jobs are waiting to be processed. On applying some spatial scheduling method, we get a better utilization of the space and no delays in processing of jobs.

Pioneering work in this area began between 1991 and 1993, when the Daewoo Shipbuilding Company and Korea Advanced Institute of Science and Technology jointly initiated the DAewoo Shipbuilding Scheduling (DAS) project to develop innovative scheduling systems that improve productivity and efficiency. Lee et al. 1997 applied the two-dimensional arrangement of convex polygons at the Daewoo shipyard. Park et al. 1996 extended this work and applied it at the block painting shop in the Hyundai shipyard. Cho et al. 2001 included failure blocks, the blocks that are scheduled but are not worked upon and workload balancing constraints that guaranteed an even distribution of workload on the equipment available. The solution system for the block painting process includes an operation strategy algorithm, a block-scheduling algorithm, a block arrangement algorithm, and a block assignment

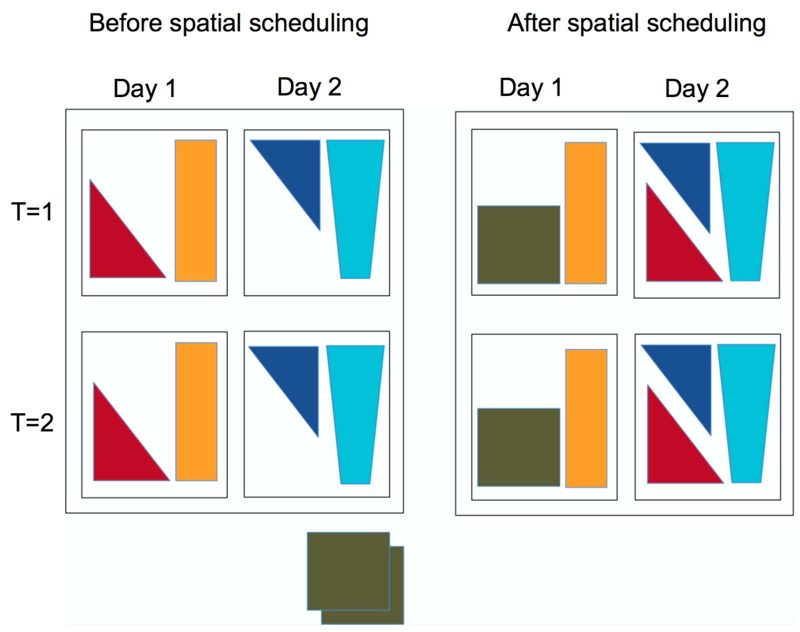


Fig. 1. Depicting the motivation for spatial scheduling with jobs each requiring two time units scheduled over a two-day horizon

algorithm. Garcia 2010 considers a class of spatial scheduling problems that involve scheduling each job into one of several possible processing areas in parallel to minimize total tardiness. The authors suggest an integer programming formulation and a strip-packing based heuristic to solve the problem. Raj and Srivastava 2007 give a mixed-integer programming model to describe the spatial scheduling problem. They make assumptions about the orientations of the blocks and use three-dimensional bin-packing approaches to analyze data collected from an Indian shipyard. They also discuss applications of stacking goods in shelves in large retail stores.

Mullen and Butler 2000 present the design of a spatially constrained harvest-scheduling model that uses a genetic algorithm as the optimization technique. A single unit of a forest area containing relatively uniform species composition is referred to as a stand. The genetic algorithm specifies the order by which to place stands in the schedule by evaluating the population of permutations of the stand identification numbers. Spatial goals are becoming more frequent aspects of forest management plans with changes in regulatory and organizational policies. Enhancing these operations will benefit a variety of economic and conservation objectives. Boston and Bettinger 2001 discuss the addition of green-up constraints, that is, harvesting restrictions that prevent the cutting of adjacent units for a specified period of time, suggesting the application of spatial schedules to forest plans. The authors discuss a two-stage method that in one stage uses linear programming to assign volume goals, and in a second stage uses a tabu search genetic algorithm technique to minimize the deviations from the volume goals while maximizing the present net revenue. BoWang and Gadow 2002 present a multi-objective function consisting of a timber component and a spatial component. The model is solved using the method of simulated annealing. It is shown that adjusting the adjacency matrix for the stands generates a variety of spatial harvesting patterns.

Much of the literature focuses on approaches with the objective of minimizing the makespan or $\max_{j \in J} C_j$, where C_j denotes the completion time of job j in a given schedule (Perng, Lai, and Ho 2009; Zheng et al. 2011; Koh et al. 2011; Zhang and Chen 2012; Caprace et al. 2013). Garcia and Rabadi 2011 provide a meta heuristic algorithm to minimize the total tardiness for instances with release dates and multiple processing areas. Kolisch 2000 develops a MIP-formulation and list-scheduling algorithm that minimizes the weighted sum of tardiness for assembly scheduling problems in the presence of part availability constraints and a spatial resource. While the applications listed above are by no means comprehensive, they demonstrate the value in investigating this problem. The remainder of this document discusses the approaches taken in the pursuit of finding reliable and efficient solution methods for SSP.

2.3 Formulation

In this study, we consider SSP which involves scheduling a set J of jobs to minimize the sum of completion times (Z) such that the jobs fit within the limited physical workspace B of height H and width W. Each job $j \in J$ requires a space of height h_j and width w_j over its processing time p_j. Without loss of generality we assume that the problem data is integer-valued.

In the presentation that follows, we describe two mathematical formulations of SSP; a time-indexed formulation and a general MIP formulation. Time-indexed formulations are known to serve as good guides for developing approximation algorithms as they provide strong lower bounds for LP relaxations of scheduling problems (Akker, Van Hoesel, and Savelsbergh 1999; Phillips, Stein, and Wein 1998; Savelsbergh, Uma, and Wein 1998; Goemans 1997; Hall et al. 1997; Sousa and Wolsey 1992). The disadvantage of using these formulations are their size; even small instances have a large number of variables and constraints (Akker, Hurkens, and Savelsbergh 2000). Although decomposition techniques and other cut generation schemes may be used to alleviate some of the computational difficulties encountered (Akker, Hurkens, and Savelsbergh 2000; Lee and Sherali 1994) while implementing these formulations, since our goal is to find solutions quickly we chose the alternate formulation. Note that both formulations presented here minimize the sum of starting times. The sum of completion times objective used in our analysis Z_{OPT} is then obtained by adding $\sum_{j \in J} p_j$ to the optimal objective obtained from solving these models.

2.3.1 Time-indexed Formulation

The following time-indexed formulation is based on the one provided in Hardin, Nemhauser, and Savelsbergh 2008. Our motivation in creating this formulation is to analyze the polyhedral structure of the problem and potentially identify valid inequalities for the problem.

$$\min \sum_{j \in J} \sum_{t \in T} \sum_{x \in X} \sum_{y \in Y} r_{xytj} \tag{2.1}$$

subject to:

$$\sum_{j \in J} q_{xytj} \leq 1 \qquad \forall x \in X, y \in Y, t \in T \tag{2.2}$$

$$\sum_{t \in T} \sum_{x \in X} \sum_{yinY} r_{xytj} = 1 \qquad \forall j \in J \tag{2.3}$$

$$\sum_{x'=x}^{x+w_j-1} q_{x'ytj} \geq w_j r_{xytj} \quad \forall x \in X, y \in Y, t \in T, j \in J \tag{2.4}$$

$$\sum_{y'=y}^{y+h_j-1} q_{xy'tj} \geq h_j r_{xytj} \quad \forall x \in X, y \in Y, t \in T, j \in J \tag{2.5}$$

$$\sum_{t'=t}^{t+p_j-1} q_{xyt'j} \geq p_j r_{xytj} \quad \forall x \in X, y \in Y, t \in T, j \in J \tag{2.6}$$

$$r_{xytj}, q_{xytj} \in \{0, 1\} \quad \forall x \in X, y \in Y, t \in T, j \in J \tag{2.7}$$

where,

J is the set of all jobs

T is the set of all the discretized units of time when a job can be scheduled

X is the set of all possible x-coordinate points that discretize the width of the workspace

Y is the set of all possible y-coordinate points that discretize the height of the workspace

$$r_{xytj} = \begin{cases} 1 & \text{if job j starts at time t at location (x,y)} \\ 0 & \text{otherwise} \end{cases}$$

$$q_{xytj} = \begin{cases} 1 & \text{if job j is processing at time t at location (x,y)} \\ 0 & \text{otherwise} \end{cases}$$

For $x \in X$, $y \in Y$, $t \in T$, and $j \in J$

Z_{OPT} is obtained by adding $\sum_{j \in J} p_j$ to the objective in (2.1). Constraint (2.2) ensures that there is only be one job that is processing at any given time and location and constraint (2.3) guarantees that each job starts exactly once. The constraints (2.4-2.6) relate the starting time of the jobs with the time periods that the job is processing. If a job j starts at time t at location (x, y), then it must be processing until time period $t + p_j - 1$ at locations x to $x + w_j - 1$ and y to $y + h_j - 1$.

2.3.2 General Mixed-Integer Programming Formulation

The following mixed-integer programming formulation based on two dimensional bin packing problem has been adapted from Garcia 2010.

$$\min \sum_{j \in J} z_j \tag{2.8}$$

subject to:

$$\begin{aligned}
-x_i + x_j - W\alpha_{ij} &\geq -W + w_i & \forall i, j \in J, i \neq j \quad (2.9)\\
-y_i + y_j - H\beta_{ij} &\geq -H + h_i & \forall i, j \in J, i \neq j \quad (2.10)\\
-z_i + z_j - T\gamma_{ij} &\geq -T + p_i & \forall i, j \in J, i \neq j \quad (2.11)\\
\alpha_{ij} + \alpha_{ji} + \beta_{ij} + \beta_{ji} + \gamma_{ij} + \gamma_{ji} &\geq 1 & \forall i, j \in J, i \neq j \quad (2.12)\\
-x_i - w_i &\geq -W & \forall i \in J \quad (2.13)\\
-y_i - h_i &\geq -H & \forall i \in J \quad (2.14)\\
x_i, y_i, z_i &\geq 0 & \forall i \in J \quad (2.15)\\
\alpha_{ij}, \beta_{ij}, \gamma_{ij} &\in \{0, 1\} & \forall i, j \in J \quad (2.16)
\end{aligned}$$

where,

J is the set of all jobs

x_j is the x-coordinate of job $j \in J$

y_j is the y-coordinate of job $j \in J$

z_j is the z-coordinate (start time) for job $j \in J$

$$\alpha_{ij} = \begin{cases} 1 & \text{if no overlap occurs between jobs i and j in the x direction} \\ 0 & \text{otherwise} \end{cases}$$

$$\beta_{ij} = \begin{cases} 1 & \text{if no overlap occurs between jobs i and j in y direction} \\ 0 & \text{otherwise} \end{cases}$$

$$\gamma_{ij} = \begin{cases} 1 & \text{if no overlap occurs between jobs i and j in z direction} \\ 0 & \text{otherwise} \end{cases}$$

For $i, j \in J$

Here, T is an upper bound on the makespan of the schedule. Z_{OPT} is obtained by adding $\sum_{j \in J} p_j$ to the objective in (2.8). Constraints (2.9)-(2.12) prevent overlap from occurring in the x (width), y (height), and z (time) dimensions. We use constraints (2.13) and (2.14) to ensure that the jobs are confined to the physical dimensions of the workspace.

2.4 Special Cases

The original motivation for studying SSP was to better understand the relationships between the spatial and temporal components of the problem. The following special cases of SSP have interesting applications and we anticipate that some of our results can aid in developing solution methods for these problems or vice-versa.

1. Bin Packing

   Given a set J of items, the bin packing problem attempts to find the minimum number of bins of capacity H required to hold the items so that each item is contained in exactly one bin. In an instance of SSP if we normalize processing times and widths of jobs such that $p_j = 1$ and $w_j = 1$, $\forall j \in J$ and $W = 1$, then solutions to SSP can be viewed as solutions to the bin packing problem, where bins of size H correspond to the the time periods in the schedule and items of size $h_j, \forall j \in J$ correspond to the jobs.

2. Single Machine Scheduling

   Instances of SSP with $w_j = W$ and $h_j = H$, $\forall$ jobs j $\in$ J reduce to instances of single machine scheduling problems, where jobs consume all of the available spatial resource during each time period that its processing.

3. Uniform Resource-Constrained Scheduling

   Let us consider the set of instances of SSP with $h_j = H$, $\forall$ jobs $j \in J$.This is the sequential processor task-scheduling problem or the uniform resource-constrained scheduling problem (URCSP). Each job $j \in J$ requires a set of sequential processors w_j over their processing time p_j. There is a constant amount W of the resource (processors) available at every time period. We are interested in finding a schedule of the jobs that adheres to the resource constraints for the minimum sum of completion times objective.

2.5 Methodology

Allocating adjacent resources (spatial or otherwise) is a difficult problem. Such problems arise in resource allocation applications such as berth allocation for ships (Lee and Chen 2009). Duin and Sluis 2006 study the adjacency requirement for check-in counters at airports. For each flight in a given planning horizon, a feasible assignment of adjacent desks to flights is required where the objective is to minimize the total number of desks involved. They show that minimizing the number of counters, given the number of counters needed at each time slot for each flight, is NP-hard. Further in SSP, the two-dimensional resource is actually a single spatial resource, i.e. the length and width of jobs are not independent of each other and have to be assigned simultaneously in contiguous units. Unlike some other resources, physical space is neither divisible nor distributable making the design of solution procedures for the spatial scheduling problem harder. Therefore, solving large instances of the problem using exact methods becomes computationally intractable. This presents a need for developing methods that yield quick and provably good solutions. Heuristic procedures and approximation algorithms have long been used to provide reasonable solutions for real-world problems, which is the direction we pursue in finding solution methods for SSP.

SSP can be imagined as a combination of a scheduling and a packing problem. We require that jobs be packed within a two-dimensional space such that some scheduling objective is minimized. Although packing and scheduling problems have diverse applications, the similarities in their mathematical structures and integrating classifications have been studied in the past (Monaci 2003; Hartmann 2000; Coffman, Garey, and Johnson 1978). As discussed earlier, approaches for the minimum sum of completion times (Z) objective are scarce. To this end, we consider approaches based on packing (iterative and efficient area models) and list-scheduling (product and ratio methods) to construct approximation algorithms for SSP that minimize Z. Detailed descriptions of these methods and analysis of their results are found in Chapters 3 and 4.

CHAPTER 3

BATCH-SCHEDULING

3.1 Introduction

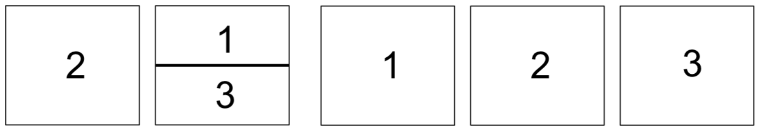
The spatial component of SSP makes understanding solution methods for multi-dimensional packing problems, particularly bin packing and strip-packing problems, both relevant and crucial. Lodi, Martello, and Monaci 2002 provide a survey of the the models and algorithms used to solve the two-dimensional bin packing (2DBP) problem. Batch-scheduling originated from the problem of scheduling 'burn-in' operations at large-scale integrated circuit manufacturing (Lee, Uzsoy, and Martin-Vega 1992; Brucker et al. 1998). Mathirajan and Sivakumar 2006 survey the literature for scheduling of batching processors in the semi-conductor industry. The central idea in batch-scheduling is grouping similar jobs together to form a 'batch'. All jobs in a batch start at the same time and the next batch starts upon completion of the longest job in the previous batch. The processing time of a batch is equal to the largest processing time of any job in the batch. Our goal is to utilize ideas from 2DBP to design batch-scheduling strategies that solve large instances of SSP. This approach let us reduce the complexity of the problem by relaxing the temporal constraints in the original problem.

Assume we have a set J of N jobs such that $p_1 \leq p_2 \leq \cdots \leq p_N$. When all the jobs fit in the space simultaneously, irrespective of the difference in their processing times p_j they are placed in the same batch. So $Z = \sum_{j \in J} p_j$. If none of the jobs simultaneously fit the space, SSP reduces to single machine scheduling (SMS). Then each job is its own batch and $Z = \sum_{j=1}^{N} \sum_{i=1}^{j} p_i$. Smith 1956 proved that ordering jobs in the nondecreasing sequence of their processing times is optimal for SMS. In general, while minimizing the sum of completion times, the more jobs we can fit earlier in our schedule the lower the objective. Therefore, it seems intuitive to always group jobs together rather than assign them to individual batches. Consider an instance of SSP with W=H=3 and job data as given in Table 2. We see that jobs 1 and 3 are the only jobs that fit the space simultaneously.

Table 2. Example instance with three jobs such that only jobs 1 and 3 simultaneously fit the space

Job	Width	Height
1	3	2
2	2	3
3	3	1

Let the processing times $[p_1, p_2, p_3]$ = [2, 3, 7] and let us assume we schedule the batch with the lowest processing time first. We define batch processing time as the maximum processing time of jobs in a batch. Therefore, the batch sequence is $\{2\}$ and $\{1, 3\}$ as seen in Figure 2(a). Then the objective value for batched jobs is calculated as $Z = p_1 + 3p_2 + p_3$ =2 + 9 + 7 = 18. Alternately, if we schedule the batches in their own batch, the sequence is $\{1\}, \{2\}, \{3\}$ as seen in Figure 2(b) and $Z = 3p_1 + 2p_2 + p_3$ = 6 + 6 + 7 = 19. This shows that grouping jobs can result in a lower sum of completion times objective.



(a) Batching (b) No Batching

Fig. 2. Batching sequence for example with three jobs

Now suppose, $[p_1, p_2, p_3] = [2,\ 5,\ 20]$. When jobs 1 and 3 are batched, $Z = p_1 + 3p_2 + p_3 = 2 + 15 + 20 = 37$. Without batching, $Z = 3p_1 + 2p_2 + p_3 = 6 + 10 + 20 = 36$. Thus in scenarios where jobs with large differences in processing times are grouped together, the batching approach does not necessarily lead to improvement in the objective.

Proposition 1 *For N jobs, assume that $p_1 < p_2 < \cdots < p_{N-1} < p_N$. When jobs with both the largest and smallest processing times are assigned to the same batch, that is $\{1, N\}$ form a batch, and $(N-1)p_1 - p_2 - \cdots - p_{N-1} < 0$, the sum of completion times obtained by batching is greater than the objective value obtained without batching.*

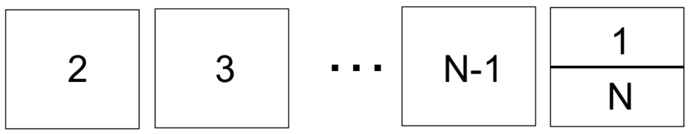


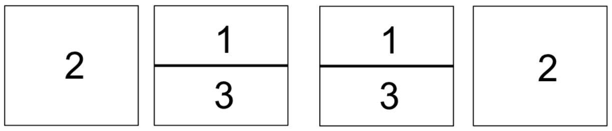
Fig. 3. Sequence in which batches are scheduled for Proposition1

Proof: Let $\sum\limits_{j \in J} C_j^b$ be the sum of completion times obtained when batching and $\sum\limits_{j \in J} C_j^n$ be the sum of completion times obtained without batching. Jobs $\{1, N\}$ form a batch, while the other jobs are each assigned individual batches. Since $p_1 < p_2 < \cdots < p_{N-1} < p_N$, batch $\{1, N\}$ is processed at the end of the schedule (see Figure 3). So $\sum\limits_{j \in J} C_j^b = p_1 + Np_2 + \cdots + 3p_{N-1} + p_N$. If each job is assigned its own batch, then $\sum\limits_{j \in J} C_j^n = Np_1 + (N-1)p_2 + \cdots + 2p_{N-1} + p_N$. Therefore,
$\sum\limits_{j \in J} C_j^b$ - $\sum\limits_{j \in J} C_j^n$

$$=[p_1 + Np_2 + \cdots + 3p_{N-1} + p_N] - [Np_1 + (N-1)p_2 + \cdots + 2p_{N-1} + p_N]$$

$$=(N-1)p_1 - p_2 - \cdots - p_{N-1}$$

Hence, when $(N-1)p_1 - p_2 - \cdots - p_{N-1} < 0$, the result follows. □



(a) Batch Sequence (b) Alt. Sequence

Fig. 4. Comparing strategies for sequencing batches

This contradicts the notion of scheduling as many jobs earlier in the schedule to minimize our objective. So, our intuitions about general scheduling problems do not always apply directly to problems with spatial resources. Batching seems to be beneficial only when processing times are similar.

Let us now look at the example with $[p_1, p_2, p_3] = [2, 3, 5]$. If we schedule batches in the sequence is $\{2\}$ and $\{1, 3\}$ as seen in Figure 4(a), the objective $Z = p_1 + 3p_2 + p_3$ $= 2 + 9 + 5 = 16$. Instead, if we were to schedule the batches in the sequence $\{1, 3\}$ then $\{2\}$ as seen in Figure 4(b), $Z = p_1 + p_2 + 2p_3 = 2 + 3 + 10 = 15$. This suggests that scheduling the jobs in the increasing order of batch processing times is not always effective.

Proposition 2 *Consider a set J of N jobs such that $p_1 < p_2 < \cdots < p_{N-1} < p_N$. If m of those jobs are in a batch B, including Job 1 and Job N, and $p_n > mp_i, \forall i \in J \backslash B$, then placing batch B at the end of a schedule provides a better objective than placing it at the beginning of the schedule.*

Proof: Let $\sum_{j \in J} C_j^b$ be the sum of completion times obtained when processing batch B at the end of the schedule and $\sum_{j \in J} C_j^a$ be the sum of completion times obtained using an alternate sequencing of batches (batch B is the first batch to be scheduled). Let $\{1, u_1, u_2, \cdots, u_{m-2}, N\}$ be the m jobs in batch B such that $p_1 < p_2 < \cdots < p_{u_1 - 1} <$ $p_{u_1} < p_{u_2} < \cdots < p_{u_{m-2}} < p_{u_{m-1}} < \cdots < p_N$.

The sequence of batches scheduled in increasing order of batch processing times is

$\{2\}, \{3\}, \cdots, \{u_1 - 1\}, \cdots, \{u_{m-1}\}, \cdots, \{N-1\}, \{B\}$.

So $\sum_{j \in J} C_j^b = (p_1 + p_{u_1} + p_{u_2} + \cdots + p_{u_{m-2}} + p_N) + (Np_2 + (N-1)p_3 + \cdots + (m+1)p_{N-1})$.

Alternately, if we place batch B at the beginning of the schedule, $\sum_{j \in J} C_j^a$ is given by $(p_1 + p_{u_1} + p_{u_2} + \cdots + p_{u_{m-2}} + (N-m+1)p_N + (N-m)p_2 + (N-m-1)p_3 + \cdots + p_{N-1})$.

Therefore, $\sum_{j \in J} C_j^a - \sum_{j \in J} C_j^b$

$$= \begin{aligned}[t] &[(p_1 + p_{u_1} + p_{u_2} + \cdots + p_{u_{m-2}} + (N-m+1)p_N + (N-m)p_2 + (N-m-1)p_3 + \cdots + p_{N-1})] - \\ &\qquad [(p_1 + p_{u_1} + p_{u_2} + \cdots + p_{u_{m-2}} + p_N) + (Np_2 + (N-1)p_3 + \cdots + (m+1)p_{N-1})] \end{aligned}$$

$$= -mp_2 - mp_3 - \cdots - mp_{N-1} + (N-m)p_N$$

Hence, when $p_N > mp_2$, $p_N > mp_3$, $\cdots$, $p_N > mp_{N-1}$, the result follows. □

These results are useful in gaining an insight into the characteristics of spatial scheduling problems, thereby resulting in the development of more efficient algorithms and strategies to approximate its solutions.

3.1.1 Forming the Batches

When forming the batches, based on our previous analysis, we observe that placing jobs with the smallest and largest processing times in the same batch does not necessarily result in a good batching scheme. So, we group jobs similar in processing time that also efficiently utilize the space to form a batch. We present two MIP models, iterative and efficient area, that identify the assignment of jobs to batches. The objective for the iterative model is to minimize the maximum difference in processing times among jobs for each batch. The efficient area model extends this idea by also minimizing the total unused area in each batch. Both MlP formulations have been adapted from the 2DBP model found in Pisinger and Sigurd 2005. Let J denote the set of jobs and B the set of batches. Since at most each job can be its own batch, the number of batches equals the number of jobs (N).

3.1.1.1 Iterative model

In the iterative model (M1), we add a constraint to limit the number of batches (S) being used by the model. The strategy is to iterate through possible values for S, starting at $S = N - 1$ and decreasing by 1 in each iteration. From the set of all solutions, we can then chose the batching that results in the lowest sum of completion times objective value. The formulation for the iterative model is given by,

$$\min \sum_{b \in B} (Zmax_b - Zmin_b) \tag{3.1}$$

$$\sum_{b \in B} r_{jb} = 1 \qquad \forall j \in J \tag{3.2}$$

$$x_j + w_j \leq W \qquad \forall j \in J \tag{3.3}$$

$$y_j + h_j \leq H \qquad \forall j \in J \tag{3.4}$$

$$x_i + w_i - x_j \leq W(1 - l_{ij}) \qquad \forall i, j \in J, i < j, b \in B \tag{3.5}$$

$$y_i + h_i - y_j \leq H(1 - b_{ij}) \qquad \forall i, j \in J, i < j, b \in B \tag{3.6}$$

$$l_{ij} + l_{ji} + b_{ij} + b_{ji} + (1 - r_{ib}) + (1 - r_{jb}) \geq 1 \qquad \forall i, j \in J, b \in B \tag{3.7}$$

$$Zmin_b \leq (p_j - M) r_{jb} + M q_b \qquad \forall j \in J, b \in B \tag{3.8}$$

$$Zmax_b \geq p_j r_{jb} \qquad \forall j \in J, b \in B \tag{3.9}$$

$$r_{jb} \leq q_b \qquad \forall j \in J, b \in B \tag{3.10}$$

$$\sum_{j \in J} r_{jb} - \epsilon q_b \geq 0 \qquad \forall b \in B \tag{3.11}$$

$$\sum_{b \in B} q_b = S \tag{3.12}$$

$$x_j, y_j \geq 0 \qquad \forall j \in J \tag{3.13}$$

$$Zmin_b, Zmax_b \geq 0 \qquad \forall b \in B \tag{3.14}$$

$$l_{ij}, b_{ij} \in \{0, 1\} \qquad \forall i, j \in J \tag{3.15}$$

$$r_{jb} \in \{0, 1\} \qquad \forall j \in J, b \in B \tag{3.16}$$

$$q_b \in \{0, 1\} \qquad \forall b \in B \tag{3.17}$$

where,

J is the set of all jobs

B is the set of all batches

x_j is the x-coordinate of job $j \in J$

y_j is the y-coordinate of job $j \in J$

$Zmax_b$ is the maximum processing time of jobs in batch $b \in B$

$Zmin_b$ is the minimum processing time of jobs in batch $b \in B$

$$r_{jb} = \begin{cases} 1 & \text{if job j is in batch b} \\ 0 & \text{otherwise} \end{cases}$$

$$l_{ij} = \begin{cases} 1 & \text{if job i is to the left of job j} \\ 0 & \text{otherwise} \end{cases}$$

$$b_{ij} = \begin{cases} 1 & \text{if job i is below job j} \\ 0 & \text{otherwise} \end{cases}$$

$$q_b = \begin{cases} 1 & \text{if batch b is nonempty} \\ 0 & \text{otherwise} \end{cases}$$

For $i, j \in J$ and $b \in B$.

Here, constraint (3.2) ensures that each job is assigned to only one batch. Constraints (3.3) and (3.4) ensure that jobs do not exceed the width and height of the space. We use constraints (3.5), (3.6), and (3.7) to prevent overlap of jobs within the space. Constraint (3.8) determines the minimum processing time within a batch, while (3.9) identifies the maximum processing time for each batch. If job j is in batch b ($r_{jb} = 1$) then constraint (3.10) makes sure batch b is non-empty ($q_b = 1$). When no jobs are present in a batch, constraint (3.11) ensures that the batch is empty or $q_b = 0$. Constraint (3.12) sets the number of batches to be used by the model to some value S. We set $\epsilon = 0.5$ and define $M = 1 + \max_{j \in J} p_j$.

3.1.1.2 Efficient Area model

While solving $N - 1$ instances of M1 for different values of S finds the best possible batch assignment, the second approach or efficient area model (M2) proposes to solve just one MIP to decide when and where to place jobs. The efficient area model includes an area utilization component to the existing objective. So, model 2 minimizes the maximum difference in processing times and the amount of workspace area that remains unused for each batch. The formulation for the efficient area model is given by,

$$\min \sum_{b \in B} (Zmax_b - Zmin_b + UA_b) \tag{3.18}$$

$$\sum_{b \in B} r_{jb} = 1 \qquad \forall j \in J \tag{3.19}$$

$$x_j + w_j \leq W \qquad \forall j \in J \tag{3.20}$$

$$y_j + h_j \leq H \qquad \forall j \in J \tag{3.21}$$

$$x_i + w_i - x_j \leq W(1 - l_{ij}) \qquad \forall i, j \in J, i < j, b \in B \tag{3.22}$$

$$y_i + h_i - y_j \leq H(1 - b_{ij}) \qquad \forall i, j \in J, i < j, b \in B \tag{3.23}$$

$$l_{ij} + l_{ji} + b_{ij} + b_{ji} + (1 - r_{ib}) + (1 - r_{jb}) \geq 1 \qquad \forall i, j \in J, b \in B \tag{3.24}$$

$$Zmin_b \leq (p_j - M) r_{jb} + M q_b \qquad \forall j \in J, b \in B \tag{3.25}$$

$$Zmax_b \geq p_j r_{jb} \qquad \forall j \in J, b \in B \tag{3.26}$$

$$r_{jb} \leq q_b \qquad \forall j \in J, b \in B \tag{3.27}$$

$$\sum_{j \in J} r_{jb} - \epsilon q_b \geq 0 \qquad \forall b \in B \tag{3.28}$$

$$WHq_b - \sum_{j \in J} w_j h_j r_{jb} = UA_b \qquad \forall b \in B \tag{3.29}$$

$$x_j, y_j \geq 0 \qquad \forall j \in J \tag{3.30}$$

$$Zmin_b, Zmax_b, UA_b \geq 0 \qquad \forall b \in B \tag{3.31}$$

$$l_{ij}, b_{ij} \in \{0, 1\} \qquad \forall i, j \in J \tag{3.32}$$

$$r_{jb} \in \{0, 1\} \qquad \forall j \in J, b \in B \tag{3.33}$$

$$q_b \in \{0, 1\} \qquad \forall b \in B \tag{3.34}$$

where,

J is the set of all jobs

B is the set of all batches

x_j is the x-coordinate of job $j \in J$

y_j is the y-coordinate of job $j \in J$

$Zmax_b$ is the maximum processing time of jobs in batch $b \in B$

$Zmin_b$ is the minimum processing time of jobs in batch $b \in B$

UA_b is the unused area in batch $b \in B$

$$r_{jb} = \begin{cases} 1 & \text{if job j is in batch b} \\ 0 & \text{otherwise} \end{cases}$$

$$l_{ij} = \begin{cases} 1 & \text{if job i is to the left of job j} \\ 0 & \text{otherwise} \end{cases}$$

$$b_{ij} = \begin{cases} 1 & \text{if job i is below job j} \\ 0 & \text{otherwise} \end{cases}$$

$$q_b = \begin{cases} 1 & \text{if batch b is nonempty} \\ 0 & \text{otherwise} \end{cases}$$

For $i, j \in J$ and $b \in B$.

Here, constraint (3.19) ensures that each job is assigned to only one batch. Constraints (3.20) and (3.21) ensure that jobs do not exceed the width and height of the space. We use constraints (3.22), (3.23), and (3.24) to prevent overlap of jobs within the space. Constraint (3.25) determines the minimum processing time within a batch, while (3.26) identifies the maximum processing time for each batch. If job j is in batch b ($r_{jb} = 1$) then constraint (3.27) makes sure batch b is non-empty ($q_b = 1$). When no jobs are present in a batch, constraint (3.28) ensures that the batch is empty or $q_b = 0$. Constraint (3.29) calculates the unused area for each batch b. We set $\epsilon = 0.5$ and define $M = 1 + \max_{j \in J} p_j$.

3.1.2 Scheduling the batches

Once the batches are identified using either M1 or M2, it is also important to decide the sequence in which to schedule the batches. Smith 1956 proved that the shortest processing time (SPT) rule, ordering jobs in the nondecreasing sequence of their job processing times, is optimal for the single machine scheduling problem. The idea is that by scheduling shorter jobs earlier in the schedule, more jobs can finish early resulting in a smaller sum. For SSP, the rule translates to scheduling the batches in the nondecreasing sequence of their batch processing times. For example, if P_1 is the maximum processing time of all jobs in batch 1 and P_2 is the maximum processing time of all jobs in batch 2; then batch 1 is scheduled before batch 2 if and only if $P_1 \leq P_2$. However, as noted before, there are instances for which this rule does not necessarily provide a better objective value. Therefore, we also consider scheduling jobs in the non-decreasing order of the average batch processing times, or the average processing time of all the jobs in a batch. We indicate the two scheduling rules as MAX and AVG respectively.

3.1.3 Post processing algorithm

By solving each instance of SSP using the iterative and efficient area models, we determine the assignments of jobs to batches that minimizes the maximum difference in processing times while efficiently utilizing the workspace. With this information, we then schedule the batches by applying either the MAX or AVG rules. Once a schedule is created, we calculate the sum of completion times for the jobs as $Z_H = \sum_{j \in J} C_j^H$, where C_j^H is the completion time for job j. With this batching algorithm, each job must wait until the previous batch has completed before it can start processing. In reality there may be jobs in the current batch that finish processing before the final job in the batch. This means that jobs in later batches may be able to start earlier in the schedule. Since neither MIP model takes into account the temporal dimension, we use the algorithm schedule_update to incorporate this observation and improve Z_H. For each batch the algorithm determines if jobs can start processing earlier in the schedule. If job j can be moved ahead in time by say t_j units, then the completion time is updated as, $\hat{C}_j = C_j^H - t_j$ and $\hat{Z} = \sum_{j \in J} \hat{C}_j$ is the new objective value.

Algorithm 1 schedule_update

```
1: Batch Sequence, $b \leftarrow 1, 2, \cdots, m$
2: $J_b \leftarrow$ Set of jobs in batch $b$
3: $C_j \leftarrow$ Completion time of job $j$
4: $ahead_j \leftarrow 0, \forall j \in J$
5: for $b = 2$ to $m$ do
6:     for $j \in J_b$ do
7:         for $t = C_j - p_j - 1$ to 1 do
8:             if Space exists then
9:                 $ahead_j \leftarrow ahead_j + 1$
10:                $flag = 1$
11:            else
12:                $ahead_j = 0$
13:                continue
14:            end if
15:            if $ahead_j > 0$ then
16:                if $flag = 1$ then
17:                    continue
18:                else
19:                    break
20:                end if
21:            end if
22:        end for
23:        if $flag = 1$ then
24:            $C_j \leftarrow C_j - ahead_j$
25:            Update schedule
26:        end if
27:    end for
28: end for
```

3.2 Performance Analysis

In this section, we present solution guarantees on the objective values Z_H generated by the batch-scheduling algorithms. We refer to Z_{OPT} as the optimal objective for the SSP formulation. We begin by analyzing instances with a set J of N jobs such that at any given time k jobs can simultaneously fit the space $(W \times H)$ and $p_1 \leq p_2 \leq \cdots \leq p_N$ that define a special case of URCSP.

Theorem 1 *When N=nk for any $n, k \in \mathbb{Z}_+^*$, $w_j \leq W$ and $h_j = H\ \forall j \in J$, and $p_1 \leq p_2 \leq \cdots \leq p_N$, if each batch has k jobs, then batch-scheduling is a k-approximation algorithm.*

Proof: Let J denote the set of nk jobs and B the set of batches. Since $p_1 \leq p_2 \leq \cdots \leq p_N$ and we are minimizing the sum of completion times objective, we would want to schedule jobs in the sequence $1, 2, \cdots, N$. By design only k jobs can simultaneously fit the space. If the first k jobs are scheduled in a batch at the beginning of the schedule, job $k+1$ does not start until any of the jobs finish processing. The first job to finish processing would be job 1. So completion time, $C_{k+1} = p_{k+1} + p_1$. Applying this reasoning we note that a lower bound on the optimal objective for these instances is given by, $Z_{OPT} \geq \sum_{j=1}^{n}(n-j+1)(p_{jk} + p_{jk-1} + \cdots + p_{jk-k+1})$.

Since only k jobs can occupy the space at any given time, the number of batches is $\frac{nk}{k} = n$. If we use the MAX rule, $Z_b = max_{j \in b} p_j \geq p_{bk}$ for each batch $b \in B$ and $Z_1 \leq Z_2 \leq \cdots \leq Z_n$. Let us order the jobs in the sequence of the batches they are assigned and in the increasing order of their processing times within each batch, so that p_{bi} now refers to the processing time of the i^{th} job in batch b. It is important to note here that p_{bi} could be greater than the bi term in the sequence $p_1, p_2, p_{bi}, \cdots, p_N$. The completion time of job j based on this new ordering is then calculated as the sum of its processing time and the completion times of the batches scheduled ahead of j as:

$$\begin{aligned}
C_j^H &= p_j, \text{ if job } j \leq k \text{ and} \\
C_j^H &= \begin{cases} p_j + Z_{b-1} + \cdots + Z_1 & \text{if job j is in batch b and j > k} \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

$$\begin{aligned}
Z_H &= \sum_{j \in J} C_j^H && (3.35)\\
&= \sum_{j \in J} p_j + kZ_1 + k(Z_1 + Z_2) + k(Z_1 + Z_2 + Z_3) + \cdots + \\
&\quad +k(Z_1 + Z_2 + ... + Z_{n-1}) && (3.36)\\
&= \sum_{j \in J} p_j + k[(n-1)Z_1 + (n-2)Z_2 + \cdots + 2Z_{n-2} + Z_{n-1}] && (3.37)\\
&= \sum_{j \in J} p_j + k[(n-1)p_k + (n-2)p_{2k} + \cdots + \\
&\quad +2p_{(n-2)k} + p_{(n-1)k}] && (3.38)\\
&= \sum_{j=1}^{n} (p_{jk-1} + \cdots + p_{jk-k+1}) + \sum_{j=1}^{n} ((nk - jk + 1)p_{jk}) && (3.39)\\
&= \sum_{j=1}^{n} (p_{jk-1} + \cdots + p_{jk-k+1}) + k \sum_{j=1}^{n} ((n - j + \frac{1}{k})p_{jk}) && (3.40)\\
&= \sum_{j=1}^{n} (p_{jk-1} + \cdots + p_{jk-k+1}) + \sum_{j=1}^{n} ((n - j + \frac{1}{k})p_{jk}) \\
&\quad +(k-1) \sum_{j=1}^{n} ((n - j + \frac{1}{k})p_{jk}) && (3.41)\\
&\leq \sum_{j=1}^{n} (p_{jk-1} + \cdots + p_{jk-k+1}) + \sum_{j=1}^{n} ((n - j + \frac{1}{k})p_{jk}) \\
&\quad +(k-1) \sum_{j=1}^{n} ((n - j + 1)p_{jk}) && (3.42)\\
&\leq \sum_{j=1}^{n} (p_{jk-1} + \cdots + p_{jk-k+1}) + \sum_{j=1}^{n} ((n - j + \frac{1}{k})p_{jk}) \\
&\quad +(k-1)Z_{OPT} && (3.43)\\
&\leq Z_{OPT} + (k-1)Z_{OPT} && (3.44)\\
&= kZ_{OPT} && (3.45)
\end{aligned}$$

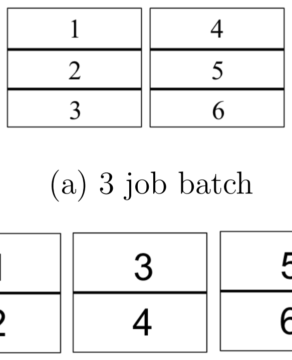
Equation (3.36) is obtained from the definition of completion times, C_j^H and we get equation (3.38) per the definition of Z_b. In each batch of k jobs, since the batch processing time is p_k, this is the only processing time included in the calculation of completion times for the batches scheduled later. The processing times of the remaining (k-1) jobs are not repeated in this objective as seen in equation (3.39). The bounds seen in (3.43) and (3.44) follow from the definition of Z_{OPT}. $\square$

The bound shown helps us understand what makes instances of SSP hard. The real difficulty in solving instances of SSP lies in the spatial constraints as reflected by the bound, which is dependent on k, the number of jobs you can simultaneously fit within the given workspace. Notice that the bound is independent of the number of jobs. Also, recall that when minimizing the sum of completion times, we want to schedule more jobs earlier in the schedule. This is because the completion times of a job includes the completion times of the jobs earlier in the schedule. When $k = 1$, SSP reduces to SMS and our batching heuristic becomes SPT, which we know is optimal (Smith 1956). Our bound depicts that as k increases, the solutions given by the batch-scheduling algorithm may get larger than the optimal objective.

Consider the instance data with 6 jobs shown in Table 3 and a 10×10 workspace. We can fit three jobs within the space, so the batches formed are $\{1, 2, 3\}$ and $\{4, 5, 6\}$ as shown in Figure 5(a). The sum of completion times before post-processing, Z_H=p_1+p_2+$4p_3$+p_4+p_5+p_6 = 92. If we were to instead schedule the batches as seen in Figure 5(b) in the sequence $\{1, 2\}$, $\{3, 4\}$, and $\{5, 6\}$, Z_H=p_1+$5p_2$+p_3+$3p_4$+p_5+p_6 = 91. Therefore, packing more jobs that are largely different in processing times because they efficiently utilize the space does not result in a lower sum of completion times objective.

Table 3. SSP instance with N=6 jobs to depict that grouping more jobs in a batch does not guarantee lower objective value

Job	Processing Time	Width	Height
1	1	2	H
2	2	4	H
3	11	2	H
4	12	4	H
5	16	2	H
6	17	4	H



(a) 3 job batch

(b) 2 job batch

Fig. 5. Example schedule with two and three jobs in a batch

Proposition 3 *For the instances defined by N=nk for any* $n, k \in \mathbb{Z}_+^*$, $w_j = \frac{W}{k}$ *and* $h_j = H$ $\forall j \in J$, *and* $p_1 \leq p_2 \leq \cdots \leq p_N$, *after using the post-processing routine (schedule_update), the sum of completion times objective* $\hat{Z} = Z_{OPT}$.

Proof: Considering the instances with $N = nk$ jobs, let C_j^H, $\hat{C}_j$, and C_j^{OPT} denote the completion time for job j and Z_H, $\hat{Z}$, and Z_{OPT} denote the objective value for the batch-scheduling algorithm, the post-processing routine, and the optimal solution respectively. First, we observe that at any given time, k jobs can simultaneously fit within the workspace, so there are n batches. So for all jobs $j \leq k$, $\hat{C}_j = C_j^{OPT}$.
Let U = $\{u_1, u_2, \cdots, u_k\}$ denote the k jobs in the next batch waiting to be scheduled, such that $p_{u_1} \leq p_{u_2} \leq \cdots \leq p_{u_k}$. Then by definition, if job j can be moved ahead in

time by say t_j units, the new completion time is given by, $\hat{C}_j = C_j^H - t_j$. Since job u_i can be processed as soon as u_{i-k} completes and space becomes available, we get the following recursive improvement on job completion times:

$\hat{C}_{u_i} = C_{u_i}^H - [(p_{u_i-1} - p_{u_i-k}) + \cdots + (p_k - p_1)]\ \forall i \in \{1, \cdots, k-1\}$ and

$\hat{C}_{u_k} = C_{u_k}^H$

So, $\hat{Z} = Z_H - \sum_{j=1}^{n} \sum_{i=1}^{j} (p_{ik} - p_{(ik-k+1)}) = Z_{OPT}$

□

3.3 Computational Analysis

In this section we provide the computational results obtained by evaluating the two proposed procedures for solving the SSP and comparing it to the optimal solution.

3.3.1 Instance Generation

We tested both the iterative model (M1) and the efficient area model (M2) on generated instances of SSP. The instance class denoted as $NnPpRr < ABCD > i$ has n= 5 or 10 jobs, processing times generated in the uniform interval of $(1, p)$ with workspace area dimension $W = H = r$. The value for r is 10 or 20 units and i is an instance indicator. A, B, C classifiers are used to indicate the distributions from which the width and height of jobs are sampled.

Class A $w_j \in$ Uniform Discrete [1,$\frac{W}{2}$] and $h_j \in$ Uniform Discrete [1, $\frac{H}{2}$]

Class B $w_j \in$ Uniform Discrete [1,$\frac{W}{2}$] and $h_j \in$ Uniform Discrete [$\frac{H}{2}$,H]

Class C $w_j \in$ Uniform Discrete [$\frac{W}{2}$,W] and $h_j \in$ Uniform Discrete [$\frac{H}{2}$,H]

Five instances of each class-type were generated, resulting in a total of 60 instances. All of the instances had jobs sorted in the increasing order of processing times. Instances in Class C have jobs that occupy more than half the area. This results in each job getting its individual batch and SSP reduces to SMS which can be solved to optimality. So for the computational analysis we only consider instances in classes A and B. By design, instances in Class B should be relatively harder to solve than instances in class A. This is because all of the jobs in class A are small compared to the dimensions of the workspace, so we can fit more jobs together. Difficult instances of the problem occur, when some jobs are small and some are large (Class B).

Larger instances were obtained from Garcia 2010. The instances have 100, 500, and 1000 jobs with a 10×7 workspace. For each job:

$w_j \in UniformDiscrete[1, 10]$

$h_j \in UniformDiscrete[1, 7]$

$p_j \in UniformDiscrete[5, 25]$

Since we did not permit rotation of jobs, we had to interchange the widths and heights in certain cases to ensure that the jobs would fit within the space.

3.3.2 Valid Values for T

Recall that T represents the maximum completion time of a schedule, which is also seen in equation (4) of the SSP IP formulation found in Chapter 2. In order to guarantee that the model solves efficiently, it is important to make an appropriate choice for the value of T, especially for large instances.

For the instance classes A and B, the values for T were determined based on bin packing. In Class A, the values for w_j and h_j in the worst-case are $\frac{W}{2}$ and $\frac{H}{2}$ respectively. So we would be require at most $N\frac{WH}{4WH}$ or $\lceil \frac{N}{4} \rceil$ bins to pack N jobs.

Since jobs also have processing times, we need to stretch the bins to accommodate the duration. Again, worst case the jobs in the bins are the last $\lceil \frac{N}{4} \rceil$ jobs. Since $p_1 < p_2 < \cdots < p_N$, class A instances have $T \leq p_{\lceil \frac{3N}{4} \rceil} + \cdots + p_N$. A similar reasoning is used for instance class B where $T \leq p_{\lceil \frac{N}{2} \rceil} + \cdots + p_N$.

For larger instances, we make use of the previously discussed bound for bin packing to determine K, the maximum number of bins required to continuously pack the jobs. Assuming, without loss of generality, that $p_1 < p_2 < \cdots < p_N$, we then have $T \leq p_{n+1-2K} + \cdots + p_N$, where

$K = \min\{ \left\lceil \frac{\sum_{j=1}^{N} h_j}{H} \right\rceil, \left\lceil \frac{\sum_{j=1}^{N} w_j}{W} \right\rceil \}$. A valid value of T was determined as the minimum of this bin packing based upper bound and sum of processing times.

3.3.3 Initial feasible solution heuristic

Table 4. Problem size comparisons between M1, M2, and original SSP MIP models for N job instances

Model	Number of Variables	Number of Constraints
M1	$5N + 2N^2$	$N^3 + 3N^2 + 4N + 1$
M2	$6N + 3N^2$	$N^3 + 3N^2 + 5N$
OPT	$3N + 3N^2$	$4N^2 - 2N$

The motivation behind creating the batching models (M1 and M2) was to reduce the size of the original SSP by looking only at the packing component of the problem. Nevertheless, we need to understand that M1 and M2 are still MIPs and as the instances grow larger, these models could take longer to solve to optimality. Table 4 shows a comparison of the number of variables and constraints between the SSP MIP formulation and the two batching models M1 and M2 for instances with N jobs. Further, an optimal solution to the batching model, does not necessarily

Algorithm 2 simple_pack

```
1: $Batch \leftarrow 1$
2: $Job \leftarrow 1$
3: while $Job \leq N$ do
4: Begin:
5:     $x,y \leftarrow 0$
6:     if $SpaceReqd[Job] \leq SpaceAvail[Batch]$ then
7: Loop X:
8:         for $i = x$ to $x + w_j$ do
9: Loop Y:
10:            for $j = y$ to $y + h_j$ do
11:                if Height units are available then
12:                    $x \leftarrow$ x + 1
13:                    goto Loop X
14:                else
15:                    if Height units are unavailable then
16:                        $y \leftarrow$ y + 1
17:                        goto Loop Y
18:                    end if
19:                end if
20:            end for
21:            if Width units are also available then
22:                Assign space to job
23:                Assign job to batch
24:            else
25:                if Width units are unavailable then
26:                    $x \leftarrow$ x + 1
27:                    goto Loop X
28:                end if
29:            end if
30:        end for
31:        if Area is available but job cannot be fit in the given space then
32:            $Batch = Batch + 1$
33:            goto Begin
34:        end if
35:    else
36:        $Batch = Batch + 1$
37:        goto Begin
38:    end if
39:    $Job \leftarrow$ Job + 1
40: end while
```

guarantee an optimal solution to SSP. In order to improve the solution time for these MIP formulations, we provide the solver with an initial feasible solution obtained from a packing heuristic (simple_pack). The pseudocode for simple_pack is presented below. Basically, we start with an instance of SSP sorted in the increasing order of job processing times, i.e. $p_1 \leq p_2 \leq \cdots \leq p_N$. We sequentially begin grouping jobs into a batch until they fit the space. Once the job can no longer fit the space, we create a new batch. This process is repeated until all jobs are assigned a batch.

3.3.4 Computational Results

In this section, we compare the solutions generated by the batch-scheduling approaches (iterative and efficient area models) to the optimal solution (OPT) obtained by solving the mixed-integer program for SSP. The batching MIPs, M1 and M2, and the SSP MIP formulation were all implemented using the C programming language and solved using Gurobi 5.0 with a thread count of 1 and cuts parameter set to default on a RedHat Enterprise 6.5 x86_64 server. The following tables compare the objective values and runtimes for the forty small instances with 5 jobs and 10 jobs and the large instances with 25 and 100 jobs (defined at the beginning of 3.3).

Table 5 lists the objective values obtained from solving instances with 5 and 10 jobs for M1 and M2 using the MAX rule and the optimal solution (OPT) for the original MIP formulation of SSP. Note that the objective reported for M1 is the best possible value among the $N-1$ potential solutions it obtains and the run time is the total time taken to iteratively solve all of the models. We observe that M2 seems to perform at least as well as M1, and both models return values close to the optimal solution. For these set of instances, the objective values returned by both models for

Table 5. Comparison of objectives obtained from M1, M2, and OPT for small instances of batch-scheduling

Instance	**M1 (Best)**	**M2**	**OPT**	**Factors**	
				M1/OPT	**M2/OPT**
N5P10R10A	27	27	27	1.00	1.00
N5P19R10B	33	33	29	1.14	1.14
N5P10R20A	26	26	26	1.00	1.00
N5P10R20B	26	25	23	1.13	1.12
N10P10R10A	49	49	49	1.00	1.00
N10P10R10B	80	72	66	1.22	1.10
N10P10R20A	54	54	51	1.05	1.05
N10P10R20B	101	88	77	1.31	1.14

the MAX and AVG rules were identical for instances with five jobs and ten jobs.

Table 6. Comparison of M1, M2, and OPT runtimes for small instances of batch-scheduling

Instance	**Runtime (seconds)**		
	M1 (Total)	**M2**	**OPT**
N5P10R10A	0.19	0.01	0.01
N5P19R10B	0.14	0.04	0.02
N5P10R20A	0.17	0.01	0.01
N5P10R20B	0.13	0.07	0.01
N10P10R10A	43.98	0.11	0.03
N10P10R10B	110.21	**82.79**	287.69
N10P10R20A	300.81	0.23	0.30
N10P10R20B	223.26	**114.17**	244.33

Table 6 presents the runtimes for solving the instances with 5 and 10 jobs using M1, M2, and the original MIP formulation. We observe that with smaller number of jobs, all three methods produce results quickly. The runtimes for M1 are larger because it iteratively solves $N-1$ models for each instance with N jobs. The solution times that are in bold face for M2 indicate that certain instances of that class took over an hour to solve. In the few cases where that occurred, the incumbent solution at the end of twenty minutes was reported.

Table 7. Comparison of objectives obtained from M1, M2, and Z_{IP} for large instances of batch-scheduling

Instance	**Objective**			**Factor**	
	M1 (Best)	**M2 (Updated)**	$\mathbf{Z_{IP}}$	$\mathbf{M1/Z_{IP}}$	$\mathbf{M2/Z_{IP}}$
N25P25E11	1697	1421	1215	1.40	1.17
N25P25E12	1518	1409	1022	1.49	1.38
N25P25E13	2204	2046	1540	1.43	1.33
N25P25E14	1555	1292	995	1.56	1.30
N25P25H11	1819	1762	1353	1.34	1.30
N25P25H12	1587	1332	965	1.64	1.38
N25P25H13	1929	1712	1169	1.65	1.46
N25P25H14	1625	1525	1066	1.52	1.43
N100P25E1	34372	24495	28205	1.22	0.87
N100P25H1	45571	24919	27672	1.65	0.90

Table 8. Comparison of M1, M2, and Z_{IP} runtimes for large instances of batch-scheduling

Instance	**Runtime (seconds)**		
	M1 (Total)	**M2 (Updated)**	$\mathbf{Z_{IP}}$
N25P25E11	1920.00	1200.00	1200.00
N25P25E12	2280.00	1200.00	1200.00
N25P25E13	1920.00	1200.00	1200.00
N25P25E14	2040.00	1200.00	1200.00
N25P25H11	1800.00	1200.00	1200.00
N25P25H12	2160.00	1200.00	1200.00
N25P25H13	2040.00	1200.00	1200.00
N25P25H14	1800.00	1200.00	1200.00
N100P25E1	>3000.00	1200.00	1200.00
N100P25H1	>3000.00	1200.00	1200.00

Tables 7 and 8 list the objective values and runtimes obtained from solving larger instances (25 and 100 jobs) for M1 and M2 using the MAX rule and the objective Z_{IP} for the original MIP formulation of SSP. Note that the objective reported for M1 is the best possible value among the $N-1$ potential solutions it obtains, with each iteration of M1 allowed two minutes of execution time. M2 and Z_{IP} report the best objective obtained after twenty minutes of execution. To improve upon the solution, M2 is given an initial feasible solution obtained using the greedy heuristic, simple_pack. The resulting solution is then updated using the post-processing algorithm. Although both models produce objectives closeito the the optimal, it is observed that with a larger number of jobs, M2 outperforms M1, and on some occasions, after twenty minutes, M2 is able to produce better solutions than the original SSP formulation.

In conclusion, the efficient area model seems to be more effective for larger instances both in terms of runtime and solution quality. Further investigations on the weights in the multi-objective function in the efficient area model (M2) could result in potential improvements in objective value. Implementing a binary search procedure for the iterative model (M1) may result in faster solution times.

CHAPTER 4

LIST-SCHEDULING APPROACH

The batch-scheduling approach gave us methods to identify batches of jobs and then schedule them using some heuristic. Since we are still solving MIPs to determine the assignment of jobs to batches, solution times can be large for larger instances. The greedy heuristic (simple_pack) that we developed aims to provide quicker results. This heuristic only looks at processing times and does not take into consideration the area of the jobs. The list-scheduling ideas presented in this chapter seek to address this observation by creating a sequence of the jobs based on both processing times and area. To gain insight into the merit of this reasoning, consider the following example instance of SSP and a 3×3 workspace.

Table 9. An example to motivate the need for list-scheduling by area and processing time

Job	Width	Height	Processing Time
1	3	3	3
2	2	2	4
3	1	2	4

If the shortest job is scheduled first, then the resulting sequence of jobs is given by $\{1, 2, 3\}$. The sum of completion times objective, $Z = 3 + 7 + 7 = 17$. If however, the smallest job is scheduled first, the sequence of the jobs is $\{3, 2, 1\}$ and the objective is $Z = 7 + 4 + 4 = 15$.

4.1 Introduction

In the past, list-scheduling algorithms have been successful in obtaining near-optimal solutions for many variants of scheduling problems (Schutten 1996; Lawler et al. 1993). These algorithms create an ordering of the jobs (list) given in each instance based on some criterion and then schedule the jobs in that sequence. The construction of the list determines the quality of solutions obtained. Smith's rule on a single machine states that if we schedule the jobs in the non-decreasing order of $\frac{p_j}{\omega_j}$, where for each job j, ω_j is the weight of the completion time, and p_j is the processing time, then, the sum of the weighted completion times is minimized (Smith 1956). If we extend this rule for multiple machines, we can use it to determine the ordering of the jobs for each individual machine. The idea is that, by scheduling the shortest jobs first, more jobs can finish early and result in a smaller sum. However, with spatial resources, a direct implementation of this rule is not possible. Nevertheless, we can use the same rationale and try to complete more jobs earlier in the schedule. With this in mind, we consider the following two variations to create an ordering of the jobs:

Recall that h_j and w_j are the height and width of each job $j \in J$ with processing time p_j. Then,

(a) Product-based approach

The jobs are arranged in a non-decreasing sequence of the product $p_j w_j h_j$. This corresponds to scheduling smaller areas and processing times first.

(b) Ratio-based approach

The jobs are arranged in a non-decreasing sequence of the ratio $\frac{p_j}{w_j h_j}$. This corresponds to scheduling larger areas and smaller processing times first.

The product rule schedules the short (processing time) and small (area) jobs early. By scheduling smaller jobs ahead of larger jobs we can fit more jobs in the workspace. On the other hand, the ratio rule gives preference to jobs that are both short and large over jobs that do not possess these attributes. The rationale for considering the ratio approach is to extend the Smith's rule idea by treating the area of the job as a weight on the completion time of the job. In the subsequent sections we show that under certain assumptions the product rule outperforms the ratio rule.

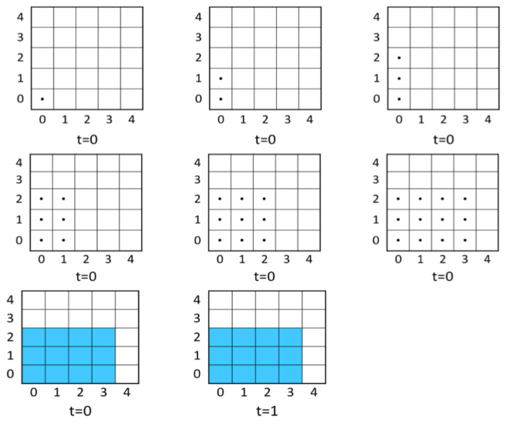


Fig. 6. List-scheduling algorithm attempts to find the same four contiguous width and three contiguous height units for two consecutive time slots

4.2 Generating the schedule

The pseudocode for the list-scheduling algorithm is presented below. Basically, we start with an instance of SSP and create a list of jobs by sorting them in the increasing order of their products (or ratios) of processing times and area. The jobs are then scheduled in the determined sequence using a First Fit packing strategy. For each job, we determine if there exists contiguous units of width and height for consecutive time slots in the scheduling horizon (see Figure 6). If such a space exists, we schedule the job in that time and space. This process is repeated until all jobs are scheduled. Finally, the sum of completion times objective is calculated.

Algorithm 3 list_schedule

```
1:  $TimeSlot \leftarrow 1$
2:  $index \leftarrow 1$
3:  while $index \leq N$ do
4:      $Job \leftarrow SchedulingOrder[index]$
5:  Begin:
6:      $x,y \leftarrow 0$
7:      if $SpaceReqd[Job] \leq SpaceAvail[TimeSlot]$ then
8:  Loop X:
9:          for $i = x$ to $x + w_j$ do
10: Loop Y:
11:             for $j = y$ to $y + h_j$ do
12:                 if Height units are available then
13:                     $x \leftarrow$ x + 1
14:                     goto Loop X
15:                 else
16:                     $y \leftarrow$ y + 1
17:                     goto Loop Y
18:                 end if
19:             end for
20:             if Width units are also available then
21:                 $TimeSlot \leftarrow$ TimeSlot + 1
22:                 goto Loop X
23:             else
24:                 $TimeSlot \leftarrow$ TimeSlot + 1
25:                 goto Begin
26:             end if
27:         end for
28:         if Same contiguous space is available for all processing time units then
29:             Assign space to job
30:         else
31:             $TimeSlot = TimeSlot + 1$
32:             goto Begin
33:         end if
34:     end if
35:     $index \leftarrow$ index + 1
36: end while
```

4.3 Performance Analysis

In this section, we want to analyze special cases and characterize instances where one version of the list-scheduling algorithm outperforms the other. In order to do that let us first define a_j=$w_j h_j$ as the area occupied by job $j \in J$. In all of the cases we consider, we assume that the processing time (p_j) is a function of the area (a_j) for each job $j \in J$. In the context of large-scale assembling and manufacturing, since larger jobs typically take longer time to process, our assumption is reasonable.

Case 1 : When p_j= βa_j and $\beta > 1$, $\forall\ j \in J$

For any job $j \in J$ the ratio ,$\frac{p_j}{a_j} = \beta$, is constant and the ordering of the jobs in the list is arbitrary.

On the other hand, the product, $p_j a_j = \beta\ a_j^2$, provides more useful information that will help schedule the jobs.

Case 2 : When p_j=$\beta a_j + x$, $\forall\ j \in J$

Proposition 4 *When p_j=$\beta a_j + x$, $\forall\ j \in J$, $x > 0$, and $\beta > 1$, both versions of the algorithm produce different ordering of jobs. Specifically, each algorithm produces the exact reverse ordering of the other version.*

Proof: Let job i be scheduled before job j using the ratio version of the algorithm.

$$\begin{aligned}
\Longleftrightarrow \frac{p_i}{a_i} &\leq \frac{p_j}{a_j} \\
\Longleftrightarrow \frac{\beta a_i + x}{a_i} &\leq \frac{\beta a_j + x}{a_j} \\
\Longleftrightarrow \frac{x}{a_i} &\leq \frac{x}{a_j} \\
\Longleftrightarrow \frac{1}{a_i} &\leq \frac{1}{a_j} \\
\Longleftrightarrow a_j &\leq a_i \\
\Longleftrightarrow \beta a_j + x &\leq \beta a_i + x \\
\Longleftrightarrow (\beta a_j + x)a_j &\leq (\beta a_i + x)a_i \\
\Longleftrightarrow p_j a_j &\leq p_i a_i
\end{aligned}$$

This implies that job j is scheduled before job i in the product version of the algorithm. □

While looking at proposition 4 we observe that the following statements are true:

(i) When $a_1 < a_2 < ... < a_n$ then $p_1 < p_2 < ... < p_n$

(ii) The sum of completion times objective using the product version of the list-scheduling algorithm is the same as that obtained using the ratio version when all the jobs can simultaneously fit in the space. The objective value can be obtained using: $\sum_{j \in J} C_j = \sum_{j \in J} p_j$

(iii) The sum of completion times objective using the product version of the list-scheduling algorithm is better (smaller) than that obtained using the ratio version of the algorithm when only one job can fit the space at any given time. The problem reduces to single machine scheduling and the objective value can be obtained using: $\sum_{j \in J} C_j = \sum_{j=1}^{n} \sum_{i=1}^{j} p_j$

(iv) The ordering of the jobs produced using the ratio version of the list-scheduling algorithm is the exact reverse of the ordering produced by the product version when only one job can fit the space at any given time. So, if the ordering produced by the product version is job 1, job 2, job 3, ..., job n; then the ordering obtained from the ratio version is job n, ..., job 1. The difference in the objective values is given by:

Objective value of Ratio version - Objective value of Product version, or

$(np_n + (n-1)p_{n-1} + ... + 2p_2 + p_1) - (np_1 + (n-1)p_2 + ... + 2p_{n-1} + p_n)$

Proposition 5 *When $p_j = \beta a_j + x$, $\forall\, j \in J$, $x < 0$, and $\beta > 1$, both versions of the algorithm produce the same ordering of jobs.*

Proof: Let job i be scheduled before job j using the ratio version of the algorithm.

$$\begin{aligned}
\iff \frac{p_i}{a_i} &\leq \frac{p_j}{a_j} \\
\iff \frac{\beta a_i - x}{a_i} &\leq \frac{\beta a_j - x}{a_j} \\
\iff \frac{-x}{a_i} &\leq \frac{-x}{a_j} \\
\iff \frac{-1}{a_i} &\leq \frac{-1}{a_j} \\
\iff a_i &\leq a_j \\
\iff \beta a_i - x &\leq \beta a_j - x \\
\iff (\beta a_i - x)a_i &\leq (\beta a_j - x)a_j \\
\iff p_i a_i &\leq p_j a_j
\end{aligned}$$

This implies that job i is scheduled before job j using the product version of the algorithm too. □

From these analyses, we conclude that the product-based list-scheduling is more reliable than the ratio-based algorithm. In the subsequent section, we provide upper bounds for special cases of the problem using the product-based list-scheduling.

4.4 Upper bounds

In this section we attempt to prove approximation factors for special cases of SSP where for each job j, the processing time (p_j) is a function t of the area (a_j) for the product-based list-scheduling algorithm.

Theorem 2 *When $w_j \geq \lceil \frac{W}{2} \rceil$, $h_j \geq \lceil \frac{H}{2} \rceil$, and p_j is a function of the area $(w_j h_j)$ $\forall$ jobs $j \in J$, the optimal solution for the sum of completion times objective is obtained using the product-based list-scheduling algorithm.*

Proof: Since, the width and height of the space for each job is at least half the width and height of the workspace, only one job can occupy the space at any given time. When we order the jobs based on the increasing order of $p_j w_j h_j$, the ordering depends only on p_j. The LS algorithm reduces to Shortest Processing Time (SPT) algorithm and SSP reduces to Single Machine Scheduling (SMS). We know that SPT finds the optimal solution for SMS (Smith 1956). Thus, the optimal solution can be found using the list-scheduling algorithm. $\square$

4.5 Computational Results

In this section, we compare the objective values generated by the product-based list-scheduling (LS) algorithm to the solution Z_{IP}, obtained by solving the original MIP for SSP, and the greedy heuristic (simple_pack). The greedy heuristic and list-scheduling algorithm were implemented using the C programming language. The SSP MIP formulation was implemented in C and solved using Gurobi 5.0 with a thread count of 1 and cuts parameter set to default setting on a RedHat Enterprise 6.5 x86_64 server.

Table 10. Comparing the objective values obtained from greedy heuristic, list-scheduling product version, and Z_{IP} for instances of SSP

Instance	**Objective**			**Factor**	
	Greedy Heuristic	**LS (Product)**	$\mathbf{Z_{IP}}$	**Greedy/**Z_{IP}	**LS/**Z_{IP}
N5P10R10A	27	27	27	1.00	1.00
N5P10R10B	33	30	29	1.07	1.06
N5P10R20A	26	26	26	1.00	1.00
N5P10R20B	26	24	23	1.05	1.08
N10P10R10A	49	49	49	1.00	1.00
N10P10R10B	82	78	66	1.25	1.20
N10P10R20A	58	53	51	1.14	1.04
N10P10R20B	96	91	77	1.24	1.18
N25P25E11	2103	1828	1215	1.73	1.50
N25P25E12	1498	1465	1022	1.47	1.43
N25P25E13	3345	2142	1540	2.17	1.39
N25P25E14	1716	1410	995	1.72	1.42
N25P25H11	2112	1785	1353	1.56	1.32
N25P25H12	1572	1727	965	1.63	1.79
N25P25H13	1741	1939	1169	1.49	1.66
N25P25H14	1629	1557	1066	1.53	1.46
N100P25E1	26877	23528	28205	0.95	0.83
N100P25H1	28914	22920	27672	1.04	0.83
N500P25E1	715783	542514	-	-	-
N500P25H1	699072	577640	-	-	-
N1000P25E1	2925853	2201137	-	-	-
N1000P25H1	2819374	2116875	-	-	-

Tables 10 and 11 list the objective values and runtimes obtained from solving instances with 5, 10, 25, 100, 500, and 1000 jobs using the product based list-scheduling problem, the greedy algorithm and the original MIP formulation of SSP. Values for Z_{IP} denote an optimal objective (5 job instances) or an incumbent solution at the end of twenty minutes (10 and 25 job instances) of execution. Missing values for Z_{IP} indicates that no solution was found until the time limit of thirty minutes.

Table 11. Comparing runtimes for greedy heuristic, list-scheduling algorithm, and Z_{IP} while solving instances of SSP

Instance	**Runtime (seconds)**		
	Greedy Heuristic	**LS (Product)**	$\mathbf{Z_{IP}}$
N5P10R10A	< 0.01	< 0.01	0.009
N5P10R10B	< 0.01	< 0.01	0.018
N5P10R20A	< 0.01	< 0.01	0.005
N5P10R20B	< 0.01	< 0.01	0.008
N10P10R10A	< 0.01	< 0.01	0.03
N10P10R10B	< 0.01	< 0.01	287.69
N10P10R20A	< 0.01	< 0.01	0.30
N10P10R20B	< 0.01	< 0.01	244.33
N25P25E11	< 0.01	0.01	1200.00
N25P25E12	< 0.01	0.01	1200.00
N25P25E13	< 0.01	0.01	1200.00
N25P25E14	< 0.01	< 0.01	1200.00
N25P25H11	< 0.01	0.01	1200.00
N25P25H12	< 0.01	0.01	1200.00
N25P25H13	< 0.01	0.01	1200.00
N25P25H14	< 0.01	< 0.01	1200.00
N100P25E1	< 0.01	0.10	1200.00
N100P25H1	< 0.01	0.08	1200.00
N500P25E1	< 0.01	1.50	1800.00
N500P25H1	< 0.01	1.52	1800.00
N1000P25E1	< 0.01	5.28	1800.00
N1000P25H1	< 0.01	5.08	1800.00

The results indicate that the product version of the list-scheduling algorithm outperforms the greedy heuristic in most cases. However, the few instances where the greedy algorithm results in a better objective can be attributed to scenarios when jobs have large areas (almost equal to the workspace area). In such cases, the processing times become more dominant and scheduling using the greedy algorithm can result in a better objective value.

It is clear that as the number of jobs increases, the MIP formulations become harder to solve but both heuristics produce quick results within a reasonable guarantee. The factors with values less that one indicate that the heuristics result in a solution that is better than the one reported by the MIP in twenty minutes. In conclusion, the product version of list-scheduling seems to be the solution method to use when the number of jobs is large and a quick result is required.

CHAPTER 5

CONCLUSIONS

The focus of this thesis has been to study the spatial scheduling problem and attempt to develop solution methods with good approximations for the minimum sum of completion times objective. We conclude by summarizing the main contributions and key results presented and by suggesting possible directions for future research. We pared the problem into its simplest form to gain a better understanding of the relationship between the spatial and temporal components of the problem. We exploited these components individually in the design of our algorithms. First, we consider just the spatial restrictions and utilize bin-packing strategies to identify batches of jobs that will efficiently utilize the space. We then schedule the jobs using rules to minimize the sum of completion times objective. We prove an approximation factor under certain conditions and also identify scenarios when grouping jobs does not necessarily result in a better objective. We also give a post-processing algorithm to improve the objective value of the batching models, which results in optimal solutions for certain instances. Second, we adopt a popular scheduling idea based on Smiths rule (Smith 1956) and create a list-scheduling algorithm. We prove an approximation factor under certain assumptions and provide computational results that demonstrate the performance of our algorithm in practice on large instances of the problem. Based on the instances we tested for both approaches, our assessment is that scheduling jobs similar in processing times within the same space yields good solutions. If processing times are sufficiently different, then grouping jobs together because they effectively utilize the space does not necessarily result in a lower sum of completion times. Among the batching models, the efficient area model outperforms the iterative model both in terms of solution quality and run time. For larger instances, we show that the list-scheduling algorithm provides quick and reasonable approximations.

Directions for future research are plentiful. We provide two MIP formulations to decide the assignment of jobs to batches, the iterative and efficient area model. Currently, we solve at most $N-1$ instances for the iterative procedure and weigh the two objectives in the efficient area model equally. Possible enhancements to these models could be to implement a binary search procedure for the iterative model and tweak the weights in the multi-objective efficient area model. This study assumes that a single spatial resource of fixed dimension is available. An interesting extension would be to look at multiple workspace problems with varying area. We may be able to use ideas from variable size bin packing to design algorithms for this problem. Another area that merits investigation is to consider weights on the completion times of the jobs. If l_j is the weight on completion time for job $j \in J$, and we assign the number of jobs in the batch containing job j as a weight on its completion time, can we get similar results for our procedures? The research opportunities mentioned here focus on the general MIP formulation for SSP. Can the performance of the solution methods described be replicated when compared to the time-indexed formulation provided in Chapter 2? Additionally, polyhedral studies of scheduling problems have provided useful insights into the mathematical structure of the problem (Queyranne and Schulz 1994). Performing such a study of SSP, possibly using the time-indexed formulation, may reveal ideas that help us design stronger formulations and efficient solution strategies. Lastly, although the results and analyses presented in this study pertain to the sum of completion times objective, the solution methods developed here can easily be applied to other objective functions of the problem. If we can show that these approaches are effective in solving a broader array of objectives, then we increase the value of our work.

Appendix A

BATCH-SCHEDULING HEURISTIC PERFORMANCE

Table 12. Comparison of objective values obtained using M1 and M2 with Z_{IP} for SSP instances with 5 jobs

Class	**Instance**	**Objective**			**Factor**	
		M1	**M2**	$\mathbf{Z_{IP}}$	$\mathbf{M1/Z_{IP}}$	$\mathbf{M2/Z_{IP}}$
A	N5P10R10A0	31	31	31	1.00	1.00
	N5P10R10A1	34	34	34	1.00	1.00
	N5P10R10A2	16	16	16	1.00	1.00
	N5P10R10A3	21	21	21	1.00	1.00
	N5P10R10A4	31	31	31	1.00	1.00
	N5P10R10A	**26.6**	**26.6**	**26.6**	**1.00**	**1.00**
B	N5P10R10B0	36	36	29	1.24	1.24
	N5P10R10B1	44	44	36	1.22	1.22
	N5P10R10B2	36	36	32	1.13	1.13
	N5P10R10B3	24	24	24	1.00	1.00
	N5P10R10B4	24	24	23	1.04	1.04
	N5P10R10B	**32.8**	**32.8**	**28.8**	**1.14**	**1.14**
A	N5P10R20A0	21	21	21	1.00	1.00
	N5P10R20A1	27	27	27	1.00	1.00
	N5P10R20A2	31	31	31	1.00	1.00
	N5P10R20A3	21	21	21	1.00	1.00
	N5P10R20A4	30	30	30	1.00	1.00
	N5P10R20A	**26**	**26**	**26**	**1.00**	**1.00**
B	N5P10R20B0	28	28	26	1.08	1.08
	N5P10R20B1	25	25	25	1.00	1.00
	N5P10R20B2	21	19	16	1.31	1.19
	N5P10R20B3	33	33	28	1.18	1.18
	N5P10R20B4	21	21	18	1.17	1.17
	N5P10R20B	**25.6**	**25.2**	**22.6**	**1.13**	**1.12**

Table 13. Comparison of objective values obtained using M1 and M2 with Z_{IP} for SSP instances with 5 jobs

Class	**Instance**	**Objective**			**Factor**	
		M1 (Best)	**M2 (Updated)**	$\mathbf{Z_{IP}}$	$\mathbf{M1/Z_{IP}}$	$\mathbf{M2/Z_{IP}}$
A	N10P10R10A0	49	49	49	1.00	1.00
	N10P10R10A1	48	48	48	1.00	1.00
	N10P10R10A2	56	56	56	1.00	1.00
	N10P10R10A3	50	50	50	1.00	1.00
	N10P10R10A4	42	42	42	1.00	1.00
	N10P10R10A	**49**	**49**	**49**	**1.00**	**1.00**
B	N10P10R10B0	83	79	72	1.15	1.10
	N10P10R10B1	111	101	92	1.21	1.10
	N10P10R10B2	89	79	67	1.33	1.18
	N10P10R10B3	78	67	62	1.26	1.08
	N10P10R10B4	39	35	35	1.11	1.00
	N10P10R10B	**80**	**72**	**66**	**1.21**	**1.09**
A	N10P10R20A0	58	58	58	1.00	1.00
	N10P10R20A1	54	54	54	1.00	1.00
	N10P10R20A2	51	51	51	1.00	1.00
	N10P10R20A3	33	33	33	1.00	1.00
	N10P10R20A4	73	73	60	1.22	1.22
	N10P10R20A	**54**	**54**	**51**	**1.04**	**1.04**
B	N10P10R20B0	127	111	94	1.35	1.18
	N10P10R20B1	75	69	63	1.19	1.10
	N10P10R20B2	110	107	88	1.25	1.22
	N10P10R20B3	108	83	78	1.38	1.06
	N10P10R20B4	84	69	63	1.33	1.10
	N10P10R20B	**101**	**88**	**77**	**1.30**	**1.13**

Table 14. Runtimes in seconds for M1, M2, and Z_{IP} when solving SSP instances with 5 jobs

Class	**Instance**	**Runtime (seconds)**		
		M1	**M2**	$\mathbf{Z_{IP}}$
A	N5P10R10A0	0.15	0.01	0.01
	N5P10R10A1	0.13	0.01	0.01
	N5P10R10A2	0.17	0.01	0.00
	N5P10R10A3	0.28	0.01	0.01
	N5P10R10A4	0.23	0.01	0.01
	N5P10R10A	**0.19**	**0.01**	**0.01**
B	N5P10R10B0	0.11	0.04	0.03
	N5P10R10B1	0.15	0.09	0.02
	N5P10R10B2	0.11	0.03	0.02
	N5P10R10B3	0.15	0.01	0.01
	N5P10R10B4	0.17	0.04	0.01
	N5P19R10B	**0.14**	**0.04**	**0.02**
A	N5P10R20A0	0.14	0.01	0.00
	N5P10R20A1	0.15	0.01	0.00
	N5P10R20A2	0.23	0.01	0.00
	N5P10R20A3	0.16	0.01	0.00
	N5P10R20A4	0.18	0.01	0.01
	N5P10R20A	**0.17**	**0.01**	**0.01**
B	N5P10R20B0	0.16	0.03	0.01
	N5P10R20B1	0.13	0.01	0.00
	N5P10R20B2	0.09	0.24	0.01
	N5P10R20B3	0.16	0.06	0.01
	N5P10R20B4	0.12	0.04	0.01
	N5P10R20B	**0.13**	**0.07**	**0.01**

Table 15. Runtimes in seconds for M1, M2, and Z_{IP} when solving SSP instances with 10 jobs

Class	**Instance**	**Runtime (seconds)**		
		M1	**M2**	**Z_{IP}**
A	N10P10R10A0	28.59	0.10	0.03
	N10P10R10A1	21.14	0.11	0.04
	N10P10R10A2	85.68	0.10	0.02
	N10P10R10A3	28.85	0.12	0.04
	N10P10R10A4	55.62	0.10	0.02
	N10P10R10A	**43.98**	**0.11**	**0.03**
B	N10P10R10B0	91.40	0.66	22.86
	N10P10R10B1	49.66	200.95	216.91
	N10P10R10B2	230.57	128.48	5.43
	N10P10R10B3	19.97	>3600	1107.26
	N10P10R10B4	159.47	1.06	86.01
	N10P10R10B	**110.21**	**82.79**	**287.69**
A	N10P10R20A0	27.67	0.11	0.03
	N10P10R20A1	143.91	0.12	0.04
	N10P10R20A2	54.85	0.10	0.02
	N10P10R20A3	1219.04	0.10	0.03
	N10P10R20A4	58.57	0.69	1.38
	N10P10R20A	**300.81**	**0.23**	**0.30**
B	N10P10R20B0	150.99	>3600	1093.63
	N10P10R20B1	729.56	205.71	12.70
	N10P10R20B2	74.82	26.89	30.77
	N10P10R20B3	62.31	>3600	52.98
	N10P10R20B4	98.62	109.92	31.59
	N10P10R20B	**223.26**	**114.17**	**244.33**

Appendix B

LIST-SCHEDULING HEURISTIC PERFORMANCE

Table 16. Comparison of objective values obtained using greedy heuristic and product version of list-scheduling algorithm with Z_{IP} for SSP instances with 5 jobs

Instance	**Greedy Heuristic**	**LS (Product)**	$\mathbf{Z_{IP}}$
N5P10R10A0	31	31	31
N5P10R10A1	34	34	34
N5P10R10A2	16	16	16
N5P10R10A3	21	21	21
N5P10R10A4	31	31	31
N5P10R10A	**27**	**27**	**27**
N5P10R10B0	34	32	29
N5P10R10B1	40	36	36
N5P10R10B2	34	33	32
N5P10R10B3	24	24	24
N5P10R10B4	31	27	23
N5P10R10B	**33**	**30**	**29**
N5P10R20A0	21	21	21
N5P10R20A1	27	27	27
N5P10R20A2	31	31	31
N5P10R20A3	21	21	21
N5P10R20A4	30	30	30
N5P10R20A	**26**	**26**	**26**
N5P10R20B0	28	31	26
N5P10R20B1	25	25	25
N5P10R20B2	19	16	16
N5P10R20B3	33	31	28
N5P10R20B4	23	19	18
N5P10R20B	**26**	**24**	**23**

Table 17. Comparison of objective values obtained using greedy heuristic and product version of list-scheduling algorithm with Z_{IP} for SSP instances with 10 jobs

Instance	**Greedy Heuristic**	**LS (Product)**	$\mathbf{Z_{IP}}$
N10P10R10A0	49	49	49
N10P10R10A1	48	48	48
N10P10R10A2	56	56	56
N10P10R10A3	50	50	50
N10P10R10A4	42	42	42
N10P10R10A	**49**	**49**	**49**
N10P10R10B0	83	93	72
N10P10R10B1	122	116	92
N10P10R10B2	90	76	67
N10P10R10B3	73	70	62
N10P10R10B4	41	37	35
N10P10R10B	**82**	**78**	**66**
N10P10R20A0	58	58	58
N10P10R20A1	63	56	54
N10P10R20A2	51	51	51
N10P10R20A3	33	33	33
N10P10R20A4	87	68	60
N10P10R20A	**58**	**53**	**51**
N10P10R20B0	125	107	94
N10P10R20B1	70	68	63
N10P10R20B2	110	100	88
N10P10R20B3	95	108	78
N10P10R20B4	80	72	63
N10P10R20B	**96**	**91**	**77**

REFERENCES

Akker, JM Van den, Cor AJ Hurkens, and Martin WP Savelsbergh (2000). “Time-indexed formulations for machine scheduling problems: Column generation”. In: *INFORMS Journal on Computing* 12.2, pp. 111–124.

Akker, JM Van den, CPM Van Hoesel, and Mathieu Willem Paul Savelsbergh (1999). “A polyhedral approach to single-machine scheduling problems”. In: *Mathematical Programming* 85.3, pp. 541–572.

Baker, Kenneth R. (1974). *Introduction to sequencing and scheduling.* Vol. 15. Wiley New York.

Boston, Kevin and Pete Bettinger (2001). “The economic impact of green-up constraints in the southeastern United States”. In: *Forest Ecology and Management* 145.3, pp. 191–202.

BoWang, Chen and Klaus v Gadow (2002). “Timber harvest planning with spatial objectives, using the method of simulated annealing”. In: *Forstwissenschaftliches Centralblatt vereinigt mit Tharandter forstliches Jahrbuch* 121.1, pp. 25–34.

Brucker, Peter (2001). *Scheduling Algorithms.* 3rd. Secaucus, NJ, USA: Springer-Verlag New York, Inc. ISBN: 3540415106.

Brucker, Peter et al. (1998). “Batch scheduling with deadlines on parallel machines”. In: *Annals of Operations Research* 83, pp. 23–40.

Caprace, J.-D. et al. (2013). “Optimization of shipyard space allocation and scheduling using a heuristic algorithm”. In: *Journal of Marine Science and Technology* 18.3, pp. 404–417.

Cho, K.K. et al. (2001). “A Spatial Scheduling System for Block Painting Process in Shipbuilding”. In: *CIRP Annals - Manufacturing Technology* 50.1, pp. 339 –342.

Coffman Jr., E., M. Garey, and D. Johnson (1978). “An Application of Bin-Packing to Multiprocessor Scheduling”. In: *SIAM Journal on Computing* 7.1, pp. 1–17.

Coffman Jr, EG (1976). “Computer and Job-Scheduling Theory”. In: *New York: Wiley* 31, pp. 55–66.

Dantzig, George Bernard (1965). *Linear programming and extensions.* Princeton university press.

Duin, C.W. and E.Van Sluis (2006). “On the Complexity of Adjacent Resource Scheduling”. In: *Journal of Scheduling* 9.1, pp. 49–62.

Dyer, Martin E and Laurence A Wolsey (1990). “Formulating the single machine sequencing problem with release dates as a mixed integer program”. In: *Discrete Applied Mathematics* 26.2, pp. 255–270.

Garcia, Christopher and Ghaith Rabadi (2011). “A Meta-RaPS algorithm for spatial scheduling with release times”. In: *International Journal of Planning and Scheduling* 1 (1), pp. 19–31.

Garcia, Christopher J. (2010). “Optimization Models and Algorithms for Spatial Scheduling”. PhD thesis. Norfolk, VA, USA: Old Dominion University.

Garey, Michael R. and David S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York, NY, USA: W. H. Freeman & Co. ISBN: 0716710447.

Garey, Michael R, David S Johnson, and Ravi Sethi (1976). “The complexity of flowshop and jobshop scheduling”. In: *Mathematics of operations research* 1.2, pp. 117–129.

Goemans, Michel X (1997). “Improved approximation algorthims for scheduling with release dates”. In: *Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms.* Society for Industrial and Applied Mathematics, pp. 591–598.

Graham, Ronald L (1966). “Bounds for certain multiprocessing anomalies”. In: *Bell System Technical Journal* 45.9, pp. 1563–1581.

Graham, Ronald L et al. (1979). “Optimization and approximation in deterministic sequencing and scheduling: a survey”. In: *Annals of discrete mathematics* 5, pp. 287–326.

Hall, Leslie A et al. (1997). “Scheduling to minimize average completion time: Off-line and on-line approximation algorithms”. In: *Mathematics of Operations Research* 22.3, pp. 513–544.

Hardin, Jill R, George L Nemhauser, and Martin W P Savelsbergh (2008). “Strong valid inequalities for the resource-constrained scheduling problem with uniform resource requirements”. In: *Discrete Optimization* 5.1, pp. 19–35.

Hartmann, S. (2000). “Packing problems and project scheduling models: an integrating perspective”. In: *Journal of the Operational Research Society* 51.9, pp. 1083–1092.

Hochbaum, Dorit S., ed. (1997). *Approximation Algorithms for NP-hard Problems.* Boston, MA, USA: PWS Publishing Co. ISBN: 0-534-94968-1.

Johnson, David S (1974). “Approximation algorithms for combinatorial problems”. In: *Journal of Computer and System Sciences* 9.3, pp. 256–278.

Karmarkar, Narendra (1984). “A new polynomial-time algorithm for linear programming”. In: *Proceedings of the sixteenth annual ACM symposium on Theory of computing.* ACM, pp. 302–311.

Koh, Shiegheun et al. (2011). “Spatial scheduling for shape-changing mega-blocks in a shipbuilding company”. In: *International Journal of Production Research* 49.23, pp. 7135–7149.

Kolisch, R. (2000). “Integrated scheduling, assembly area and part-assignment for large-scale, make-to-order assemblies”. In: *International Journal of Production Economics* 64.13, pp. 127 –141.

Land, Ailsa H and Alison G Doig (1960). “An automatic method of solving discrete programming problems”. In: *Econometrica* 28.3, pp. 497–520.

Lawler, Eugene L et al. (1993). “Sequencing and scheduling: Algorithms and complexity”. In: *Handbooks in operations research and management science* 4, pp. 445–522.

Lee, Chung-Yee, Reha Uzsoy, and Louis A Martin-Vega (1992). “Efficient algorithms for scheduling semiconductor burn-in operations”. In: *Operations Research* 40.4, pp. 764–775.

Lee, Kyu et al. (1997). “Developing scheduling systems for Daewoo Shipbuilding: {DAS} project”. In: *European Journal of Operational Research* 97.2, pp. 380 – 395.

Lee, Youngho and Hanif D Sherali (1994). “Unrelated machine scheduling with time-window and machine downtime constraints: An application to a naval battle-group problem”. In: *Annals of Operations Research* 50.1, pp. 339–365.

Lee, Yusin and Chuen-Yih Chen (2009). “An optimization heuristic for the berth scheduling problem”. In: *European Journal of Operational Research* 196.2, pp. 500–508.

Lenstra, Jan Karel, AHG Rinnooy Kan, and Peter Brucker (1977). “Complexity of machine scheduling problems”. In: *Annals of discrete mathematics* 1, pp. 343–362.

Leung, Joseph, Laurie Kelly, and James H. Anderson (2004). *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Boca Raton, FL, USA: CRC Press, Inc.

Lodi, Andrea, Silvano Martello, and Michele Monaci (2002). “Two-dimensional packing problems: A survey”. In: *European Journal of Operational Research* 141.2, pp. 241 –252.

Mathirajan, M and AI Sivakumar (2006). “A literature review, classification and simple meta-analysis on scheduling of batch processors in semiconductor”. In: *The International Journal of Advanced Manufacturing Technology* 29.9-10, pp. 990–1001.

Monaci, Michele (2003). “Algorithms for packing and scheduling problems”. In: *Quarterly Journal of the Belgian, French and Italian Operations Research Societies* 1.1, pp. 85–87.

Mullen, David S and Ralph M Butler (2000). “The design of a genetic algorithm based spatially constrained timber harvest scheduling model”. In: *UNITED STATES DEPARTMENT OF AGRICULTURE FOREST SERVICE GENERAL TECHNICAL REPORT NC*, pp. 57–65.

Nemhauser, George L and Laurence A Wolsey (1988). *Integer and combinatorial optimization*. Vol. 18. Wiley New York.

Park, Kyungchul et al. (1996). “Modeling and solving the spatial block scheduling problem in a shipbuilding company”. In: *Computers & Industrial Engineering* 30.3, pp. 357–364.

Perng, Chyuan, Yi-Chiuan Lai, and Zih-Ping Ho (2009). “A Space Allocation Algorithm for Minimal Early and Tardy Costs in Space Scheduling”. In: *New Trends in Information and Service Science, 2009. NISS ’09. International Conference on*, pp. 33–36.

Phillips, Cynthia, Clifford Stein, and Joel Wein (1998). “Minimizing average completion time in the presence of release dates”. In: *Mathematical Programming* 82.1-2, pp. 199–223.

Pinedo, Michael L. (2008). *Scheduling: Theory, Algorithms, and Systems.* 3rd. Springer Publishing Company, Incorporated. ISBN: 0387789340, 9780387789347.

Pisinger, David and Mikkel Sigurd (2005). “The two-dimensional bin packing problem with variable bin sizes and costs”. In: *Discrete Optimization* 2.2, pp. 154 –167.

Queyranne, Maurice and Andreas S Schulz (1994). *Polyhedral approaches to machine scheduling.* Citeseer.

Raj, Piyush and Rajiv K. Srivastava (2007). “Analytical and heuristic approaches for solving the spatial scheduling problem”. In: *Industrial Engineering and Engineering Management, 2007 IEEE International Conference on*, pp. 1093–1097.

Savelsbergh, Martin WP, RN Uma, and Joel Wein (1998). “An experimental study of LP-based approximation algorithms for scheduling problems”. In: *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms.* Society for Industrial and Applied Mathematics, pp. 453–462.

Schutten, JMJ (1996). “List scheduling revisited”. In: *Operations Research Letters* 18.4, pp. 167–170.

Smith, Wayne E (1956). “Various optimizers for single-stage production”. In: *Naval Research Logistics Quarterly* 3.1-2, pp. 59–66.

Sousa, Jorge P and Laurence A Wolsey (1992). “A time indexed formulation of non-preemptive single machine scheduling problems”. In: *Mathematical programming* 54.1-3, pp. 353–367.

Vazirani, Vijay V (2001). *Approximation algorithms.* springer.

Williamson, David P and David B Shmoys (2010). “The Design of Approximation Algorithms. 2010”. In: *preprint http://www. designofapproxalgs. com.*

Zhang, Zhiying and Jie Chen (2012). “Solving the spatial scheduling problem: a two-stage approach”. In: *International Journal of Production Research* 50.10, pp. 2732–2743.

Zheng, Junli et al. (2011). “Spatial scheduling algorithm minimising makespan at block assembly shop in shipbuilding”. In: *International Journal of Production Research* 49.8, pp. 2351–2371.

VITA

Sudharshana Srinivasan was born in Chennai, Tamil Nadu on December 9, 1985 to R. Srinivasan and Saradha Mai. After graduating from Chettinad Vidyashram in 2003, she attended Valliammai Engineering College (an affiliate of Anna University) to earn a bachelors in Computer Science and Engineering. In 2007, she moved to Richmond, Virginia to pursue a masters degree in Applied Mathematics from Virginia Commonwealth University (VCU). Upon graduation, she was offered an opportunity to continue her education in the Department of Statistics and Operations Research at VCU. In May of 2014, Sudharshana was awarded the Doctor of Philosophy in Systems Modeling and Analysis.