Exact and Heuristic Approaches to Batching-and-Scheduling for the Composites Manufacturing Problem

by

Tanya Y. Tang

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science
Graduate Department of Mechanical and Industrial Engineering
University of Toronto

Abstract

Exact and Heuristic Approaches to Batching-and-Scheduling for the Composites Manufacturing Problem

Tanya Y. Tang
Master of Applied Science
Graduate Department of Mechanical and Industrial Engineering
University of Toronto
2020

The process of manufacturing composite materials is a complex problem that lacks robust and scalable solutions. This thesis aims to characterize that process and offer solutions by defining and solving a novel optimization problem, the Composites Manufacturing Problem (CMP), and its abstraction, the Two-Stage Bin Packing and Hybrid Flowshop Scheduling Problem (2BPHFSP). The CMP involves batching and scheduling jobs in a four-stage flowshop with time-varying resource requirements. The 2BPHFSP offers a simplified version of the CMP which allows us to focus on the key complexities of the CMP and deepens our understanding of the problem. By extending state-of-the-art techniques from a wide literature body, we construct solution approaches using techniques from Mixed Integer Programming, Constraint Programming, Logic-Based Benders Decomposition, heuristic algorithms, genetic algorithms, and constrained-clustering algorithms. Empirical analyses were performed for both problems on randomly generated instances based on real data from our industrial collaborator.

Acknowledgements

I would like to first and foremost thank my supervisor Professor J. Christopher Beck. His invaluable advice and dedication to helping me do my best pushed me to accomplish more than I ever thought I could and made these past two years incredibly fulfilling. I firmly believe I could not have had a more thoughtful, patient, and inspirational mentor.

I would also like to thank my committee member Professor Andre Cire for taking the time to read my thesis and provide feedback.

Thank you to our industrial partner, Visual8, for providing the data used in this thesis.

To the members of TIDEL, I will always be grateful for your support. Especially Chiara, Eldan, Margarita, Kyle and Arik, thanks for helping me when I knew absolutely nothing and giving me the opportunity to laugh and have fun during hard times.

I want to thank my parents, who have always been my biggest supporters and go out of their way to make me feel loved and safe. I will be eternally grateful for their love and sacrifices.

Finally, to my partner-in-crime and favourite person in the world, Andrew Robinson McMillen IV, thank you for everything you do.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Composite materials are created when two or more constituent materials are combined to form a new material with different characteristics than any of its individual components. One of the significant innovations in aerospace manufacturing came in the 1940's when fibreglass, a composite material made of polymeric materials and glass fibres, was first used in the Boeing 707 passenger jet [103]. From the 1970's, carbon fibre reinforced polymer (CFRP), a composite material made of polymeric materials and carbon fibres, started to replace fibreglass as the main composite material used in airplane manufacturing [103]. Today, the Airbus A350 XWB is comprised of 52% CFRP [90]. The growing need for CFRP has placed more pressure on the manufacturing processes that create airplane composite parts such as fuselages and wings. Due to the complexity of these processes, the aerospace industry often faces massive delays which can cost billions of dollars [110]. Companies have been trying to improve process efficiency for years [46] but many manufacturing facilities are still struggling to effectively schedule activities in the composites manufacturing process. Thus, in conjunction with our industrial partner, we have identified an important industrial problem that, to the best of our knowledge, does not have a well-tested, state-of-the-art solution approach.

This thesis aims to define and characterize a novel optimization problem arising from the production of composite materials and develop scalable solution approaches. To accomplish this goal, we first define the *Composites Manufacturing Problem* (CMP), which consists of four stages in the manufacturing process: tool preparation, layup, autoclave curing, and demould. Then, we define an abstraction of the CMP, the *Two-Stage Bin Packing and Hybrid Flowshop Scheduling Problem* (2BPHFSP) to help us better understand the inherent problem structure. We present an extensive set of solution approaches for each problem, utilizing techniques from Mixed Integer Programming (MIP), Constraint Programming (CP), Logic-based Benders Decomposition (LBBD), heuristics, metaheuristics, and constrained clustering. Empirical analyses are performed for both problems to understand the benefits and drawbacks of each approach.

Results from the 2BPHFSP imply that strictly exact methods such as monolithic MIP and CP models or LBBD models are most likely unable to produce good schedules for real-life instances due to their size and complexity. However, a hybrid approach comprised of a heuristic and a CP model performed very well and showed high scalability. Results from the CMP further support this observation, where

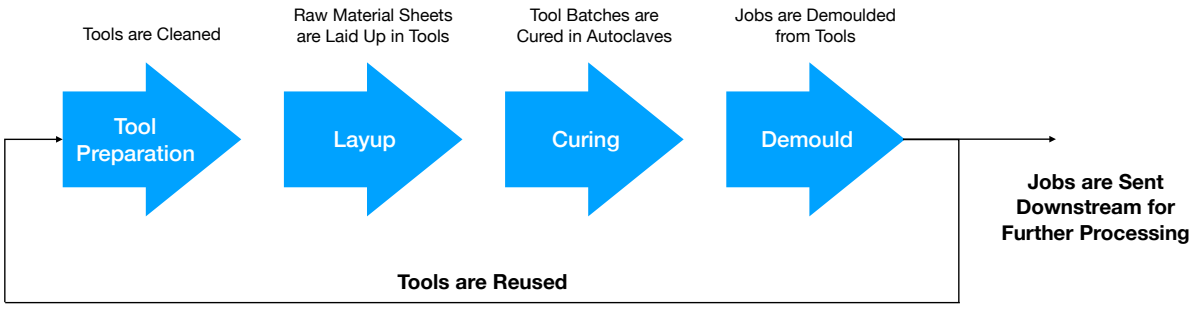


Figure 1.1: Summary of the CMP process.

a similar hybrid approach using a heuristic and a CP model consistently performed better than all of the other models. Other non-exact approaches, such as genetic algorithms and constrained clustering, performed slightly worse than the heuristic-CP hybrid, but still appear to be much more robust than the exact approaches.

1.1 Background and Problem Characteristics

A composite material is made up of two or more constituent materials that remain distinct and separate in the final material. The process of making the composite material used in aerospace manufacturing, CFRP, consists of combining a binding polymer with carbon fibre and several optional additives [114]. The most common binding polymer is epoxy, which is a type of reactive polymer that starts as a liquid but hardens once it is cured using heat or a catalytic reaction. If carbon fibres are added to the epoxy before curing, the resulting material, CFRP, has an extremely high strength-to-weight ratio and stiffness (rigidity), two highly desirable properties in an airplane [114]. Carbon fibre can reduce the weight of an airplane by up to 20%, saving an estimated $1 million dollars per kilogram over the lifetime of the plane [22]. Some other materials that can be added along with carbon fibres include aramids, glass fibres, silica, and carbon nanotubes [114].

The manufacturing method most airplane manufacturers use to make CFRP parts is moulding, where sheets of carbon fibre cloth are layered with other optional materials and liquid epoxy into a mould in the shape of the final product(s) [114]. The mould is then air-cured, or, more often, heat-cured in an autoclave. Parts must be taken out of a mould after curing in a stage called demould, and then the mould is cleaned in a process called tool preparation before it can be reused again. Thus, we have a four-stage process surrounding the curing of a composite part in an autoclave: moulds (hereby referred to as tools) are first cleaned in *Tool Preparation*, then layers of raw materials and epoxy are placed in the mould in a process called *Layup*, the laid up tools go into an autoclave for *Curing*, and finally the parts are removed from their tools in *Demould*. This four-stage manufacturing process is the core of the scheduling problem in the CMP and is summarized in Figure 1.1.

One of the challenges of composites manufacturing is that the moulds are scarce resources due to their high cost. Another complicating factor in filling moulds with epoxy and raw materials is the existence of caps or covers that secure the part in its tool. We refer to these caps as *Top Tools* and the

tool that contains parts as the *Bottom Tool*. Top tools and bottom tools work together to hold a part in place during layup and curing, and only certain top tools can be used with certain bottom tools. The combination of top and bottom tools containing one or more parts can be considered a tool batch. Multiple tool batches can be processed simultaneously in an autoclave, given that there is enough space in the autoclave. A group of tool batches can then be considered an autoclave batch. These batches therefore form a hierarchical structure, where parts are first batched into tool batches and then tool batches into autoclave batches. This two-stage hierarchical batching process is the core of the batching problem in the CMP.

In summary, the CMP is characterized by two integrated processes: batching parts and tools at the same time as they are being processed in sequential stages. The theoretical separability of these two processes, contrasted with their inherent connectedness when making decisions, is a major theme throughout this thesis.

1.2 Thesis Outline

Chapter 2 formally defines and provides notation for the Composites Manufacturing Problem (CMP). The chapter closes with an analysis of real-life data provided by our industrial partner and an explanation of how instances used in our numerical experiments are generated.

Next, Chapter 3 first provides a literature review on problems related to the CMP, spanning the fields of batch scheduling, bin packing, resource-constrained project scheduling, and constrained clustering. Then we introduce the fundamentals of several solution techniques: MIP, CP, decomposition models, and heuristics.

Chapter 4 introduces and formally defines the Two-Stage Bin Packing and Hybrid Flowshop Scheduling Problem (2BPHFSP), an abstraction of the CMP that removes several problem complexities. The purpose of this chapter is to explore the inherent structure of the CMP and to understand how different decisions influence each other. Five approaches are developed and described: Mixed Integer Programming (MIP), Constraint Programming (CP), an Earliest-Due Date (EDD) heuristic, and two Logic-based Benders Decomposition (LBBD) models. Numerical results from testing the five approaches on randomly generated problem instances are discussed and analyzed. Lastly, two hybrid approaches are created based on the best performing approaches and tested on the same sets of problem instances. The LBBD models performed the best overall but do not appear to be very scalable. However, a hybrid approach using the EDD heuristic combined with CP performs comparably to LBBD and exhibits high scalability.

Chapter 5 takes the solution approaches tested in Chapter 4 and the information we learned about the problem itself to create similar models and algorithms for the CMP. We focus on scaling up the complexity in this chapter and perform experiments on smaller-than-production size instances. Four approaches are created: MIP packing with CP scheduling, CP packing with CP scheduling, an LBBD model, and an EDD heuristic algorithm for packing with CP scheduling. As in Chapter 4, we show numerical results obtained from testing the approaches on randomly generated problem instances and

analyze each approach's performance. The EDD heuristic with CP outperformed all other models by a significant margin, implying that we should not attempt to scale the other models to solve larger instances.

Chapter 6 takes the best approach from Chapter 5, the EDD packing heuristic with CP scheduling, creates and describes five new approaches, and tests these six approaches on production-scale instances. The five new approaches are: EDD packing with a parallel scheduling heuristic, EDD packing with a genetic algorithm, constrained clustering packing with CP scheduling, constrained clustering packing with a parallel scheduling heuristic, and constrained clustering with a genetic algorithm. The numerical results in this chapter show how different techniques might perform if implemented in an actual plant. Similarly to Chapter 5, the EDD heuristic with CP outperforms all other models. Finally, we analyze and discuss the inherent bottlenecks in the system and how they may impact the lower bounds of our approaches. We found two notable bottlenecks, a small bottleneck formed by the stage-specific resources in the layup stage, and a large bottleneck formed by a few tools that are fully utilized across almost the entire schedule horizon.

The thesis concludes with Chapter 7 which summarizes the results and conclusions made throughout the chapters and discusses directions for future research.

1.3 Contributions

The following list describes the main contributions of this thesis.

- We introduced two novel optimization problems motivated by real-world composites manufacturing: the Two-Stage Bin Packing and Hybrid Flowshop Scheduling Problem (2BPHFSP) and the Composites Manufacturing Problem (CMP).
- We developed seven solution approaches to the 2BPHFSP and nine solution approaches to the CMP using techniques from Mixed Integer Programming (MIP), Constraint Programming (CP), Logic-based Benders Decomposition (LBBD), heuristic algorithms, genetic algorithms, and constrained clustering algorithms.
- We performed empirical analyses on both problems and found that the overall best-performing approach to be a hybrid approach of a heuristic algorithm and CP.
- We discovered the existence of bottleneck resources in the CMP which imply that our best solutions to the CMP are reasonably close to optimal.

This thesis also provides general research value in several important optimization fields. First, the decomposition models developed for this thesis can be used as a blueprint for any optimization problem where the difficulty lies in connecting a batching problem with a scheduling problem. For example, such a situation may arise in batch chemical manufacturing where the orders fulfilled by each batch of chemicals need to be determined in conjunction with when these batches are scheduled to be created [20]. Second, the approach taken by the MIP model developed for the *Pattern Bin Packing* problem, introduced and explained in Chapter 4, can be applied to set partitioning problems. And lastly, we showed the mapping between size-constrained clustering and the one-stage bin packing problem in Chapter 6.

Chapter 2

Problem Preliminaries

In this chapter, we formally define our problem of interest, the Composites Manufacturing Problem (CMP). Furthermore, we will discuss some key complexities of the CMP that cause this problem to be more complex and layered than most existing problems of a similar nature in literature. This chapter closes with an overview of the real-world data provided by our industrial collaborator.

2.1 Problem Definition

Let us first define the inputs to the CMP. We are given a set of jobs $\mathcal{J}$ that need to be processed sequentially in four stages: tool preparation, layup, curing, and demould. Tool preparation and demould are auxiliary activities that take up a negligible amount of time compared to layup and curing, which dominate the process. Each job has a due date d_j and a processing time for each stage pt_j^{stage} where $stage \in \{prep, layup, cure, demould\}$. A job produces one distinct vehicular part at the end of the demould stage that is sent to downstream processes for finishing and assembly. This high level process is shown in Figure 2.1.

We have a set of non-identical, renewable resources called tools that are divided into bottom tools $\mathcal{N}$ and top tools $\mathcal{T}$. Each job needs to be processed in all four stages accompanied by one bottom tool

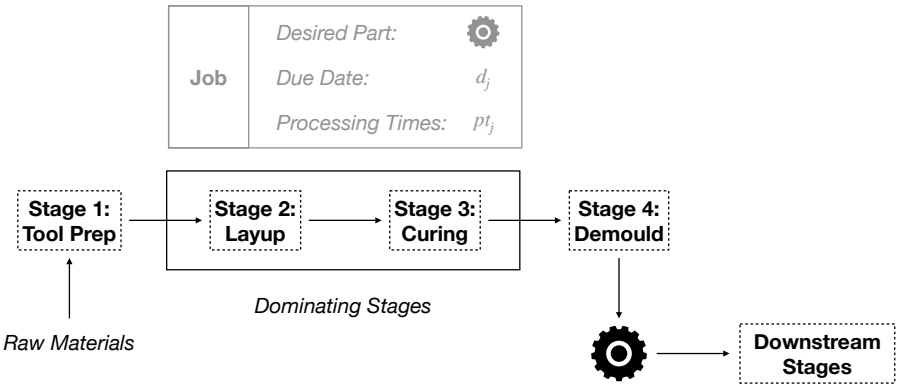


Figure 2.1: Job routing through the CMP.

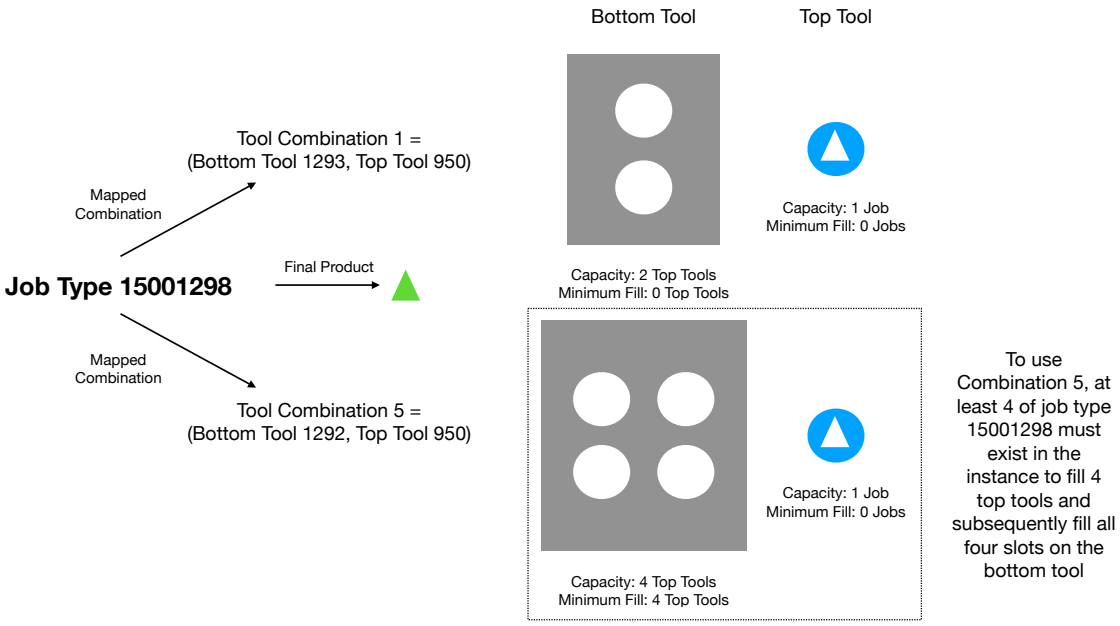


Figure 2.2: Connections between tool combinations, jobs, and tools.

and one top tool. Let us denote a pair of one bottom tool b and one top tool o as a tool combination $c = (b, o)$. Each job is mapped to one or two tool combinations; a job j can only be processed by one of its mapped combinations $c \in \mathcal{C}^j$. Each top tool contains one or more slots to hold assigned jobs, and each bottom tool contains one or more slots to hold top tools. If either a top or bottom tool has multiple slots, the tool may also have a minimum capacity requirement which enforces that the tool can only be used if all of its slots are used. For example, if a bottom tool b has three slots to hold top tool o and a minimum capacity requirement of three, we can only use b if we have enough jobs to fill three o tools and thus fill all three slots on b. Figure 2.2 shows an example that illustrates the connections between tool combinations, jobs, and tools.

Let us denote each filled bottom tool as a tool batch. Tool batches are formed in the tool preparation and layup stages. During tool preparation, bottom and top tool slots are cleaned and prepared to process assigned jobs. The total processing time for a tool batch in tool preparation is the sum of tool preparation processing times of jobs in the batch, as slots are cleaned and prepared sequentially. Processing times are sequence independent, so the order of preparation does not matter. Next, in the layup stage, sheets of raw materials such as carbon fibre or fibreglass are layered in the mould slots along with an epoxy resin. Similar to tool preparation, slots are laid up sequentially with no particular order, so the total layup time for a tool batch is the sum of layup processing times of jobs in the batch. After layup, we have a complete tool batch consisting of one bottom tool, at least one top tool, and at least one job.

Next, complete tool batches are cured in autoclaves (industrial ovens) where the laid-up materials and epoxy resin are bound to form a new composite material. Autoclaves are very large, so multiple tool batches can be cured at the same time. In addition, the extreme temperature and pressure conditions

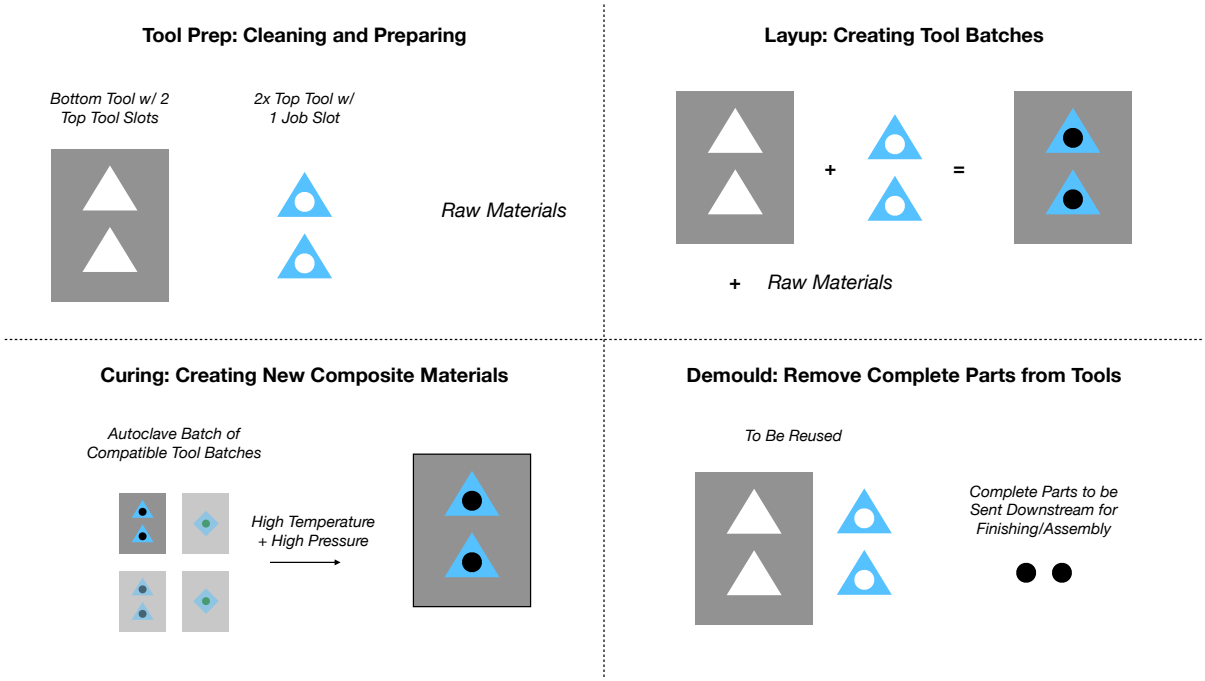


Figure 2.3: Detailed look at stages of the CMP.

within autoclaves during the curing process require large amounts of energy to maintain, so it is desirable to cure as many tool batches at the same time as possible to reduce the number of autoclave operational hours. Another constraint is that each tool batch needs to be cured using a certain autoclave cycle type that is predetermined based on the selected tool combination. Cycle type determines the temperature, pressure, and time needed to completely cure tool batches. Each cycle type can only be run on a specific autoclave, so only tool batches with the same cycle type can be cured together. We know the size of each tool batch, determined by its bottom tool. The sum of the sizes of the tool batches being cured simultaneously cannot exceed the autoclave's capacity. A group of tool batches being processed together in an autoclave is therefore denoted as an autoclave batch.

The last stage is demould, where completed jobs are removed from their tool slots and sent to downstream processes. This stage is very similar to tool preparation and layup, jobs are removed sequentially with no particular order and the total demould time is the sum of demould processing times of jobs in the tool batch. After jobs are removed, the bottom and top tools of a tool batch can be cleaned and reused to make new tool batches. The completion time of a job is after it has been removed from its tool batch in demould, and can be compared to its due date to obtain job tardiness. Figure 2.3 shows a more detailed view of the CMP stages.

Throughout all four stages, we need to use machine resources. Tool preparation, layup, and demould machines are specific sections of the shop floor, and curing machines are autoclaves. Each tool batch takes up one floor section in tool preparation, layup, and demould, and each autoclave batch takes up one autoclave in curing. Tool batches also require labour resources during tool preparation, layup and demould. Machine and labour resources are predetermined for each tool combination, therefore, we can

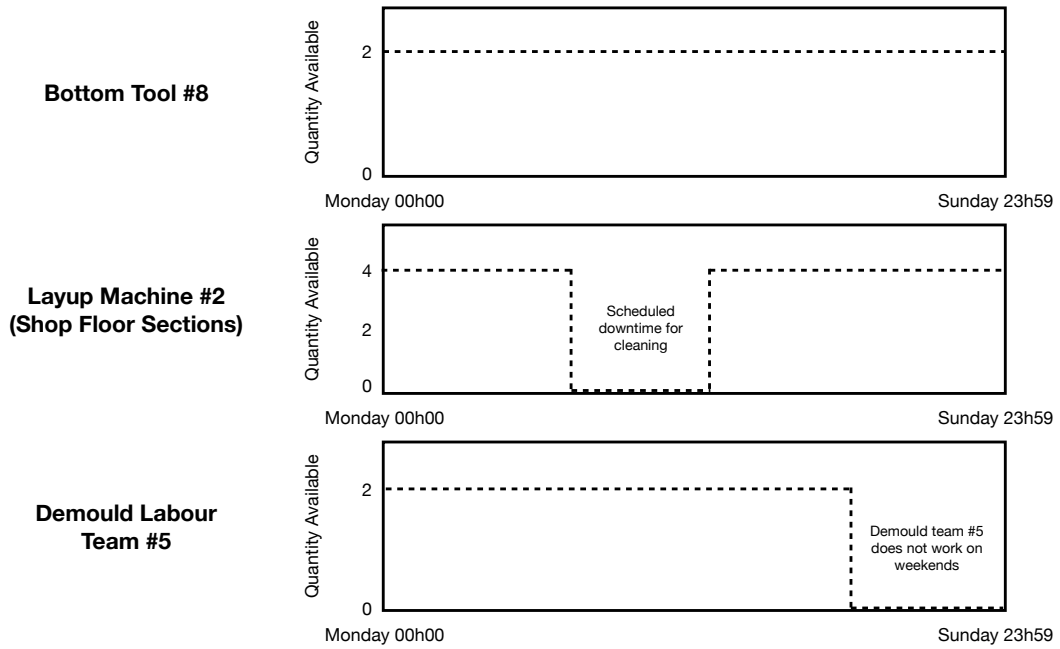


Figure 2.4: Sample resource capacities over time.

easily ascertain the resources required by each tool batch. We are also given the schedules of machines and labour teams, which are renewable resources with time-varying capacities. For example, one autoclave machine may not be operational during weekends, or we may have five units of labour type 25 from Monday to Thursday but only two units from Friday to Sunday. As aforementioned, tools are also renewable resources, but have constant capacities. Therefore, the CMP utilizes three distinct sets of resources: tools, machines, and labour. Figure 2.4 shows an example of resource capacity over time in the CMP.

The CMP is a highly specialized and complex variation of traditional batch scheduling. Tool batch processing times in tool preparation, layup, and demould are sums of processing times of its jobs. Thus, we can consider these stages to be batching steps that fall under the family batch scheduling model (see Section 3.2.1). After layup, each tool batch must then be cured in an autoclave simultaneously with other tool batches. We can consider this step to be another batching step that falls under the batching machines model (see Section 3.2.1).

We can also consider the CMP as the union of two distinct problems, multi-stage bin packing (see Section 3.2.2), and hybrid flow-shop scheduling [83]. Multi-stage bin packing encompasses the batching steps, i.e. jobs into top tools, top tools into bottom tools to form tool batches, and tool batches into autoclave batches, constrained by tool slot availabilities and autoclave capacities. A solution to the multi-stage bin packing problem then forms the parameters of the hybrid flow-shop scheduling problem. Each tool batch must be scheduled in tool preparation, layup, and demould, each autoclave batch must be scheduled in curing, and precedence exists between stages.

Formal Definition. Now, let us formally define the problem. We are given a set of jobs $\mathcal{J}$, each job j in $\mathcal{J}$ is associated with the parameter set $\{d_j, pt_j^{prep}, pt_j^{layup}, pt_j^{cure}, pt_j^{demould}, \mathcal{C}^j\}$: d_j is the due date, pt_j^{stage} is the processing time in each stage, and $\mathcal{C}^j$ is the set of allowed tool combinations. Table 2.1 contains a summary of the general notation introduced in this chapter.

A tool combination is formed of one top tool from the set of top tools $\mathcal{T}$ and one bottom tool from the set of bottom tools $\mathcal{N}$. Each top tool $o \in \mathcal{T}$ is associated with the parameter set $\{cap_o, min_o, q_o\}$: cap_o is the capacity (i.e. the maximum number of jobs that can be slotted into this top tool), min_o is the minimum capacity requirement (i.e. the minimum number of jobs that need to be slotted into this top tool if used), and q_o is the available quantity. Each bottom tool $b \in \mathcal{N}$ is associated with the parameter set $\{cap_b, min_b, q_b, s_b\}$: cap_b is the capacity (i.e. the maximum number of top tools that can be slotted into this bottom tool), min_b is the minimum capacity requirement (i.e. the minimum number of top tools that need to be slotted into this bottom tool if is used), q_b is the available quantity, and s_b is the size.[1] Each tool combination $c \in \mathcal{C}$ is associated with the parameter set $\{m_c^{prep}, m_c^{layup}, m_c^{cure}, m_c^{demould}, l_c^{prep}, l_c^{layup}, l_c^{demould}, lq_c^{prep}, lq_c^{layup}, lq_c^{demould}, ac_c\}$: m_c^{stage} is the required machine at each stage, l_c^{stage} is the required labour team type at the appropriate stages, lq_c^{stage} is the required quantity of labour teams at the appropriate stages, and ac_c is the autoclave cycle type.

There are two more types of resources: machines and labour teams. q_n^m and q_n^l represent the available quantity of machine m and labour team l at time period n. We can split the entire set of machines $\mathcal{M}$ into the set of tool preparation machines $\mathcal{M}^{prep}$, the set of layup machines $\mathcal{M}^{layup}$, the set of autoclaves $\mathcal{M}^{cure}$, and the set of demould machines $\mathcal{M}^{demould}$. We can also split the entire set of labour teams $\mathcal{L}$ into the set of tool preparation labour teams $\mathcal{L}^{prep}$, the set of layup labour teams $\mathcal{L}^{layup}$, and the set of demould labour teams $\mathcal{L}^{demould}$.

The result of batching is a set of tool batches $\mathcal{B}^1$ and a set of autoclave batches $\mathcal{B}^2$. Each tool batch $k \in \mathcal{B}^1$ contains a set of jobs and each autoclave batch $i \in \mathcal{B}^2$ contains a set of tool batches.

2.1.1 Key Complexities

We can summarize the CMP with four key complexities:

1. The multi-stage bin packing problem consists of packing jobs into tool batches defined by tool combinations, then packing tool batches into autoclave batches. This problem is further complicated by the presence of autoclave cycle types, where tool batches can only be processed in an autoclave batch with other tool batches that have the same cycle type. Thus, we need to solve a separate multi-stage bin packing problem for each subset of tool batches with the same cycle type.

2. A job can only be assigned to one of its mapped tool combinations. Some tool combinations are formed using tools with a non-zero minimum fill requirement. Thus, depending on the instance, we may not be able to use some tool combinations because there are not enough jobs to fill such a tool batch.

3. A subset of machines in the tool preparation, layup, and demould stages contain multiple cells,

[1] Top tools are slotted into bottom tools so we do not have to represent the size of top tools.

Table 2.1: General notation.

Parameter	Description
$j \in \mathcal{J}$	Set of jobs
$k \in \mathcal{B}^1$	Set of tool batches
$i \in \mathcal{B}^2$	Set of autoclave batches
$c \in \mathcal{C}$	Set of tool combinations
$c \in \mathcal{C}^j$	Set of tool combinations mapped to job j
$o \in \mathcal{T}$	Set of top tools
$b \in \mathcal{N}$	Set of bottom tools
$m \in \mathcal{M}$	Set of machines
$l \in \mathcal{L}$	Set of labour teams
$n \in \mathcal{H}$	Set of time periods
d_j	Due date of job j
pt_j^{prep}	Processing time of job j in tool preparation
pt_j^{layup}	Processing time of job j in layup
pt_j^{cure}	Processing time of job j in curing
$pt_j^{demould}$	Processing time of job j in demould
cap_o	Capacity of top tool o
min_o	Minimum fill requirement of top tool o
q_o	Quantity of top tool o
cap_b	Capacity of bottom tool b
min_b	Minimum fill requirement of bottom tool b
q_b	Quantity of bottom tool b
s_b	Size of bottom tool b
m_c^{prep}	Required tool preparation machine for tool combination c
m_c^{layup}	Required layup machine for tool combination c
m_c^{cure}	Required curing machine for tool combination c
$m_c^{demould}$	Required demould machine for tool combination c
l_c^{prep}	Required tool preparation labour team for tool combination c
l_c^{layup}	Required layup labour team for tool combination c
$l_c^{demould}$	Required demould labour team for tool combination c
lq_c^{prep}	Required tool preparation labour team units for tool combination c
lq_c^{layup}	Required layup labour team units for tool combination c
$lq_c^{demould}$	Required demould labour team units for tool combination c
ac_c	Autoclave cycle type of tool combination c
q_n^m	Available quantity of machine m during time period n
q_n^l	Available quantity of labour team l during time period n

i.e. multiple tool batches can be processed simultaneously. Thus, we can describe the scheduling aspect of the CMP as a multi-stage hybrid flowshop scheduling problem.

4. There are three types of resources: tools, machines, and labour teams. Bottom tools exist in limited quantities and can be represented as constant-capacity resources. We can assume top tools exist as unlimited resources; top and bottom tools are always used in tandem and bottom tools are responsible for the tool availability bottleneck. Machines and labour teams have changing quantities available over time and can be represented as resources with time-varying capacities. If a tool is used by a tool batch, the tool is used from the beginning of the tool preparation stage to the end of the demould stage for that tool batch. Machines and labour teams are used for the duration of their associated stage.

2.2 Data Overview

This section looks at the parameters that characterize the operational data provided by our industrial partner. We can separate these parameters into five categories that completely define the environment that a job is processed within.

1. A list of all possible jobs that may be present in an order list. There are a total of 395 distinct jobs and each job produces a distinct vehicular part. For each job, we know the set of mapped tool combinations and its size. The estimated size of a job is an estimation of how much space a job is likely to take up within an autoclave batch after it has been packed into a tool batch. Table 2.2 shows a sample list.

2. A list of all top and bottom tools along with their sizes, capacities, available quantities, and minimum capacity requirements. There is a total of 429 bottom tools and 793 top tools. The capacity of a top tool is the number of slots available to hold jobs, whereas the capacity of a bottom tool is the number of slots available to hold top tools. Therefore, if a top tool has capacity 2 and a bottom tool has capacity 3, the bottom tool can hold a maximum of 6 jobs. However, the majority of tools have a capacity of 1. Due to the configuration of tool batches where top tools are fixed on top of bottom tools, only bottom tools possess a size parameter. Therefore, a tool batch inherits its size from its bottom tool. This size will be used to determine how many tool batches can fit within an autoclave. Figure 2.5 provides an overview of tool parameters. We can see that most tools, regardless of being a bottom or top tool, exist as a single tool with a capacity of 1. This implies that most tool batches will be a single job batched with one top tool and one bottom tool.

3. Tool combinations have some of the most important parameters, giving us the exact machines and labour teams, along with the number of labour team units, required for processing in each stage. Table 2.3 shows a sample list.

4. The machine parameters describe all available machines at each stage and their capacities. There are 9 tool preparation machines, 16 layup machines, 12 autoclave machines, and 10 demould machines. Each autoclave can process one autoclave batch at a time and has a limit on the autoclave batch size. This limit cannot be exceeded by the sum of tool batch sizes for any autoclave batch. On the other hand, non-autoclave machines do not have size limitations, one tool batch

Table 2.2: Sample list of job parameters.

Job Type	Mapped Tool Combinations	Combination Tools	Size
14000365	Combination 1	Bottom Tool 1200, Top Tool 950	1000
14001006	Combination 10	Bottom Tool 1141, Top Tool 1001	400
14001006	Combination 11	Bottom Tool 1140, Top Tool 800	400
...	...	...	...

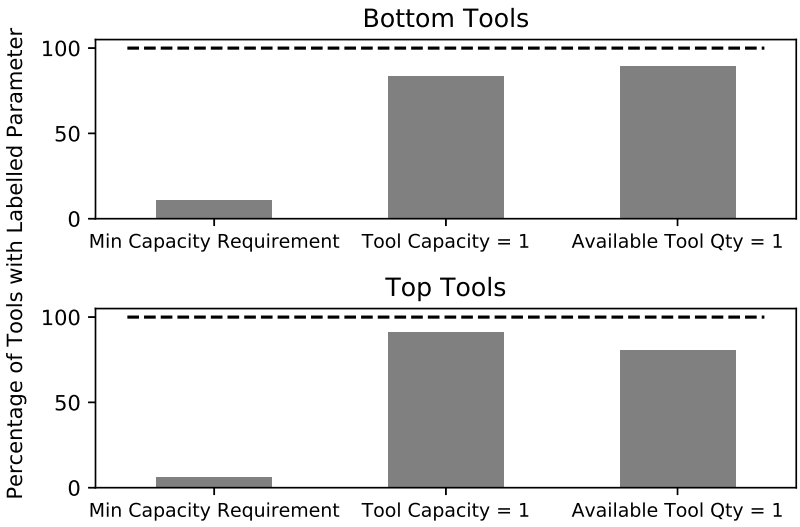


Figure 2.5: Overview of tool parameters, analysis showing which parameter values are the most common. Tools with a capacity equal to 1 are not included in the percentage of tools with a minimum capacity requirement.

is treated as one unit regardless of its size. Some non-autoclave machines are also capable of processing multiple tool batches at once. As described in the previous section, any non-autoclave machine is merely a section on the shop floor. Figure 2.6 shows how such a machine exists in real life.

5. We have the weekly availabilities of machines and labour teams. A week is split into 49 shifts and the parameters tell us how many units of the machine or labour team are available during every shift. Table 2.4 shows a sample list.

2.2.1 Instance Generation

Our industrial partner also provided a sample order of 4565 jobs. On average, an order contains 4000 jobs, so we cannot create sample instances to test our models and algorithms by splitting the sample order directly. Thus, to produce instances with a similar distribution as the real-world dataset, we create bootstrap instances by sampling with replacement from the real-world dataset. Each job in an instance is also assigned a due date in number of weeks, sampled from a uniform distribution from 1 to 4. Table 2.5 shows a sample instance of 10 jobs.

Table 2.3: Sample list of tool combination parameters.

Combination	Prep			...	Demould		
	Machine	Labour	Qty	...	Machine	Labour	Qty
Combination 1 = (Bottom Tool 1200, Top Tool 950)	Tool Prep 869	Team 1159	2	...	Demould 878	Team 1301	3
Combination 2 = (Bottom Tool 1840, Top Tool 1002)	Tool Prep 869	Team 1169	1	...	Demould 878	Team 1159	1
Combination 3 = (Bottom Tool 1423, Top Tool 901)	Tool Prep 980	Team 2164	2	...	Demould 878	Team 1329	1
...	...	...	...	...	...	...	...

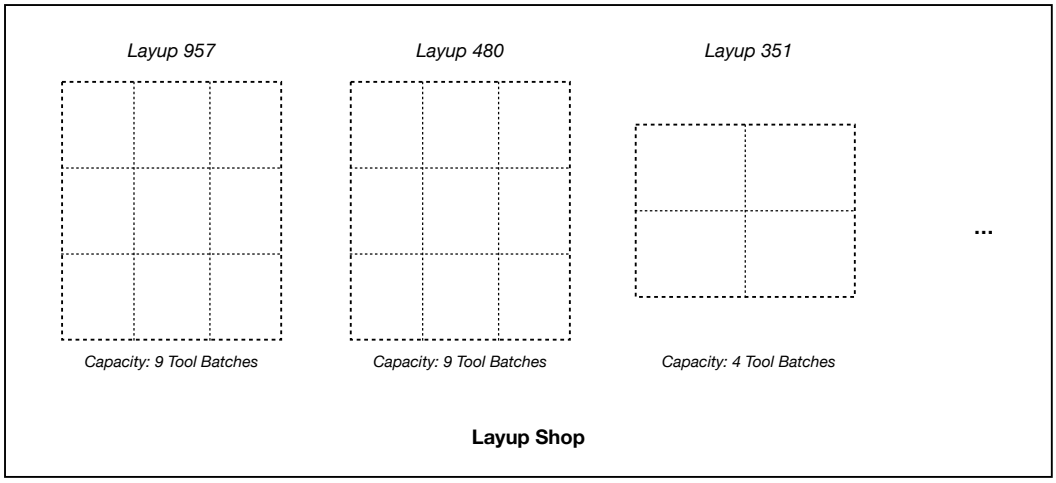


Figure 2.6: Layout of non-autoclave machines within a layup shop.

Table 2.4: Sample list of machine/labour team weekly availabilities

Machine/Team	Day 1 - 0:00 to 4:59	Day 1 - 5:00 to 6:29	...	Day 7 - 17:30 to 22:29	Day 7 - 22:30 to 23:59
Team 1159	1	1	...	3	3
...	...	...	...	...	...
Tool Prep 859	0	0	...	10	10
...	...	...	...	...	...

Table 2.5: Sample instance of 10 randomly generated jobs.

Job Type	Due Date
15003215	Week 1
15003209	Week 3
15001667	Week 2
15002220	Week 3
14004521	Week 4
14003764	Week 3
15003215	Week 1
15001667	Week 4
15003209	Week 1
15003209	Week 2

2.3 Summary

In this chapter, we introduced and formally defined the Composites Manufacturing Problem (CMP). The CMP is a complex problem that spans multiple stages and uses multiple resources, with problem instances that contain 4000 jobs on average. We summarized the CMP using four key complexities: the multi-stage bin packing nature of the batching steps, the minimum fill requirements of some tools, the multi-capacity nature of resources, and the time-varying capacities of some resources.

The following chapter presents a literature review of existing problems, models, and algorithms that can be related to the CMP as well as some solution techniques that are commonly used to solve hard combinatorial problems like the CMP. However, due to the complexity of the CMP, we are most likely unable to apply out-of-the-box techniques to solve the problem as is. Thus, we will first focus on understanding the underlying problem structures of the CMP better by solving an abstraction in Chapter 4. This abstraction will contain some of the key complexities described in this chapter, namely the multi-stage bin packing and multi-capacity resources. Results from modelling and solving the abstracted problem can help guide us in developing solutions for the full problem. Then, Chapter 5 will focus on solving the CMP with its full complexity on smaller problem instances, and finally, Chapter 6 will turn to solving the CMP with its full complexity on full-scale problem instances.

Chapter 3

Literature Review

This chapter first introduces relevant works in four areas related to the composites manufacturing problem (CMP): batch scheduling, bin packing, project scheduling, and constrained clustering. Then we present the foundational ideas and theories behind four important techniques used to solve discrete optimization problems: Mixed Integer Programming (MIP), Constraint Programming (CP), decomposition methods, and heuristics.

3.1 Composites Manufacturing

A few authors have tackled problems related to composites manufacturing in some form over the past two decades. To the best of our knowledge, composites manufacturing first appeared in literature when Hindle and Duffin [59] presented a simulation-based system to schedule the composites manufacturing process. However, they do not define the CMP nor do they provide the actual simulation model; the paper's main focus is on explaining how the system can aid facilities to improve production efficiency.

Collart [33] appears to have been the first to formally define a scheduling problem related to composites manufacturing. However, their definition makes two assumptions about tools that drastically reduce the problem complexity, namely that each job can only be processed using one type of tool and that a tool can only process one job at a time. Collart presents a monolithic mixed integer programming (MIP) model along with a decomposition model that uses MIP to pack batches and simulated annealing to schedule batches. The MIP and decomposition models were both able to solve instances with up to 260 autoclave batches over four weeks. Azami et al. [6] presents another MIP model and a genetic algorithm to solve the same problem as Collart [33]. The MIP model was tested on instances with up to 100 jobs and the genetic algorithm was tested on instances with up to 300 jobs. However, the assumptions in the problem definition render these models inadequate for our problem.

Most recently, Hueber et al. [64] developed a custom branch-and-bound algorithm, named APOLLO (Autoclave Production Order, Layup and Logistics Optimization), for scheduling the layup and curing stages of a composites manufacturing process. They showed that APOLLO was able to schedule one week's worth of orders, however, the calculation times are extremely long, e.g. five days of computation time to schedule four batches to optimality.

We do not compare the approaches developed in this thesis against existing models in literature as their assumptions do not fit our problem. Thus, we can conclude that the CMP as defined in this thesis is a novel optimization problem.

Due to the lack of proper models that solve the CMP in its entirely, we look at similar but more well-researched classes of problems. Next, we present a discussion of related problems that could be projected onto the CMP along with their associated solution techniques; we review exact models as well as heuristic/metaheuristic algorithms for these problems.

3.2 Related Problems

Batch scheduling encompasses a large family of models that share the characteristic that jobs must be processed together in some manner [101]. All four stages of the CMP can be considered batch scheduling, as jobs are processed sequentially in tool preparation, layup, and demould and jobs are processed simultaneously in curing. Alternatively, the batching aspect in batch scheduling can be transformed into an equivalent bin packing problem [78].

There are also other bodies of literature with connections to the CMP that are less obvious. Project scheduling [25] is a tremendously popular area of research, and we can map the time-varying availability of resources in the CMP (machines and labour teams) to the time-varying resource capacities found in project scheduling. Another area of research that can be connected to the CMP is constrained clustering [11]. If we think of an autoclave batch as a cluster of tool batches, the batching problem of tool batches to autoclave batches can be mapped to a clustering problem with constraints on cluster size and the cardinality of tools within clusters.

We present notable works from each of these four bodies of literature along with their common solution techniques.

3.2.1 Batch Scheduling

Batch scheduling has been separated into two main classes of problems: the *Family Batching Model* and the *Batching Machines Model*, as categorized by Potts and Kovalyov [101]. In the family batching model, jobs that share certain characteristics belong to predefined families with no setup costs required to sequentially process jobs within the same family. Thus, a batch in this context is a set of jobs that are scheduled contiguously on a machine with a single setup. Under the batching machines model, jobs are processed simultaneously on the same machine. The most common application of the batching machines model is in semiconductor manufacturing, where the 'burn in' operation to cure circuit boards is performed simultaneously for many boards in the same oven [94].

In addition to the family batching model and the batching machines model categories, batch scheduling problems can be characterized as single-machine, parallel-machine or shop problems. Table 3.1 presents papers in literature that fall within each category.

Table 3.1: Categorization of batch scheduling problems.

	Single Machine	Parallel Machines	Shop
Family Batching Model	[95, 47, 35, 93, 36]	[7, 95]	[76, 122, 113]
Batching Machines Model	[26]	[26, 15]	[119, 80, 50, 117, 97, 54, 87, 74]

Family Batching Model. The single-machine variant of the family batching model with different objectives has been studied for many years. For example, Monma and Potts [95] showed that in an optimal schedule for the total weighted completion time objective, the shortest weighted processing time (SWPT) priority rule applies within each family. However, these basic problems are still NP-complete [101] as, for example, the families need to be sequenced. Multiple formulations of exact algorithms such as dynamic programming [47] and branch-and-bound [35] have been developed through the years for such problems. Heuristic approaches have also been studied, such as genetic algorithms [93] and variants of neighbourhood search heuristics [36]. Algorithms developed for the single machine case have been extended to accommodate other variants of the family batching model, such as parallel machines and multi-machine shop problems [101], and many MIP formulations have also been developed for variants of the family batching model. Balakrishnan et al. [7] presented a MIP model alongside a Benders decomposition model for the objective of minimizing weighted earliness/tardiness with parallel machines. Kurz and Askin [76] developed MIP and heuristic approaches for minimizing makespan in a flow shop.

The body of literature surrounding family batching is extremely broad and much of the work is summarized in three comprehensive survey papers by Allahverdi et al. [2], Potts and Kovalyov [101], and Allahverdi et al. [3], arguably the most influential papers in the field of batch scheduling.

Batching Machines Model. Many algorithms of the same nature as the ones previously described for the family batching model have also been developed for the batching machines model. For the single-machine case, Brucker et al. [26] showed that the optimal schedule is where batches are created when jobs are ordered by the shortest processing time (SPT) priority rule and adjacent ordered jobs are grouped to form batches. Much of the early work in batching machines was introduced by Brucker et al. [26], including dynamic programming formulations for the single-machine and parallel machines cases with different objective functions.

One of the first MIP models developed to solve a batching machines problem was presented by T'kindt et al. [119], where binary decision variables choose between permutations of job sequences in a two-machine flowshop with one batching machine. Building on that, Liao and Liao [80] present MIP models that solve a two-machine flowshop problem where both machines are batching machines. Many more variants on batching machine flowshop problem exist, including setup times or costs [50], transportation considerations between stages [117], parallel machines [15], and release dates [97].

Some recent work on batching machines has seen the application of Constraint Programming (CP) to these scheduling problems. Ham et al. [54] present a CP approach using existing global constraints

to solve a parallel batching machine problem where jobs have release times, non-identical sizes, and incompatible families. An alternate approach was taken by Malapert et al. [87] where they developed a novel global constraint called `sequenceEDD` that minimizes the maximal lateness for a single batching machine where jobs have non-identical sizes. Kosch and Beck [74] presented a MIP model for the same batching machines problem as Malapert et al. that outperformed `sequenceEDD`.

In general, exact algorithms or models have been developed for many of the variants that can arise from the categorized problems in Table 3.1, e.g. jobs with release dates, incompatible job families, multiple stages, etc. However, almost all of these variants are too abstract to be useful for our application. Thus, to the best of our knowledge, the CMP or problems of similar complexity have not been solved in batch scheduling literature.

3.2.2 Bin Packing

Bin packing is one of the most famous combinatorial optimization problems and solution techniques to its variants and applications have been studied for decades [68]. Many surveys have been published throughout the years [84, 28, 32], most recently by Delorme et al. [40]. The basic, one-stage bin packing problem (BPP) is defined as follows: we are given a set of $j \in \mathcal{J}$ items with non-identical sizes and an unlimited number of identical bins that each have a capacity c; the objective is to pack all $j \in \mathcal{J}$ items into bins such that the sum of sizes of items in each bin does not exceed the bin capacity and the number of bins used is minimized.

Next, we present a basic MIP model of the BPP. Let $y_i \in \{0,1\}$ be a set of binary variables where $y_i = 1$ if bin $i \in \mathcal{I}$ is used in the solution and 0 otherwise. Let $x_{i,j} \in \{0,1\}$ be a second set of binary variables where $x_{i,j} = 1$ if item j is packed into bin i and 0 otherwise. Lastly, we define w_j to be the size of an item j.

$$\min \quad \sum_{i \in \mathcal{I}} y_i \tag{3.1}$$

$$\text{s.t.} \quad \sum_{i \in \mathcal{I}} w_j x_{i,j} \leq c y_i \qquad \forall i \in \mathcal{I} \tag{3.2}$$

$$\sum_{i \in \mathcal{I}} x_{i,j} = 1 \qquad \forall j \in \mathcal{J} \tag{3.3}$$

$$y_i \in \{0,1\}\ \forall i \in \mathcal{I};\ x_{i,j}\ \in \{0,1\}\ \forall i \in \mathcal{I},\ \forall j \in \mathcal{J}$$

Constraint (3.2) enforces capacity constraints on each bin and constraint (3.3) makes sure each item is assigned to exactly one bin.

Variants on the BPP include multi-dimensional bin packing [84], bin packing with cardinality constraints instead of capacity constraints [75], bin covering [5], etc. Table 3.2 presents a selection of papers in literature that fall within popular bin packing categories.

Martello and Toth [92] presented the most powerful branch-and-bound algorithm for the BPP, called MTP. However, branch-and-price subsequently became the most popular exact algorithm for solving the

Table 3.2: Categorization of bin packing problems.

Problem Characteristic	Existing Literature
One-Dimensional	[92, 52, 106, 112, 24]
Cardinality-Constrained	[69, 75, 24]
Multi-Dimensional	[16, 84, 91, 44, 85]

BPP. The literature on branch-and-price for bin packing is large and a full summary is beyond our scope; some important papers include Belov and Scheithauer's branch-and-price algorithm for the BPP and the two-dimensional capacity-constrained bin packing problem [16], Desaulniers et al.'s generic framework for cutting planes in branch-and-price [41], and Ben Amor and Valerio de Carvalho's survey on column generation [17]. Krause et al. [75] analyzed several approximation algorithms for the one-dimensional cardinality-constrained bin packing problem and found them to all have an asymptotic worst-case performance ratio of 2. Kellerer and Pferschy [69] built on the work by Krause et al. and presented a new heuristic with an asymptotic worst-case bound of 3/2. Brandao and Pedroso [24] presented an arc flow formulation capable of being applied to the BPP and cardinality-constrained bin packing problems.

Two-dimensional bin packing is a well-studied variant. Here we are given a set of $j \in \mathcal{J}$ items that now have a width w_j and a height h_j. Our set of bins $i \in \mathcal{I}$ also has a width W and a height H. The objective is the same as the BPP, but we now have two dimensions of capacity constraints. Lodi et al. [84] presented a survey on exact and heuristic methods to solve the two-dimensional bin packing problem. Much of the work categorized by Lodi et al. are either exact algorithms, such as branch-and-bound, or approximation algorithms, such as constructive heuristics or metaheuristics. Some MIP models have also been developed; the first one was presented by Gilmore and Gomory [48] and uses a column generation approach based on enumerating all the possible packing patterns of items in a single bin. The other type of multi-stage bin packing, the three-dimensional bin packing problem, has also been solved using branch-and-bound [91], local search [44], and numerous other approaches.

A special case of the two-dimensional bin packing problem occurs when items need to be packed "by levels." A level is formed by dividing the bin into horizontal strips and items packed within each strip are only constrained by their widths. This special case is important to this thesis as it can be projected onto the hierarchical batching of the CMP. Lodi et al. [85] presented a MIP model with a polynomial number of variables and constraints for the two-dimensional level bin packing problem. Without loss of generality, they assumed that:

1. The leftmost item packed in each level is the tallest of all items in the level.

2. The bottommost level packed in each bin is the tallest of all levels in the bin.

3. Items are sorted by height in descending order.

A level is then defined as being initialized by the lowest-indexed, and thus tallest, item assigned to the level, and a bin is initialized by the lowest-indexed, and thus tallest, level assigned to the bin. There are four sets of decision variables: two sets associated with packing items into levels and two sets associated

with packing levels into bins.

$$y_i = \begin{cases} 1 & \text{if item } i \text{ initializes level } i \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \in \mathcal{I} \tag{3.4}$$

$$x_{i,j} = \begin{cases} 1 & \text{if item } j \text{ initializes level } i \\ 0 & \text{otherwise} \end{cases} \qquad j > i \quad \forall i \in \{0, ..., |\mathcal{I}| - 1\} \tag{3.5}$$

$$q_k = \begin{cases} 1 & \text{if level } k \text{ initializes bin } k \\ 0 & \text{otherwise} \end{cases} \qquad \forall k \in \mathcal{K} \tag{3.6}$$

$$z_{k,i} = \begin{cases} 1 & \text{if level } i \text{ is allocated to bin } k \\ 0 & \text{otherwise} \end{cases} \qquad i > k, \quad \forall k \in \{0, ..., |\mathcal{K}| - 1\} \tag{3.7}$$

The MIP model is defined as follows.

$$\min \quad \sum_{k \in \mathcal{K}} q_k \tag{3.8}$$

$$\text{s.t.} \quad \sum_{i=1}^{j-1} x_{i,j} + y_j = 1 \qquad \forall j \in \mathcal{J} \tag{3.9}$$

$$\sum_{j=i+1}^{|\mathcal{I}|} w_j x_{i,j} \leq (W - w_i) y_i \qquad \forall i \in \{0, ..., |\mathcal{I}| - 1\} \tag{3.10}$$

$$\sum_{k=1}^{i-1} z_{k,i} + q_i = y_i \qquad \forall i \in \mathcal{I} \tag{3.11}$$

$$\sum_{i=k+1}^{|\mathcal{K}|} h_i z_{k,i} \leq (H - h_k) q_k \qquad \forall k \in \{0, ..., |\mathcal{K}| - 1\} \tag{3.12}$$

$$y_i \in \{0, 1\} \ \forall i \in \mathcal{I};\ x_{i,j} \in \{0, 1\} \ \forall i \in \mathcal{I},\ \forall j \in \mathcal{J};\ q_k \in \{0, 1\} \ \forall k \in \mathcal{K}$$

$$z_{k,i} \in \{0, 1\} \ \forall k \in \mathcal{K},\ \forall i \in \mathcal{I}$$

Constraint (3.9) makes sure each item is packed once, and either initializes a level or gets packed into an already initialized level. Constraint (3.10) imposes the width constraint on each level. Constraint (3.11) makes sure each used level is packed once, and either initializes a bin or gets packed into an already initialized bin. Constraint (3.12) imposes the height constraint on each level.

Lodi et al. [85] prove a number of lower bounds and provide constraints for several variants on the two-dimensional level bin packing problems. Puchinger and Raidl [105] improved the MIP models developed by Lodi et al. and extended the model to fit the three-dimensional case. More recent work on higher dimensional bin packing problems have focused on approximation algorithms [29]; some approaches using machine learning have also been proposed in the last few years [89].

3.2.3 Resource-Constrained Project Scheduling

The resource-constrained project scheduling problem (RCPSP) consists of scheduling a set of activities $j \in \mathcal{J}$ that have precedence and resource utilization constraints [25]. We have a set of renewable resources $r \in \mathcal{R}$ and each resource has a maximum capacity K_r. Activity j has a set of preceding activities

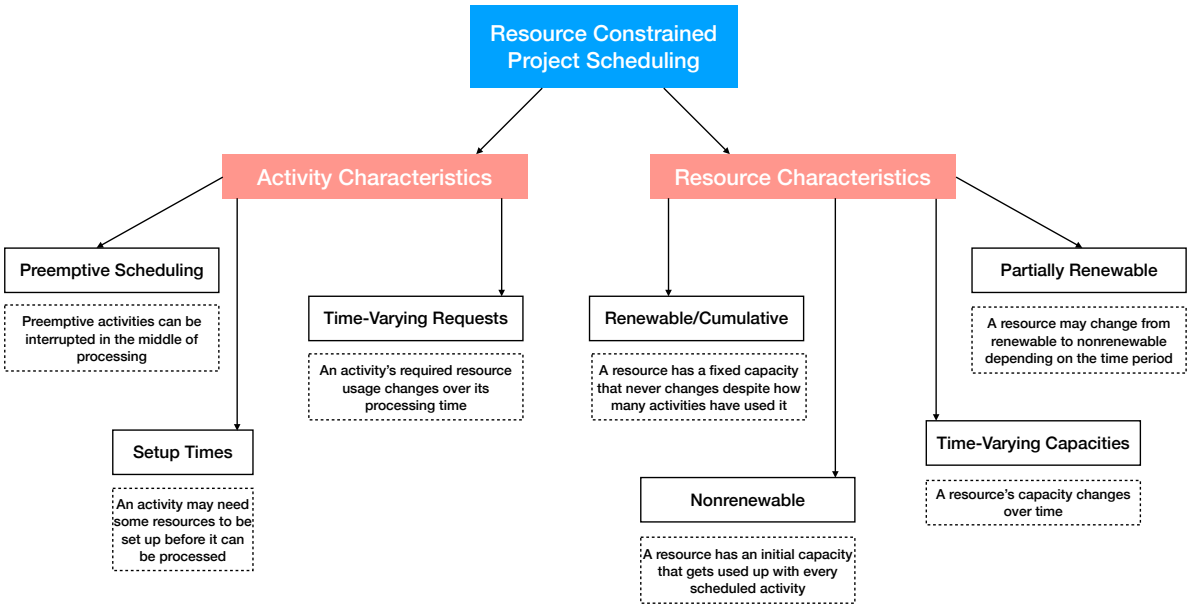


Figure 3.1: Characteristics of popular variants to the RCPSP.

Table 3.3: Solution techniques for the RCPSP.

Solution Technique	Existing Literature
Integer Programming	[104, 96]
Constraint Programming	[9, 57, 82]
Other Exact Approaches	[19]
Heuristics	[14, 70, 118, 21, 39]
Metaheuristics	[55, 4, 45, 42]

$\mathcal{P}_j$ and takes up $k_{j,r}$ units of resource r while being processed.

Several survey papers exist on project scheduling, the most recent one being a review on the variants and extensions of the RCPSP by Hartmann and Briskorn [56]. Pritsker et al. [104] presented the first mathematical model to solve the RCPSP using linear programming (LP). More recently, hybrid exact models have shown promising results. Berthold et al. [19] hybridized LP, CP, and satisfiability testing (SAT) into a single branch-and-bound algorithm. The problem is modelled using CP and the search process is enhanced with LP relaxation bounds and conflict analysis. CP is naturally suited to modelling the RCPSP due to the availability of interval variables and the `cumulative` global constraint [9]. However, the most common approach to solve the RCPSP is to use one of the two best known heuristics for project scheduling: the serial scheduling scheme, proposed by Kelley [70] and the parallel scheduling scheme, proposed by Brooks and also termed the "Brooks algorithm" [14]. Figure 3.1 shows several popular variants on the RCPSP and Table 3.3 presents existing literature for the RCPSP categorized by solution technique.

The serial method [70] consists of $|\mathcal{J}|$ stages where, in each stage, one activity is selected and scheduled. We maintain two disjoint sets of activities through the stages: the scheduled set $\mathcal{S}_n$, which

contains all activities scheduled so far that make up the partial schedule at stage n, and the decision set $\mathcal{D}_n$, which contains activities that have all their predecessors in the scheduled set. At stage n, an activity is selected from $\mathcal{D}_n$ using some priority rule and scheduled at its earliest possible time then moved to $\mathcal{S}_n$. Any activity not in either activity set that now has fulfilled precedence is then added to $\mathcal{D}_n$. Algorithm 1 formally describes the serial scheduling scheme. Let $K_{r,t}$ be the leftover capacity of renewable resource r in period t, $\mathcal{A}_t$ be the set of activities being processed in period $t \in \mathcal{T}$, and FT_j be the finish time of activity j. $K_{r,t}$ and $\mathcal{D}_n$ are defined as follows:

$$K_{r,t} = K_r - \sum_{j \in \mathcal{A}_t} k_{j,r} \tag{3.13}$$

$$D_n = \{j \mid j \notin \mathcal{S}_n, \mathcal{P}_j \subseteq \mathcal{S}_n\} \tag{3.14}$$

Algorithm 1: Serial Scheduling Scheme

Result: Feasible schedule using the serial scheduling scheme
$n \leftarrow 1$;
$\mathcal{S}_n \leftarrow \emptyset$;
while $|\mathcal{S}_n| < J$ **do**
  update $\mathcal{D}_n$ and $K_{r,t} \forall t \in \mathcal{T} \quad \forall r \in \mathcal{R}$;
  $j^* \leftarrow$ selected activity using some priority rule;
  $FT_{j^*} = \min\{t \mid t > \max\{FT_i \mid i \in \mathcal{P}_{j^*}\} + d_{j^*}, k_{j^*,r} \leq K_{r,\tau}, \tau \in \{t - d_{j^*} + 1, ..., t\}, r \in \mathcal{R}\}$;
  $\mathcal{S}_{n+1} \leftarrow \mathcal{S}_n \cup \{j^*\}$;
  $n \leftarrow n + 1$;
end

The parallel method [14] also consists of at most $|\mathcal{J}|$ stages. Each stage n is associated with a schedule time t_n where $t_m \leq t_n$ for all stages $m < n$. The set of scheduled activities is split into two subsets. The first, called the complete set $\mathcal{C}_n$ contains any activity that was scheduled and has completed processing before t_n. The second, called the active set $\mathcal{A}_n$, contains any activity that was scheduled but is still being processed at t_n. The set of decision activities, $\mathcal{D}_n$ now holds unscheduled activities that are available for scheduling with respect to precedence and resource capacity constraints. t_n is equal to the earliest completion time of activities in the active set of the previous stage. Algorithm 2 formally describes the parallel scheduling scheme. In each stage n, we update t_n and all activity sets. Then, activities in $\mathcal{D}_n$ are chosen using some priority rule and scheduled to start at t_n one by one until $\mathcal{D}_n$ is empty. $\mathcal{D}_n$ is thus redefined as follows:

$$D_n = \{j \mid j \notin \{\mathcal{S}_n \cup \mathcal{A}_n\}, \mathcal{P}_j \subseteq \mathcal{C}_n, k_{j,r} \leq \pi K_r \forall r \in \mathcal{R}\} \tag{3.15}$$

Many other heuristic methods that build on the serial and parallel scheduling schemes have also been developed. These heuristics range from metaheuristics such as genetic algorithms [55], tabu search [4], variable neighbourhood search [45], and scatter search [42] to multi-pass methods where multiple schedules are generated using priority rules [118, 21, 39]. According to a survey of heuristic solution techniques for the RCPSP conducted by Kolisch and Hartmann [72], the four best heuristics are all variations of genetic algorithms.

Algorithm 2: Parallel Scheduling Scheme

Result: Feasible schedule using the parallel scheduling scheme
$n \leftarrow 1$;
$t_n \leftarrow 0$;
$\mathcal{A}_n \leftarrow \emptyset$;
$\mathcal{C}_n \leftarrow \emptyset$;
while $|\mathcal{A}_n \cup \mathcal{C}_n| < |\mathcal{J}|$ **do**
  update t_n, $\mathcal{A}_n$, $\mathcal{C}_n$, $\mathcal{D}_n$, and $\pi K_r forall r \in \mathcal{R}$;
  while $\mathcal{D}_n \neq \emptyset$ **do**
    $j^* \leftarrow$ selected activity using some priority rule;
    $FT_{j^*} \leftarrow t_n + d_{j^*}$;
    $\mathcal{A}_n \leftarrow \mathcal{A}_n \cup \{j^*\}$;
  end
  $n \leftarrow n + 1$;
end

3.2.4 Constrained Clustering

Clustering is the task of dividing a set of data points into clusters such that points in the same cluster are more similar to each other than to points not in that cluster [127]. Clusters can be formed using several approaches. Xu and Tian [126] give a comprehensive survey of clustering algorithms and how they may be classified.

One recently defined variant on clustering is known as constrained clustering [11], or semi-supervised clustering. Traditionally, we do not know anything about relationships between points or how clusters should be formed in unsupervised clustering. However, there are many applications where we know that, for example, certain points should not be in the same cluster or clusters should be close to a certain size.

This information can be used to guide the clustering algorithm in quickly finding better clusters that adhere to the known constraints. The most common algorithm for unsupervised clustering is the k-means algorithm [86] and the most common algorithms for constrained clustering are modifications of the k-means algorithm. The first modified k-means clustering algorithm, denoted as *COP-KMEANS*, for constrained clustering was proposed by Wagstaff et al. [124]. Table 3.4 presents existing literature in constrained clustering categorized by types of constraint and Algorithm 3 formally defines the k-means algorithm.

Algorithm 3: K-means Algorithm

Result: Clusters of points using k-means
$\mathcal{P}$ is the set of points to be clustered;
$k \leftarrow$ initial number of clusters;
$\mathcal{C} \leftarrow$ randomly select k points from $\mathcal{P}$ to act as cluster centroids;
while *clusters not converged* ***and*** *iterations* $\leq$ *max* **do**
  compute the sum of squared distance between data points and all cluster centroids;
  assign data points to their closest cluster based on the previously calculated distances;
  update cluster centroids to be the average location of all data points assigned to the cluster;
end

Table 3.4: Types of constraints in constrained clustering.

Type of Constraint	Existing Literature
Pairwise Constraints	[124, 12, 129, 38, 79]
Cluster Size Constraints	[115, 8, 131, 130]
Cluster Cardinality Constraints	[109, 128]

Most of the work on constrained clustering has been focused on integrating pairwise constraints between points [11]: a Must-Link constraint says that two points must be in the same cluster and a Cannot-Link constraint requires those two points to be in different clusters. Two popular modified k-means algorithms that consider pairwise constraints are Partial Constrained k-means (*PCKmeans*) [12] and Partial Closure-Based Constrained k-means (*PCCKmeans*) [129]. Other approaches for pairwise constraints include CP [38] and spectral regularization [79].

Several authors have recently introduced constraints on cluster size and distribution where the size of a cluster is the number of points in the cluster and the distribution refers to the set of all cluster sizes. It is beneficial for many applications to be able to find balanced clusters, where clusters have very similar sizes. Various approaches to tackling the size constraints have been proposed, such as graph partitioning [115], using frequency sensitive competitive learning methods [8], and MIP [131]. Zhang et al. [130] presented a modified k-means algorithm, denoted as *KmeansS*, to consider cluster size constraints and, to the best of our knowledge, also presented the first algorithm, *PCS*, that integrated pairwise and size constraints. The PCS algorithm extends the PCKmeans algorithm to include size constraints.

3.2.5 Summary of Related Problems

We have reviewed the available literature in several disjoint fields of study. Each of these fields can be connected to the problem of scheduling in composites manufacturing in some aspect, implying that the solution techniques presented in this chapter may be combined and extended to fit the constraints of composites manufacturing. Figure 3.2 provides an overview of the connections between fields and the CMP.

At the top level in Figure 3.2, we can see that the batching and scheduling components of the CMP together can be constructed as a batch scheduling problem. Referring to Table 3.1, the CMP would be categorized as a four-stage hybrid flowshop where tool preparation, layup, and demould fall under the family batching model and curing falls under the batching machines model. However, we can also separate the CMP into individual batching and scheduling problems. The batching subproblem can be mapped to bin packing and constrained clustering. Referring to Table 3.2, batching in the CMP would be categorized as a two-dimensional level bin packing problems with both capacity (autoclave sizes) and cardinality constraints (tool quantities). Table 3.4 shows that batching in the CMP would be a constrained clustering problem with both size and cardinality constraints. The scheduling subproblem can be mapped to a resource-constrained project scheduling problem. Referring to Figure 3.1, scheduling in the CMP would have renewable resources and time-varying capacities.

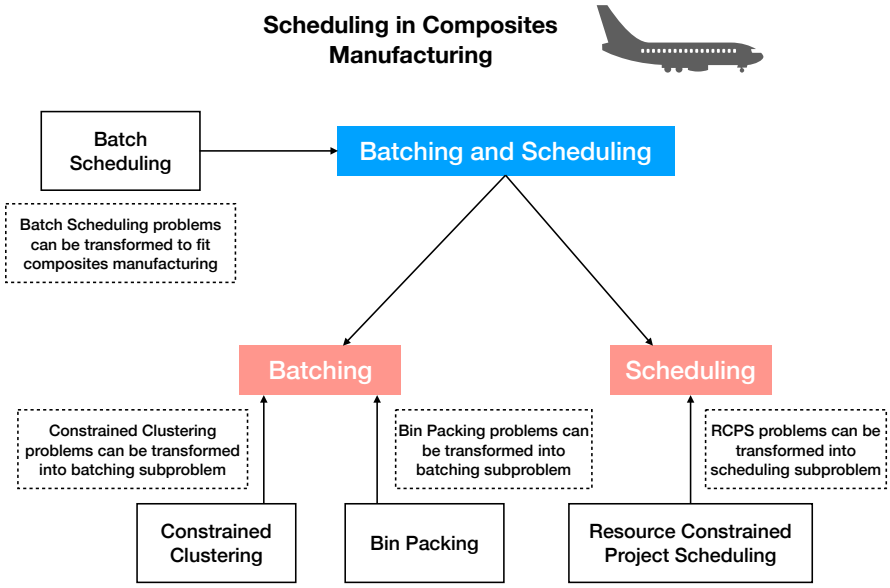


Figure 3.2: Overview of how different fields connect to the CMP.

Exact methods such as MIP, CP, and exact algorithms as well as heuristic methods of all varieties have been developed for all of the problems referenced in this chapter. However, to the best of our knowledge, no method utilizing CP or decomposition has been developed for complex batch scheduling problems such as the CMP. In addition, it appears that there have not been any attempts at combining techniques from batch scheduling, bin packing, project scheduling, and clustering in one piece of work.

3.3 Solution Technique Preliminaries

Next, we present some solution approaches and concepts required to understand the models and algorithms discussed in the majority of this thesis. Scheduling is the study of how to allocate scarce resources to jobs over time. The CMP can be considered a scheduling problem where jobs are competing for tool, machine, and labour resources within a constrained structure defined by resource availability and batching requirements. Thus, we will also discuss common methods for applying each approach to scheduling problems.

3.3.1 Mixed Integer Programming

One of the most popular approaches to solving discrete optimization problems is using *Mixed Integer Programming* (MIP). A MIP model in canonical form is expressed as [99]:

$$\begin{aligned} \max \quad & cx + c'y \\ \text{s.t.} \quad & Ax + Cy \leq b \\ & x \in \mathbb{Z}^n \end{aligned}$$

Where b is a vector, A and C are matrices, x is a vector of size n representing the integer decision variables, y is a vector of size m representing the continuous decision variables, and $cx + c'y$ is the objective function. If the set of integer variables x is empty, then the model can be classified as a *Linear Programming* model, and if the set of continuous variables y is empty, then the model can be classified as a *Integer Programming* model. The most popular method of solving MIP models is branch-and-bound [77], an exact tree search algorithm. Commercial solvers such as CPLEX [66], Gurobi [53], and Google OR-tools [100] implement their own versions of branch-and-bound.

There are two types of MIP models commonly used to solve scheduling problems: disjunctive models and time-indexed models. A *Disjunctive* model [88, 81] typically involves two sets of decision variables. First, $\{s_j \mid j \in \mathcal{J}, s_j \geq 0\}$ are continuous variables that represent the start times of jobs in the set of all jobs $\mathcal{J}$. Second, $\{z_{j,k} \mid j,k \in \mathcal{J}, z_{j,k} \in \{0,1\}\}$ are binary integer variables (i.e. variables that can only take values from $\{0,1\}$) that represent the decision to schedule job j before job k. $z_{j,k} = 1$ if job j completes before job k starts and $z_{j,k} = 0$ otherwise. A disjunctive MIP model uses the following constraints to ensure jobs do not overlap on a machine:

$$s_j \geq s_k + p_k - M \times z_{j,k} \qquad \forall j,k \in \mathcal{J}, j \leq k \tag{3.16}$$

$$s_k \geq s_j + p_j - M \times (1 - z_{j,k}) \qquad \forall j,k \in \mathcal{J}, j \leq k \tag{3.17}$$

Where p_j (p_k) is the processing time of job j (k) and M is a sufficiently large number. Constraint (3.16) ensures that the start time of j is after the end time of k if $z_{j,k} = 0$. Constraint (3.17) ensures that the start time of k is after the end time of j if $z_{j,k} = 1$. Constraints (3.16) and (3.17) are also known as *big-M* constraints.

The second type of MIP model commonly used for scheduling is the *Time-Indexed* model [23, 73]. The disjunctive model focused on deciding when to schedule jobs based on their order, i.e. job j comes before job k or vice versa. The time-indexed model decides when the schedule jobs based on available time slots in the schedule. Here, we have a set of binary variables $\{x_{j,t} \mid j \in \mathcal{J}, t \in \mathcal{T}\}$ where $x_{j,t} = 1$ if job j is scheduled to start at time t and $x_{j,t} = 0$ otherwise. The set $\mathcal{T}$ contains the discretized time periods $\{0, 1, ..., T\}$ that make up the fixed scheduling horizon. A time-indexed MIP model uses the following constraints to ensure jobs do not overlap on a machine:

$$\sum_{t=0}^{T-p_j+1} x_{j,t} = 1 \qquad j \in \mathcal{J} \tag{3.18}$$

$$\sum_{j \in \mathcal{J}} \sum_{t'=t-p_j+1}^{t} x_{j,t'} \leq 1 \qquad \forall t \in \mathcal{T} \tag{3.19}$$

Where p_j is the processing time of job j. Constraint (3.18) makes sure each job is scheduled to start once. Constraint (3.19) makes sure that at any time, at most one task is scheduled.

One drawback to using MIP for scheduling problems is that MIP models usually have poor scalability and are ill-suited to representing more complex scheduling constraints, such as multi-capacity resources or multi-stage shop problems. We are limited by the restrictions of MIP variables and constraints.

3.3.2 Constraint Programming

Constraint Programming (CP) is an approach developed by the artificial intelligence community and established as a successful paradigm for solving hard combinatorial optimization problems including scheduling problems [108, 9]. To use CP as a solution technique, problems with objective functions are first defined as constraint optimization problems (COP) [9]. Formally, a COP consists of a four-tuple ($\mathcal{X}$, $\mathcal{D}$, $\mathcal{C}$, Z) where $\mathcal{X} = \{x_1, x_2, ..., x_n\}$ is a set of n decision variables, $\mathcal{D} = \{D_1, D_2, ..., D_n\}$ are the domains of the variables in $\mathcal{X}$, $\mathcal{C} = \{c_1, c_2, ..., c_m\}$ is a set of m constraints acting on variables in $\mathcal{X}$, and Z is the objective function to be minimized or maximized. A solution to the COP is a complete assignment of values to variables in $\mathcal{X}$ that satisfies all constraints in $\mathcal{C}$ and evaluates Z to its global minimum or maximum value.

Like MIP, a COP can be solved using branch-and-bound [58]. One of the central ideas of the CP methodology is applying constraint propagation algorithms at each node in the branch-and-bound search tree to perform domain reduction and enforce consistency [62]. Strong propagation algorithms can significantly reduce the search tree size by pruning large subtrees.

A constraint system $\mathcal{C}$ can be *consistent* in several aspects. Arc consistency is the most basic form of consistency; the variables x_i and x_j are considered to be arc-consistent with a binary constraint $C_k(x_i, x_j) \in \mathcal{C}$ if, for every value a in D_i (the domain of x_i), there exists a value b in D_j (the domain of x_j) such that (a, b) satisfies $C_k(x_i, x_j)$, and vice versa [108]. If two variables are not arc-consistent with some constraint, we can remove values that do not satisfy the above condition from the domains of those variables until they either become arc-consistent or we prove the problem is infeasible by removing all values from one of the domains. Figure 3.3 shows how two variables can be pruned to be arc-consistent with a binary constraint. A constraint propagation algorithm can make the entire COP arc-consistent by repeating this process for all variable pairs until we cannot prune values from any more domains. Basic arc consistency for binary constraints can be extended to all constraints, known as generalized arc-consistency [108].

More specialized constraint propagation algorithms, known as global constraints [107], have also been developed throughout the years. Such algorithms focus on exploiting certain problem structures

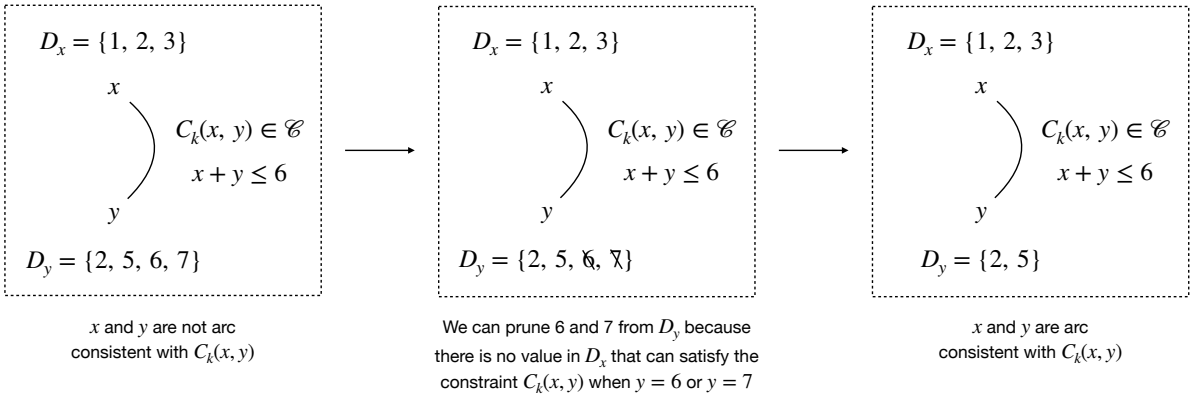


Figure 3.3: Enforcing arc consistency between two variables and their binary constraint.

that commonly appear in combinatorial optimization problems. For example, the *allDifferent* global constraint enforces that each variable in a given set $\{x_1, x_2, ..., x_m\}$ must take on a distinct value by enforcing generalized arc-consistency over the given variables and their constraints.

After consistency is maintained at a node, we can heuristically give a value to an unassigned variable and continue to the next node in the search process where we can apply propagation again. However, if a variable ends up with an empty domain, the current node is infeasible and can be pruned, along with its subtree.

CP has been successful in solving many different types of scheduling problems [10]. An interval variable represents an interval of time during which a job is processed. Formally, an interval variable a is a variable whose domain $dom(a)$ is a subset of $\{\perp\} \cup \{[s, e) \mid s, e \in \mathbb{Z}, s \leq e\}$ [67]. An interval variable is absent when $a = \perp$ and the variable is present when $a = [s, e)$. The availability of interval variables in many modern CP solvers allows us to easily model complex scheduling problems [10]. We can enforce constraints on start and end times of interval variables without having to create the complex auxiliary variables necessary when modelling with MIP.

In addition, there are many global constraints that can be used to model complex scheduling behaviours. For example, cumulative constraints, introduced by Aggoun and Beldiceanu [1], can be used to model multi-capacity resources and are based on applying inequality constraints to a cumulative function expression. A cumulative function expression F is the algebraic sum of multiple elementary function expressions where each elementary function expression represents either a consumption or restoration of resources by an interval variable [111]. Figure 3.4 shows examples of elementary function expressions and how they can be summed together. The *alwaysIn* global constraint, written as `alwaysIn`$(F, u, v, h_{min}, h_{max})$ is a cumulative constraint that takes a cumulative function expression F and makes sure any value of F must always fall within the range $[h_{min}, h_{max}]$ anywhere on the time interval defined by $[u, v)$. There are many other global constraints that can be applied to interval variables [13, 123], making CP an attractive choice for solving scheduling problems.

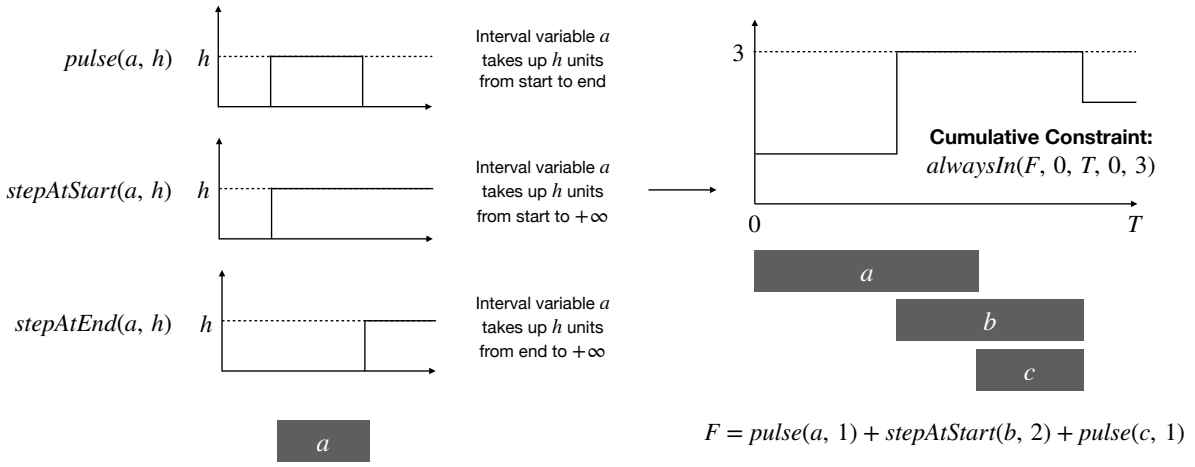


Figure 3.4: Elementary function expressions: `pulse`, `stepAtStart`, and `stepAtEnd` and an example of how the cumulative constraint `alwaysIn` is used to constrain resource usage over time. Interval variables a, b, and c use the same resource, which has a capacity of 3, and are scheduled from $\{0, T\}$.

3.3.3 Decomposition Techniques

Techniques that decompose a large problem into subproblems have been studied for years across many fields of study [121]. We can categorize decomposition techniques as general or problem-specific. In integer programming, Dantzig and Wolfe presented the famous Dantzig-Wolfe decomposition technique [37] and Benders followed with the classical Benders decomposition technique [18]. The third well-known decomposition algorithm in integer programming is Lagrangian decomposition, proposed by Guignard and Kim [51]. These decomposition techniques are general and can be applied to any problem with a suitable structure as defined in their respective problem definitions. In contrast, Potts and Wassenhove [102] presented one of the first problem-specific decomposition algorithms for job shop scheduling where an algorithm is used to separate the problem into subproblems and those subproblems are then solved using dynamic programming. Ovacik and Uzsoy [98] compiled a review on problem-specific decomposition methods designed for job shop scheduling problems.

Decomposition techniques have also been developed specifically for the application of the batching machines model. Sung et al. [116] proposed a heuristic decomposition algorithm to solve a flowshop consisting of n batching machines in series. The heuristic decomposes the problem into two-stage flowshop subproblems that are solved using dynamic programming.

Logic-based Benders decomposition (LBBD) was introduced and formally developed by Hooker [61]. This technique is unique in that, like classical Benders decomposition, the format is general and can be applied to any problem with the correct structure. However, the structure criteria is much looser for LBBD and the division of subproblems is reliant on exploiting problem-specific characteristics. Next, we will formally define the LBBD technique based on Hooker [61].

Let us consider a general optimization problem:

$$\min\{f(x) \mid C(x), x \in \mathcal{D}\} \tag{3.20}$$

where $C(x)$ is the constraint set containing variables $x \in \mathcal{X}$ and $\mathcal{D}$ is the domain of x. We define the inference dual as another optimization problem with the objective of finding the tightest lower bound v on the primal objective:

$$\max\left\{v \mid C(x) \overset{P}{\vdash} (f(x) \geq v), v \in \mathbb{R}, P \in \mathcal{P}\right\} \tag{3.21}$$

where $C(x) \overset{P}{\vdash} (f(x) \geq v)$ says that proof P implies the expression $f(x) \geq v$ from constraint set $C(x)$. Any solution v to the inference dual is therefore a lower bound on the primal objective. We now define the LBBD technique as applied to a problem of the form:

$$\min\{f(x,y) \mid C(x,y), C^{'}(x), x \in \mathcal{D}_{\S}, y \in \mathcal{D}_{\dagger}\} \tag{3.22}$$

where the set $C^{'}(x)$ is the subset of constraints that only contain the variable x. If we fix the variable x to be $\overline{x}$, we end up with a subproblem of the original problem:

$$\min\{f(\overline{x},y) \mid C(\overline{x},y), y \in \mathcal{D}_{\dagger}\} \tag{3.23}$$

We define the inference dual of the subproblem as:

$$\max\left\{v \mid C(\overline{x},y) \overset{P}{\vdash} (f(\overline{x},y) \geq v), v \in \mathbb{R}, P \in \mathcal{P}\right\} \tag{3.24}$$

If we assume v^* is the optimal solution to the subproblem and proof P^* implies the bound $f(\overline{x},y) \geq v^*$, then this same proof may also imply a bound for other fixed values of x. Thus, a Benders cut is defined as $z \geq B_{\overline{x}}(x)$ where z is the primal objective and $B_{\overline{x}}(\overline{x}) = v^*$. A Benders cut is added to the master problem at each iteration of the procedure; the master problem at iteration k is thus defined as:

$$z_k = \min\{z \mid C^{'}(x), z \geq B_{x^i}(x), i \in \{1,...,k\}, x \in \mathcal{D}_{\S}\} \tag{3.25}$$

where x^i is the solution in iteration i to the master problem. The LBBD procedure terminates when the optimal value of the master problem and the optimal value of any previously solved subproblem are equal.

LBBD has been applied to many scheduling problems [63, 31]. Hooker [63] formulated LBBD approaches for assigning then scheduling a set of tasks in a set of facilities with three different objectives: minimizing cost, minimizing makespan, and minimizing total tardiness. The master problem consists of assigning tasks to facilities and then the remaining constraints decompose into several subproblems, each one responsible of scheduling tasks in a facility. As aforementioned, LBBD is reliant on exploiting problem-specific characteristics, and we see this when each of the three objectives necessitates the development of unique Benders cuts and proofs.

Emde et al. [43] applied LBBD to the single batching machine problem with objective of minimizing maximal lateness. The master problem is formulated using MIP and finds an assignment of jobs to

batches without any scheduling considerations. The subproblem finds the minimum maximal lateness given the fixed assignment of jobs to batches from the master and is solved using an algorithm. To the best of our knowledge, there do not appear to be any other attempts at using LBBD to solve batch scheduling problems.

3.3.4 Heuristics

A common drawback of exact solution approaches like MIP and CP is scalability. The methodical search process of branch-and-bound takes longer to execute as problem instance sizes grow. Ultimately, many MIP and CP models are unable to find good solutions within a reasonable time limit when faced with real-world instances [64]. This drawback has motivated the development of heuristic techniques that do not provide any optimality guarantees but are often much faster than exact models. Most heuristic algorithms are ad-hoc approaches that look to exploit specific structures inherent to the problems they are solving [27]. Metaheuristics [49] make up a class of more general heuristic methods that can be applied to many different problems. Popular metaheuristics include simulated annealing, tabu search, and genetic algorithms.

One of the most widely used metaheuristics is a *Genetic Algorithm*. Introduced by Holland [60], genetic algorithms are inspired by the theory of evolution and use biologically inspired operations such as reproduction, mutation, and survival of the fittest to generate high-quality solutions from an initial pool of lower-quality solutions. For the remainder of this section, we will provide a more in-depth explanation on the basics of genetic algorithms to provide the necessary background for understanding a solution approach introduced in Chapter 6.

There are five phases in a basic genetic algorithm:

1. Initialize population
2. Define and apply a fitness function
3. Select reproducing pairs
4. Reproduce by crossover
5. Mutate reproduced children

The first step is to initialize the population with a set of individuals, where each individual represents a unique solution to the problem being solved. An individual is characterized by a set of parameters known as genes that corresponds to the decision variables of the problem. For example, if a problem has a set of binary decision variables $\{x \in \{x_0, x_1, x_2, x_3\} \mid x \in \{0, 1\}\}$, then a gene would correspond to one of $\{x_0, x_1, x_2, x_3\}$ and each gene can only take on values from $\{0, 1\}$. The set of genes belonging to an individual is also known as a chromosome.

Next, a fitness function needs to be defined that takes an individual's chromosome as the input and gives the individual's "fitness" as the output. A fitness function is analogous to the objective function of a MIP or CP model. Fitter individuals have a higher probability of being selected in phase 3 to reproduce. Each reproducing pair, known as parents, produce two offspring by exchanging genes through

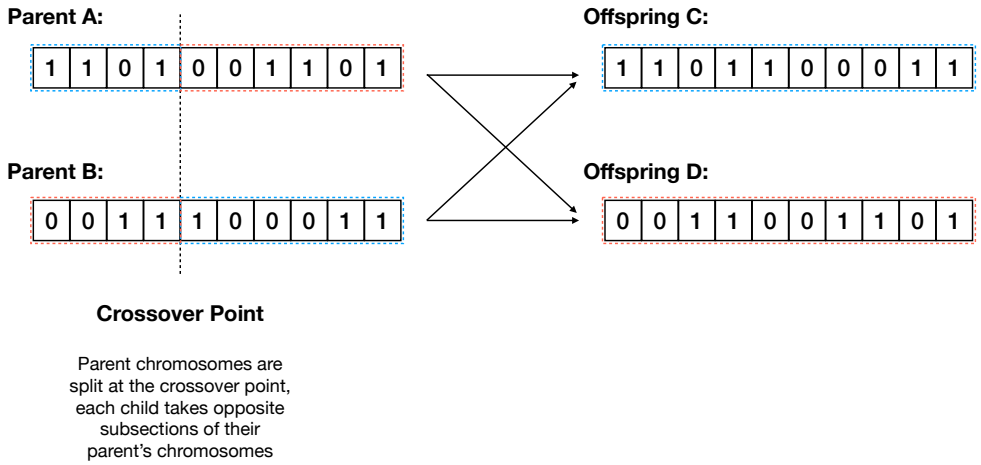


Figure 3.5: One-point crossover reproduction.

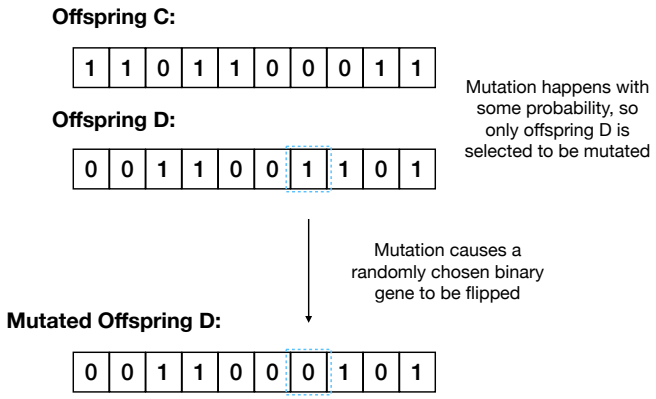


Figure 3.6: Example of genetic mutation of a chromosome with binary encoding.

a crossover operation. The most basic crossover operation is one-point crossover, shown in Figure 3.5.

Offspring produced in phase 4 are subject to mutation of their genes with some probability. Figure 3.6 shows an example of a mutation in a chromozome. Mutation occurs to prevent premature convergence and provide slight perturbations that maintain genetic diversity. Once all reproducing pairs produce offspring, the same fitness function as defined in phase 2 is applied to new individuals in the population. Then, a certain number of the least fit individuals are removed and phases 3 to 5 are repeated for the new population. A genetic algorithm terminates if the population converges (i.e. offspring are no longer significantly different from the previous generation).

A binary encoding of genes, as shown in the preceding example, is often the best choice when developing the genetic representation. However, for scheduling problems, it is not clear in advance which encoding strategy is best [125]. A popular encoding strategy is an operation-based representation, where the a gene (j, m) represents the order which job j is processed on machine m. For example if gene $(1, 5)$ has an index of 3 in its chromosome, its corresponding solution would have job 1 scheduled in third place after two other jobs on machine 5. Werner [125] provides a comprehensive survey of genetic

algorithms and their applications to scheduling problems.

3.4 Summary

In this chapter, we presented background on problems related to the composites manufacturing problem (CMP) and common methodologies for solving combinatorial problems. First, we introduced related problems from the areas of batch scheduling, bin packing, constrained clustering, and resource-constrained project scheduling. All four areas of research can be mapped to the CMP or a subproblem of the CMP. The foundations of Mixed Integer Programming (MIP) and Constraint Programming (CP) were explained, with extra commentary on how MIP and CP are commonly used to solve scheduling problems. We also presented an overview of decomposition approaches, focusing on Logic-based Benders Decomposition, and closed the chapter with a presentation of heuristic methods, focusing on genetic algorithms.

Chapter 4

Solving an Abstraction of the Composites Manufacturing Problem

This chapter formally defines an abstraction of the Composites Manufacturing Problem (CMP) and presents five solution techniques: Mixed Integer Programming (MIP), Constraint Programming (CP), two variations of Logic-based Benders Decomposition (LBBD), and an Earliest Due Date heuristic (EDD). We show the full models/algorithms for each solution technique, then close the chapter with numerical results and an analysis and discussion of those results.

4.1 Abstracted Problem Definition

The full CMP is a complex and multi-layered problem that is difficult to model, thus, it is worthwhile to define an abstracted problem to test different solution approaches. This abstracted problem will include the key complexities of the problem, namely the bin packing and hybrid flow shop scheduling aspects of the CMP, and will only focus on the dominating stages of the CMP (layup and curing). We designate the abstracted problem to be the *Two-Stage Bin Packing and Hybrid Flowshop Scheduling Problem* (2BPHFSP).

The 2BPHFSP is described as follows. We are given a set of jobs $\mathcal{J}$, a set of empty tool batches $\mathcal{B}^1$, and a set of empty autoclave batches $\mathcal{B}^2$. There exists three sets of resources: an unlimited set of tools $\mathcal{T}$, a set of unary-capacity stage 1 machines $\mathcal{M}^1$, and a set of unary-capacity stage 2 machines $\mathcal{M}^2$. Each job $j \in \mathcal{J}$ is associated with a one-dimensional size parameter s_j, a due date d_j, and a processing time p_j, and each tool $t \in \mathcal{T}$ is associated with a one-dimensional size parameter v_t. Each job must be assigned to a tool batch, and each tool batch must be assigned to a specific tool such that the sum of job sizes in that tool batch is less than or equal to the size of the assigned tool. The size of a tool batch $k \in \mathcal{B}^1$ is the size of its assigned tool. Each tool batch must be aggregated into autoclave batches such that the sum of tool sizes is less than or equal to the autoclave capacity.

The upper bound on the number of tool batches and autoclave batches is the number of jobs, where each job is assigned to its own tool batch and each tool batch is assigned to its own autoclave batch. Thus, we create $|\mathcal{J}|$ empty batches in each of $\mathcal{B}^1$ and $\mathcal{B}^2$. However, we almost always find solutions that

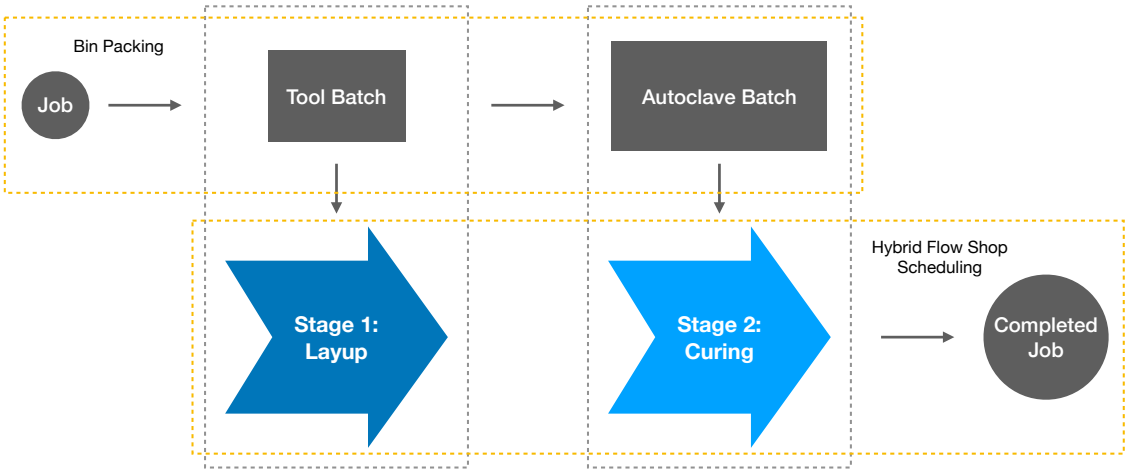


Figure 4.1: Job flow in the 2BPHFSP.

use fewer batches, so we denote any non-empty batch as *open*.

Once packing is completed, we need to schedule each open tool batch on a stage 1 machine and each open autoclave batch on a stage 2 machine. As defined for the CMP, the processing time of a tool batch is the sum of processing times of its jobs. The processing time of an autoclave batch is simplified from the CMP to be uniform across all batches, thus ignoring cycle types. There are precedence constraints between the end of tool batches and the start of their assigned autoclave batches. Figure 4.1 shows the flow of a job in the 2BPHFSP.

The objective of 2BPHFSP is to minimize a weighted sum of the number of open autoclave batches and job tardiness. We minimize the number of open autoclave batches to decrease autoclave operational costs, and we minimize job tardiness to ensure parts are delivered to downstream machines on time. Although we are relaxing many aspects of the CMP, such as the existence of slots on tools and the limits on resources (i.e. tools, labour, etc.), the 2BPHFSP encompasses the major ideas from items 1 and 3 from the key complexities presented in Section 2.1.1.

4.2 Solution Approaches

In this section, we present the mathematical models and pseudocode for our five approaches.

4.2.1 Mixed Integer Programming

First, we present the monolithic mixed integer programming formulation, designated as *MIP*. We chose to pursue a time-indexed formulation [23] due to the multi-capacity nature of the two stages. There are two sets of binary decision variables that are related to bin packing and two sets of binary decision variables that are related to flow shop scheduling. Model variables and parameters are shown in Table 4.1.

Table 4.1: MIP parameters and variables.

Parameter	Description
$j \in \mathcal{J}$	Set of jobs
$k \in \mathcal{B}^1$	Set of tool batches
$i \in \mathcal{B}^2$	Set of autoclave batches
$t \in \mathcal{T}$	Set of tools
$n \in \mathcal{H}$	Set of time points
s_j	Size of job j
p_j	Processing time of job j
d_j	Due date of job j
v_t	Size of tool t
V^1	Maximum capacity of tool batches
V^2	Maximum capacity of autoclave batches
P^2	Processing time of autoclave batches
C^1	Number of stage 1 machines
C^2	Number of stage 2 machines
α_j	Weighting for tardiness of job j in the objective function
β	Weighting for an open autoclave batch in the objective function

Variable	Description
$x_{j,k}$	Binary variable, 1 if job j is in tool batch k, 0 otherwise
$y_{k,t,i}$	Binary variable, 1 if tool batch k in on tool t and in autoclave batch i, 0 otherwise
$\gamma_{k,n}$	Binary variable, equals 1 if tool batch k starts at time n
$\sigma_{i,n}$	Binary variable, equals 1 if autoclave batch i starts at time n
$a_{k,n}$	Binary variable, equals 1 if tool batch k ends at time n
$b_{i,n}$	Binary variable, equals 1 if autoclave batch i ends at time n
τ_k	Size of tool assigned to tool batch k
f_j	Final end time of job j
l_j	Tardiness of job j
o_k	Indicator if tool batch k is open
q_i	Indicator if autoclave batch i is open

The bin packing variables consists of two sets of binary decision variables. The first set contains two-indexed variables that assign of jobs to tool batches, $x_{j,k}$. The second set contains three-indexed variables that assign tool batches to tools as well as autoclave batches, $y_{k,t,i}$. This separation of stages is inspired by the multi-stage bin packing model presented by Puchinger et al. [105] (see Section 3.2.2). In particular, their model uses indicators to represent if a bin is open. Therefore, we introduce two sets of binary indicator variables, o_k and q_i, that equal 1 if tool batch k or autoclave batch i is open, respectively.

The scheduling variables consist of two sets of binary decision variables that determine the start times of tool and autoclave batches. Each stage has multiple identical unary-stage machines, so we can represent each stage as a multi-capacity resource. Two sets of binary variables are introduced as auxiliary variables to indicate the end times of tool and autoclave batches, depending on the start time and processing time of each batch. To model the multi-capacity nature of each stage in the flow shop in a time-indexed MIP model (see Section 3.3.1), we must add constraints that limit the total number of batches processed at any time point to be less than the respective stage's capacity. These constraints are written using sums of the binary variables indicating start and end times.

This model will be presented in two sections. The first section will only contain packing constraints and the second section will only contain scheduling constraints.

$$\min \quad \beta \sum_{i \in \mathcal{B}^2} q_i + \sum_{j \in \mathcal{J}} \alpha_j l_j \tag{4.1}$$

$$\text{s.t.} \quad \sum_{k \in \mathcal{B}^1} x_{j,k} = 1 \qquad \forall j \in \mathcal{J} \tag{4.2}$$

$$o_k \geq x_{j,k} \qquad \forall j \in \mathcal{J} \quad \forall k \in \mathcal{B}^1 \tag{4.3}$$

$$o_k \leq \sum_{j \in \mathcal{J}} x_{j,k} \qquad \forall k \in \mathcal{B}^1 \tag{4.4}$$

$$\sum_{t \in \mathcal{T}} \sum_{i \in \mathcal{B}^2} y_{k,t,i} = o_k \qquad \forall k \in \mathcal{B}^1 \tag{4.5}$$

$$q_i \geq \sum_{t \in \mathcal{T}} y_{k,t,i} \qquad \forall k \in \mathcal{B}^1 \quad \forall i \in \mathcal{B}^2 \tag{4.6}$$

$$q_i \leq \sum_{k \in \mathcal{B}^1} \sum_{t \in \mathcal{T}} y_{k,t,i} \qquad \forall i \in \mathcal{B}^2 \tag{4.7}$$

$$o_k \geq o_{k+1} \qquad \forall k \in \{0, ..., |\mathcal{B}^1| - 1\} \tag{4.8}$$

$$q_i \geq q_{i+1} \qquad \forall i \in \{0, ..., |\mathcal{B}^2| - 1\} \tag{4.9}$$

$$\tau_k = \sum_{t \in \mathcal{T}} \sum_{i \in \mathcal{B}^2} v_t y_{k,t,i} \qquad \forall k \in \mathcal{B}^1 \tag{4.10}$$

$$\sum_{j \in \mathcal{J}} s_j x_{j,k} \leq \tau_k \qquad \forall k \in \mathcal{B}^1 \tag{4.11}$$

$$\sum_{t \in \mathcal{T}} \sum_{k \in \mathcal{B}^1} v_t y_{k,t,i} \leq V^2 \qquad \forall i \in \mathcal{B}^2 \tag{4.12}$$

$$x_{j,k} \in \{0,1\} \ \forall j \in \mathcal{J}, \ \forall k \in \mathcal{B}^1; \ y_{k,t,i} \in \{0,1\} \ \forall k \in \mathcal{B}^1, \ \forall t \in \mathcal{T}, \ \forall i \in \mathcal{B}^2$$

$$\tau_k \in \{0, ...V^1\} \ \forall k \in \mathcal{B}^1; \ o_k \in \{0,1\} \ \forall k \in \mathcal{B}^1; \ q_i \in \{0,1\} \ \forall i \in \mathcal{B}^2$$

Constraint (4.2) assigns each job to one tool batch. Constraints (4.3) and (4.4) enforce that the indicator variable o_k is equal to 1 if tool batch k is open, and equal to 0 otherwise. Constraint (4.5) assigns each open tool batch to one autoclave batch. Constraints (4.6) and (4.7) enforce that the indicator variable q_i is equal to 1 if autoclave batch i is open, and equal to 0 otherwise. Constraints (4.8) and (4.9) open tool and autoclave batches in order of index, to reduce symmetry. Constraint (4.10) defines each tool batch's size, and Constraint (4.11) makes sure the sum of job sizes in each tool batch is less than or equal to its size. Constraint (4.12) constrains the sum of tool sizes in each autoclave batch to be less than the autoclave capacity.

Next, we present the scheduling constraints.

$$\sum_{n\in\mathcal{H}} \gamma_{k,n} = o_k \qquad \forall k \in \mathcal{B}^1 \tag{4.13}$$

$$\sum_{n\in\mathcal{H}} a_{k,n} = o_k \qquad \forall k \in \mathcal{B}^1 \tag{4.14}$$

$$\sum_{n\in\mathcal{H}} n a_{k,n} = \sum_{n\in\mathcal{H}} n\gamma_{k,n} + \sum_{j\in\mathcal{J}} p_j x_{j,k} \qquad \forall k \in \mathcal{B}^1 \tag{4.15}$$

$$\sum_{n\in\mathcal{H}} \sigma_{i,n} = q_i \qquad \forall i \in \mathcal{B}^2 \tag{4.16}$$

$$\sum_{n\in\mathcal{H}} b_{i,n} = q_i \qquad \forall i \in \mathcal{B}^2 \tag{4.17}$$

$$\sum_{n\in\mathcal{H}} n b_{i,n} = \sum_{n\in\mathcal{H}} n\sigma_{i,n} + P^2 \qquad \forall i \in \mathcal{B}^2 \tag{4.18}$$

$$\sum_{n\in\mathcal{H}} n\sigma_{i,n} \geq \sum_{n\in\mathcal{H}} n a_{k,n} - |\mathcal{H}|\left(1 - \sum_{t\in\mathcal{T}} y_{k,t,i}\right) \qquad \forall k \in \mathcal{B}^1, \quad \forall i \in \mathcal{B}^2 \tag{4.19}$$

$$\sum_{k\in\mathcal{B}^1} \sum_{n^*\in\mathcal{I}} (\gamma_{k,n^*} - a_{k,n^*}) \leq C^1 \qquad \mathcal{I} = \{0, ..., n\}, \quad \forall n \in \mathcal{H} \tag{4.20}$$

$$\sum_{i\in\mathcal{B}^2} \sum_{n^*\in\mathcal{I}} (\sigma_{i,n^*} - b_{i,n^*}) \leq C^2 \qquad \mathcal{I} = \{0, ..., n\}, \quad \forall n \in \mathcal{H} \tag{4.21}$$

$$f_j \geq \sum_{n\in\mathcal{H}} n b_{i,n} - |\mathcal{H}|\left(2 - x_{j,k} - \sum_{t\in\mathcal{T}} y_{k,t,i}\right) \qquad \forall j \in \mathcal{J}, \quad \forall k \in \mathcal{B}^1, \quad \forall i \in \mathcal{B}^2 \tag{4.22}$$

$$l_j \geq f_j - d_j \qquad \forall j \in \mathcal{J} \tag{4.23}$$

$$\gamma_{k,n} \in \{0,1\}\ \forall k \in \mathcal{B}^1,\ \forall n \in \mathcal{H};\ a_{k,n} \in \{0,1\}\ \forall k \in \mathcal{B}^1,\ \forall n \in \mathcal{H};\ \sigma_{i,n} \in \{0,1\}\ \forall i \in \mathcal{B}^2,\ \forall n \in \mathcal{H}$$

$$b_{i,n} \in \{0,1\}\ \forall i \in \mathcal{B}^2,\ \forall n \in \mathcal{H};\ f_j \in \{0, ...|\mathcal{H}|\}\ \forall j \in \mathcal{J};\ l_j \in \{0, ...|\mathcal{H}|\}\ \forall j \in \mathcal{J}$$

Constraints (4.13) to (4.15) make sure each tool batch starts and ends once, and is processed for the appropriate length of time. Constraints (4.16) to (4.18) make sure each autoclave batch starts and ends once, and is processed for the appropriate length of time. Constraint (4.19) says that each tool batch must finish processing before its associated autoclave batch can start. Constraint (4.19) is a big-M constraint with $M = |\mathcal{H}|$ because the packing decisions determine which batches need to have precedence. If a tool batch k is packed into an autoclave batch i, then the two constraints need to be enforced for (k, i). However, the two constraints for all $\{(k, i^*) \mid i^* \in \mathcal{B}^2, i^* \neq i\}$ should not be enforced. Therefore, (4.19) needs to be a big-M constraint. Constraints (4.20) and (4.21) enforce that the total number of

Table 4.2: CP variables.

Variable	Description
$x_{j,k}$	Interval variable for job j in tool batch k
$y_{k,i}$	Interval variable for tool batch k in autoclave batch i
γ_k	Interval variable for tool batch k
σ_i	Interval variable for autoclave batch i
t_k	Tool assigned to tool batch k
τ_k	Size of tool assigned to tool batch k
l_j	Tardiness of job j

tool batches or autoclave batches being processed at one time is at most the respective stage's capacity. Constraint (4.22) defines the final end time, i.e. the time at which its autoclave batch is completed, for each job. Constraint (4.23) defines job tardiness.

4.2.2 Constraint Programming

Next, we present the monolithic constraint programming formulation, designated as *CP*. This model uses two sets of interval decision variables for bin packing and two sets of interval decision variables for scheduling. Model variables are shown in Table 4.2; see Table 4.1 for model parameters.

The first set of bin packing variables are optional two-indexed interval variables, $x_{j,k}$. We assign job j to a tool batch k by forcing one variable from $\{x_{j,k} \mid k \in \mathcal{B}^1\}$ to be present for each job j. The same concept is applied to the second set of bin packing variables, $y_{k,i}$, where tool batch k is assigned to autoclave batch i when $y_{k,i}$ is present. We use two sets of single-index interval variables for scheduling batches on the horizon $\mathcal{H}$, γ_k for tool batches and σ_i for autoclave batches. We also define a set of variables t_k that assigns tool batch k to a specific tool so we can calculate the size of k, τ_k. Figure 4.2 uses an example to illustrate the connections between variables in the monolithic CP model.

Similar to the monolithic MIP model, this model will be presented in two sections. The first section will focus on packing and the second section will focus on scheduling.

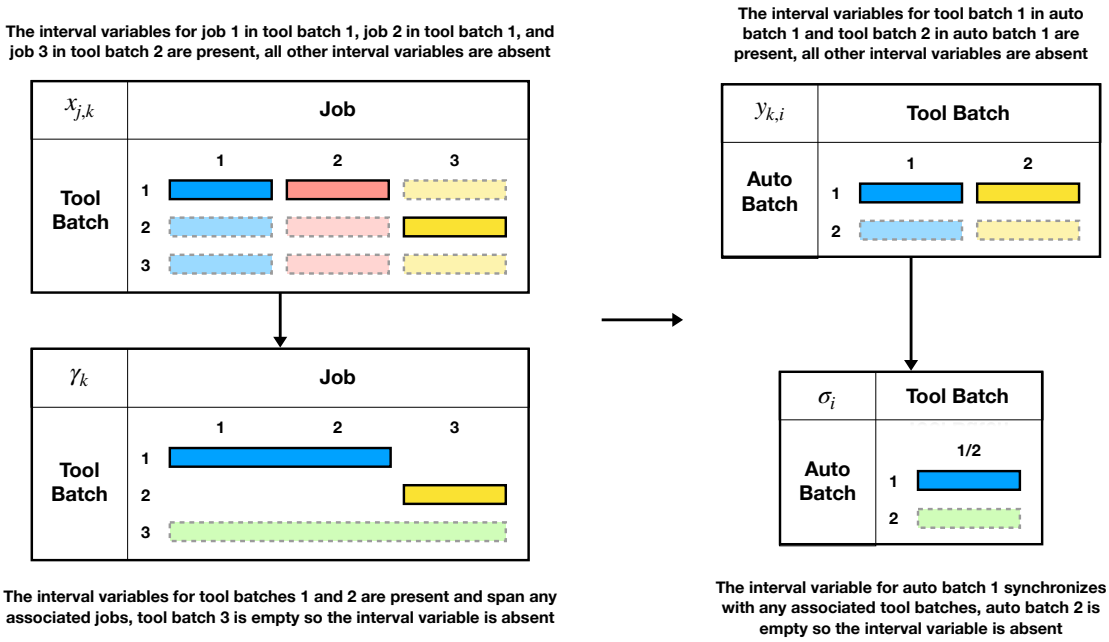


Figure 4.2: Connections between variables and their meanings in the monolithic CP model.

$$\min \quad \beta \sum_{i \in \mathcal{B}^2} \texttt{presenceOf}(\sigma_i) + \sum_{j \in \mathcal{J}} \alpha_j l_j \tag{4.24}$$

$$\text{s.t.} \quad \sum_{k \in \mathcal{B}^1} \texttt{presenceOf}(x_{j,k}) = 1 \qquad \forall j \in \mathcal{J} \tag{4.25}$$

$$\sum_{i \in \mathcal{B}^2} \texttt{presenceOf}(y_{k,i}) = \texttt{presenceOf}(\gamma_k) \qquad \forall k \in \mathcal{B}^1 \tag{4.26}$$

$$\texttt{presenceOf}(\gamma_k) \geq \texttt{presenceOf}(\gamma_{k+1}) \qquad \forall k \in \{0, ..., |\mathcal{B}^1| - 1\} \tag{4.27}$$

$$\texttt{presenceOf}(\sigma_i) \geq \texttt{presenceOf}(\sigma_{i+1}) \qquad \forall i \in \{0, ..., |\mathcal{B}^2| - 1\} \tag{4.28}$$

$$\tau_k = \texttt{element}(\{v_t \mid t \in \mathcal{T}\}, t_k) \qquad \forall k \in \mathcal{B}^1 \tag{4.29}$$

$$\sum_{j \in \mathcal{J}} s_j \times \texttt{presenceOf}(x_{j,k}) \leq \tau_k \qquad \forall k \in \mathcal{B}^1 \tag{4.30}$$

$$\sum_{k \in \mathcal{B}^1} \tau_k \times \texttt{presenceOf}(y_{k,i}) \leq V^2 \qquad \forall i \in \mathcal{B}^2 \tag{4.31}$$

$$t_k \in \{0, ...\mathcal{T}\} \; \forall k \in \mathcal{B}^1; \; \tau_k \in \{0, ...V^1\} \; \forall k \in \mathcal{B}^1$$

Constraint (4.25) assigns each job to one tool batch. Constraint (4.26) assigns each open tool batch to one autoclave batch. Constraints (4.27) and (4.28) enforce that tool and autoclave batches are opened in order of index, to reduce symmetry. Constraint (4.29) is an element constraint, and picks the tool size parameter associated with tool $t_k \in \mathcal{T}$ to assign to tool batch k.[1] Constraint (4.30) makes sure the sum of job sizes in each tool batch is less than its tool size. Constraint (4.31) constrains the sum of tool sizes in each autoclave batch to be less than the autoclave capacity.

[1] An alternative approach is to create tool batches with predefined tools. However, this approach expands the number of possible tool batches from $|\mathcal{J}|$ to $|\mathcal{J}||\mathcal{T}|$.

The standard global *packing* constraint does not allow the size of an item to depend on other decision variables, and therefore cannot be used for two-stage bin packing. Thus, we choose to use interval variables for the packing stage to easily connect them with the corresponding scheduling variables via existing global constraints.

Next, we present the scheduling constraints.

$$\texttt{sizeOf}(x_{j,k}) = p_j \qquad \forall j \in \mathcal{J} \quad \forall k \in \mathcal{B}^1 \tag{4.32}$$

$$\texttt{span}\left(\gamma_k, \{x_{j,k} \mid j \in \mathcal{J}\}\right) \qquad \forall k \in \mathcal{B}^1 \tag{4.33}$$

$$\texttt{sizeOf}(\gamma_k) = \sum_{j \in \mathcal{J}} p_j \times \texttt{presenceOf}(x_{j,k}) \qquad \forall k \in \mathcal{B}^1 \tag{4.34}$$

$$\texttt{sizeOf}(y_{k,i}) = P^2 \qquad \forall k \in \mathcal{B}^1 \quad \forall i \in \mathcal{B}^2 \tag{4.35}$$

$$\texttt{synchronize}(\sigma_i, \{y_{k,i} \mid i \in \mathcal{B}^2\}) \qquad \forall i \in \mathcal{B}^2 \tag{4.36}$$

$$\texttt{alwaysIn}\left(\sum_{k \in \mathcal{B}^1} \texttt{pulse}(\gamma_k, 1), 0, |\mathcal{H}|, 0, C^1\right) \tag{4.37}$$

$$\texttt{alwaysIn}\left(\sum_{i \in \mathcal{B}^2} \texttt{pulse}(\sigma_i, 1), 0, |\mathcal{H}|, 0, C^2\right) \tag{4.38}$$

$$\texttt{endBeforeStart}(\gamma_k, y_{k,i}) \qquad \forall k \in \mathcal{B}^1, \quad \forall i \in \mathcal{B}^2 \tag{4.39}$$

$$l_j \geq \text{endOf}(\sigma_i) - d_j \qquad \forall j \in \mathcal{J} \mid j \text{ assigned to } i \tag{4.40}$$

$$l_j \in \{0, ...|\mathcal{H}|\} \; \forall j \in \mathcal{J}$$

Constraint (4.32) sets the processing time of each job. Constraints (4.33) and (4.34) make sure jobs in a tool batch are scheduled sequentially without overlapping. The actual sequence of jobs in a tool batch does not matter because their processing times are not sequence-dependent, and jobs cannot leave the first stage until the entire tool batch is processed. Thus, a no-overlap constraint is unnecessary. Constraint (4.35) sets the processing time of each tool batch. Constraint (4.36) makes sure all tool batches in the same autoclave batch are processed simultaneously. Constraints (4.37) and (4.38) enforce that the total number of tool batches or autoclave batches being processed at one time is at most the respective stage's capacity. Constraint (4.39) says that each tool batch must finish processing before its associated autoclave batch can begin processing. Constraint (4.40) defines job tardiness.

4.2.3 Logic-based Benders Decomposition

We introduce a three-stage decomposition that separates the 2BPHFSP into a one-stage bin packing problem, a pattern bin packing problem, and a two-stage hybrid flow shop scheduling problem, which will be denoted as the *m-master problem*, the *m-subproblem*, and the *subproblem*, respectively. If we can solve these problems iteratively while adding cuts, we may be able to converge to the optimal or a near-optimal solution within a reasonable time limit. This decomposition is shown in Figure 4.3.

The two loops in Figure 4.3 show the order in which the problems are solved. The m-master problem is a one-stage bin packing problem which packs jobs into capacity-constrained autoclave batches, while

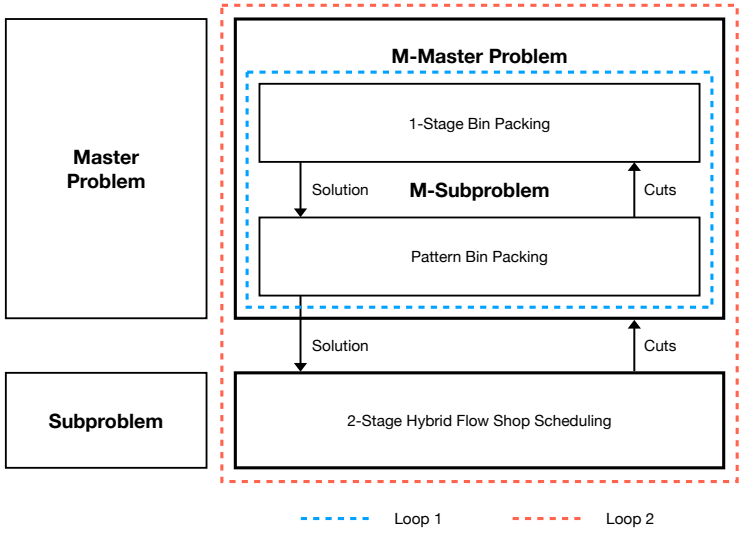


Figure 4.3: Decomposition flow between problems, each iteration of loop 2 finds a feasible solution to the entire problem.

minimizing the number of open autoclave batches. The m-master problem solution is therefore a set of open autoclave batches. We chose to batch jobs into autoclave batches directly because part of the objective is to minimize the number of open autoclave batches, and the job-to-autoclave batching is a relaxation of the two-stage batching requirement.

A feasible packing of jobs in an autoclave batch may not be feasibly partitionable into tool batches as jobs may not fit perfectly on tools. The m-subproblem attempts to find such a feasible partitioning for each autoclave batch. We denote this constrained partitioning problem as the *Pattern Bin Packing Problem*. If we find a feasible packing for all autoclave batches, we schedule the tool and autoclave batches in the subproblem. If no feasible packing exists for an autoclave batch, a feasibility cut is added to the m-master problem.

Once we reach and solve the subproblem, there are two approaches to adding optimality cuts to the master problem. One approach is to merely cut off the incumbent solution and rerun the algorithm to obtain a different feasible solution. The problem with this approach is that we would need to traverse the entire search space to prove optimality, as we are not adding any direct lower bounds on tardiness. Therefore, for the second approach, we will solve for lower bounds on total tardiness by fixing one autoclave batch at a time and solving a relaxed scheduling problem. We add these lower bounds, along with the first approach's optimality cut, to the m-master problem. Lastly, we can add an upper bound on total tardiness in the m-master problem based on total tardiness for each subproblem solution. Next, we formally present the mathematical models for this decomposition.

M-Master Problem. he m-master problem is modelled with CP. It is not common to use CP to model a bin packing master problem, however, in this particular decomposition, we need to use CP in order to capitalize on its global constraints. The feasibility cuts that are added from the m-subproblem to the m-master problem, which will be explained in subsequent pages, take the form of a CP global

Table 4.3: LBBD m-master problem variables.

Variable	Description
x_j	Autoclave batch index that job j is packed into
θ_i	Sum of job sizes in autoclave batch i
q	Number of open autoclave batches

constraint called the *Global Cardinality Constraint*.

The objective is to minimize the number of open autoclave batches. Model variables are shown in Table 4.3; see Table 4.1 for model parameters.

$$\min \quad q \tag{4.41}$$

$$\text{s.t.} \quad \texttt{pack}\left(\{\theta_i \mid i \in \mathcal{B}^2\}, \{x_j \mid j \in \mathcal{J}\}, \{s_j \mid j \in \mathcal{J}\}\right) \tag{4.42}$$

$$q = \texttt{max}\left(\{x_j \mid j \in \mathcal{J}\}\right) \tag{4.43}$$

$$x_j \in \{0, ...|\mathcal{B}^2|\}\ \forall j \in \mathcal{J};\ \theta_i \in \{0, ...V^2\}\ \forall i \in \mathcal{B}^2;\ q \in \{0, ...|\mathcal{B}^2|\}$$

Constraint (4.42) is a global constraint which packs jobs in set $\mathcal{J}$ into autoclave batches in set $\mathcal{B}^2$ while keeping the sum of job sizes in each autoclave batch below the autoclave capacity. Constraint (4.43) defines the number of open batches.

M-Subproblem. The m-subproblem is modelled with MIP and two sets of binary decision variables. The first set creates tool batches from jobs and the second set assigns each tool batch to a tool. The objective function minimizes the total sum of tool sizes.

This MIP formulation is based on models presented by Emde et al. [43] (see Section 3.3.3) and Kosch and Beck [74] (see Section 3.2.1). Consider a tool batch k that contains the set of jobs $\mathcal{J}^*$, let $j^{'}$ be the job with the lowest index in $\mathcal{J}^*$. We denote job $j^{'}$ as the representative job of tool batch k and create decision variables $y_{j,j^{'}}$ that assign jobs to representative jobs instead of directly to tool batches. If job j is assigned to the tool batch with representative job $j^{'}$, then $y_{j,j^{'}}$ is equal to 1, otherwise, $y_{j,j^{'}} = 0$. Therefore, every $y_{j^{'},j^{'}} = 1$ indicates an open tool batch.

Only jobs from the same autoclave batch can be assigned to the same tool batches, so we define a $|\mathcal{J}|$ by $|\mathcal{J}|$ matrix $\mathbf{A}$, where entry $A_{j,j^{'}}$ is equal to 1 if job j and job $j^{'}$ are in the same autoclave batch in the m-master problem solution, and equal to 0 otherwise. Model variables are presented in Table 4.4; see Table 4.1 for model parameters.

Table 4.4: LBBD m-subproblem variables.

Variable	Description
$y_{j,j'}$	Binary variable, equals 1 if job j is in tool batch defined by job $j^{'}$, 0 otherwise
$z_{j',t}$	Binary variable, equals 1 if the tool batch defined by job $j^{'}$ is on tool t
$\tau_{j'}$	Size of tool for the tool batch defined by job $j^{'}$
$\rho_{j'}$	Processing time of tool batch defined by job $j^{'}$
a	Binary variable, equals 1 if there exists an infeasible autoclave batch

$$\min \quad \sum_{j' \in \mathcal{J}} \tau_{j'} \tag{4.44}$$

$$\text{s.t.} \quad \sum_{j' \in \mathcal{J}} y_{j,j'} = 1 \qquad \forall j \in \mathcal{J} \tag{4.45}$$

$$y_{j,j'} \leq A_{j,j'} \qquad \forall j, j^{'} \in \mathcal{J} \tag{4.46}$$

$$y_{j,j'} = 0 \qquad \forall j, j^{'} \in \mathcal{J} : j < j^{'} \tag{4.47}$$

$$y_{j,j'} \leq y_{j',j'} \qquad \forall j, j^{'} \in \mathcal{J} \tag{4.48}$$

$$\sum_{t \in \mathcal{T}} z_{j',t} = y_{j',j'} \qquad \forall j^{'} \in \mathcal{J} \tag{4.49}$$

$$\tau_{j'} = \sum_{t \in \mathcal{T}} v_t z_{j',t} \qquad \forall j^{'} \in \mathcal{J} \tag{4.50}$$

$$\tau_{j'} \geq \sum_{j \in \mathcal{J}} s_j y_{j,j'} - \sum_{j \in \mathcal{J}} s_j (1 - y_{j,j'}) \tag{4.51}$$

$$\rho_{j'} = \sum_{j \in \mathcal{J}} p_j y_{j,j'} \qquad \forall j^{'} \in \mathcal{J} \tag{4.52}$$

$$y_{j,j'} \in \{0,1\} \; \forall j, j^{'} \in \mathcal{J}; \; z_{j',t} \in \{0,1\} \; \forall j^{'} \in \mathcal{J}, \; \forall t \in \mathcal{T}$$

$$\tau_{j'} \in \{0, ...V^1\} \; \forall j^{'} \in \mathcal{J}; \; \rho_{j'} \in \{0, ..., \textstyle\sum_{j \in \mathcal{J}} p_j\} \; \forall j^{'} \in \mathcal{J}$$

The objective function minimizes the sum of tool sizes, which in turn finds the partitioning of jobs within the autoclave batch with the least wasted space (i.e. extra space jobs cannot fill on a tool). Constraint (4.45) makes sure each job is assigned to one tool batch. Constraints (4.46) and (4.47) enforce assignment restrictions. If job $j^{'}$ defines a tool batch, it is forced by Constraint (4.47) to be the lowest indexed job in that tool batch, so Constraint (4.48) tightens the linear relaxation. Each open tool batch is associated with a job $j^{'}$, and only jobs with a higher index than $j^{'}$ can be assigned to the tool batch associated with $j^{'}$. Job j and job $j^{'}$ can only be assigned together if both jobs have been assigned to the same autoclave batch in the m-master problem solution, i.e. if $A_{j,j'} = 1$. Constraint (4.49) makes sure each open tool batch is assigned to a specific tool. Constraints (4.50) and (4.51) define and constrain each tool batch's tool size to be larger than its sum of job sizes. Constraint (4.52) defines the processing time for each tool batch.

Feasibility Cuts. Once the m-subproblem is solved to optimality, if the sum of tool sizes for autoclave batch i^* is larger than the autoclave capacity then i^* is an infeasible batch. Cuts are added to prevent the subset of jobs in each infeasible autoclave batch from being packed together again. Let $\hat{\mathcal{B}^2}$ be the set of infeasible autoclave batches from the incumbent m-master problem solution, and let $\mathcal{J}^{i^*}$ be the

Table 4.5: LBBD subproblem parameters and variables.

Parameter	Description
$k \in \mathcal{B}^{1^*}$	Set of open tool batches
$i \in \mathcal{B}^{2^*}$	Set of open autoclave batches
$(k,i) \in \mathcal{E}^*$	Tool and autoclave batches that are associated with each other obtained from master problem solution
$(j,i) \in \mathcal{F}^*$	Jobs and autoclave batches that are associated with each other obtained from master problem solution
ρ_k^*	Processing time of tool batch m from master problem solution

Variable	Description
γ_k	Interval variable for tool batch k
σ_i	Interval variable for autoclave batch i
l_j	Tardiness of job j

set of jobs assigned to batch $i^* \in \hat{\mathcal{B}^2}$.

These cuts are written as a global cardinality constraint (GCC) for each infeasible autoclave batch. The standard GCC format is $\texttt{gcc}(\{cards\},\{vals\},\{vars\})$ where sets $\{cards\}$ and $\{vals\}$ are the same size. Over the set of variables in $\{vars\}$, each value $val[i]$ should only be taken $cards[i]$ times; $cards[i]$ can be a single value or a range of values. Using GCC cuts also removes symmetrical solutions arising from batch indexing. Constraint (4.53) forms the feasibility cut.

$$\texttt{gcc}\left(\{[0,...,|\mathcal{J}^{i^*}|-1],...\},\{1,...,|\mathcal{B}^2|\},\{x_j \mid j \in \mathcal{J}^{i^*}\}\right) \qquad \forall i^* \in \hat{\mathcal{B}^2} \tag{4.53}$$

Subproblem. The subproblem is modelled using CP and has two sets of interval decision variables, one to schedule tool batches and one to schedule autoclave batches. The objective is to minimize the sum of job tardiness. Model variables and parameters are presented in Table 4.5.

$$\min \quad \sum_{j \in \mathcal{J}} \alpha_j l_j \tag{4.54}$$

$$\text{s.t.} \quad \texttt{sizeOf}(\gamma_k) = \rho_k^* \qquad \forall k \in \mathcal{B}^{1^*} \tag{4.55}$$

$$\texttt{sizeOf}(\sigma_i) = P^2 \qquad \forall i \in \mathcal{B}^{2^*} \tag{4.56}$$

$$\texttt{endBeforeStart}\,(\gamma_k, \sigma_i) \qquad \forall (k,i) \in \mathcal{E}^* \tag{4.57}$$

$$\texttt{alwaysIn}\left(\sum_{k \in \mathcal{B}^1} \texttt{pulse}(\gamma_k, 1), 0, |\mathcal{H}|, 0, C^1\right) \tag{4.58}$$

$$\texttt{alwaysIn}\left(\sum_{i \in \mathcal{B}^2} \texttt{pulse}(\sigma_i, 1), 0, |\mathcal{H}|, 0, C^2\right) \tag{4.59}$$

$$l_j = \texttt{endOf}(\sigma_i) - d_j \qquad \forall (j,i) \in \mathcal{F}^* \tag{4.60}$$

$$l_j \in \{0,...|\mathcal{H}|\}\ \forall j \in \mathcal{J}$$

Constraints (4.55) and (4.56) define the interval variable lengths for tool and autoclave batches, re-

spectively. Constraint (4.57) makes sure any autoclave batch starts after all its associated tool batches have finished processing. Constraints (4.58) and (4.59) enforce resource capacities for each stage. Constraint (4.60) defines job tardiness.

No-good Optimality Cuts: Approach 1. Once the subproblem is solved after a loop 2 cycle, a feasible solution to the entire problem has been found. Thus, to start the cycle again, the incumbent solution must be removed from the search space. Due to the fact that only one autoclave batch needs to be changed to cut off the incumbent solution, we need to add a disjunction to require at least one autoclave batch to be different. GCC constraints cannot be disjoined in the solver we are using [65]. Thus, we introduce a new set of binary variables, $\omega_{i^*} \in \{0,1\}$, where $i^* \in \mathcal{B}^{2*}$ is the set of open autoclave batches.

The binary variable ω_{i^*} is set to be equal to 1 if autoclave batch i^* is required to be different in subsequent solutions, and equal to 0 otherwise. If autoclave batch i^* is forced to be different, then constraints must be added such that only a strict subset of jobs in that batch can be assigned to the same autoclave batch. We add a set of counting constraints that limit how many times each autoclave batch index can be assigned in the set of decision variables belonging to jobs in that batch. Then, a final constraint needs to be added to enforce that at least one binary variable out of the set $\{\omega_{i^*} \mid i^* \in \mathcal{B}^{2*}\}$ is equal to 1, forcing the incumbent solution to change. These constraints remove the incumbent solution as well as any symmetrical solutions that arise from batch indexing. Constraints (4.61) and (4.62) form the *No-good Optimality Cuts*.

$$(\omega_{i^*} == 1) \rightarrow \Big(\texttt{count}(\{x_j \mid j \in \mathcal{J}^{i^*}\}, i) \leq |\mathcal{J}^{i^*}| - 1\Big) \qquad i \in \mathcal{B}^2, \quad i^* \in \mathcal{B}^{2*} \tag{4.61}$$

$$\sum_{i^* \in \mathcal{B}^{2*}} \omega_{i^*} \geq 1 \tag{4.62}$$

$$\omega_{i^*} \in \{0,1\}\ \forall i^* \in \mathcal{B}^{2*}$$

Dual Optimality Cuts: Approach 2. As aforementioned, we have a feasible solution once the subproblem is solved after a loop 2 cycle. If we add the set of tardiness variables, l_j, to the master problem, we can put lower bounds on tardiness based on the master problem packing decisions and a global upper bound. For example, if we are on iteration m, consider a specific subset of jobs in an autoclave batch $\mathcal{J}^{i^*}$, a lower bound on tardiness given that batch decision T_{i^*}, and the best upper bound so far, U. If there are master problem solutions containing the grouping of jobs $\mathcal{J}^{i^*}$ in a batch where the lower bound on tardiness is larger than U because of the contribution from T_{i^*}, then adding that lower bound from iteration m will cut off a portion of the search space. Therefore, being able to find these lower bounds may help us go through fewer iterations of loop 2.

To find a lower bound on tardiness based on batch decisions from the master problem, it is necessary to solve a relaxed version of the subproblem. First, we formally define this problem, which we denote as the relaxed scheduling problem (RSP). The objective is the same as the subproblem, which is a weighted sum of tardiness. One autoclave batch and its associated tool batches are chosen to be fixed from the incumbent master problem solution. However, we no longer consider the existence of any other batches, and instead, each job not in a fixed batch is processed once in the first stage then once in the second

Table 4.6: RSP parameters and variables.

Parameter	Description
$k^* \in \mathcal{B}^{i^*}$	Set of fixed tool batches in autoclave batch i^*
i^*	Fixed autoclave batch i^*
$j \in \mathcal{J}$	Set of all jobs
$j \in \mathcal{J}^+$	Set of jobs that are not in any fixed batches
$(j, k^*) \in \mathcal{G}$	Jobs and fixed tool batches that are associated with each other
$n \in \mathcal{H}$	Set of time points
d_j	Due date of job j
s_j	Size of job j
p_j	Processing time of job j
θ_{i^*}	Sum of job sizes in fixed autoclave batch i^*
V^2	Maximum capacity of autoclave batches
P^2	Processing time of autoclave batches
C^1	Number of stage 1 machines
C^2	Number of stage 2 machines
α_j	Weighting for tardiness of job j in objective function

Variable	Description
γ_j	Interval variable for job j in stage 1
σ_j	Interval variable for job j in stage 2
Γ_{k^*}	Interval variable for fixed tool batch k^* in stage 1
Σ_{i^*}	Interval variable for fixed autoclave batch i^* in stage 2
l_j	Tardiness of job j

stage. We model the first stage as a multi-capacity machine and each single job or fixed tool batch occupies one unit of capacity during processing. The second stage is also a multi-capacity machine, but each single job occupies s_j units of space during processing and the fixed autoclave batch occupies V^2 units of space during processing. The second stage machine has a constant capacity of $C^2 \times V^2$. There are precedence constraints between the stages of each job. The RSP is a relaxation of the 2BPHFSP, so the optimal objective value found by the RSP is a valid lower bound to the 2BPHFSP.[2] Model variables and parameters are shown in Table 4.6.

The CP model for the RSP is presented next.

$$\min \quad \sum_{j \in \mathcal{J}} \alpha_j l_j \tag{4.63}$$

$$\text{s.t.} \quad \texttt{sizeOf}(\gamma_{k^*}) = \sum_{(j,k^*) \in \mathcal{G}} p_j \qquad \forall k^* \in \mathcal{B}^{i^*} \tag{4.64}$$

$$\texttt{sizeOf}(\sigma_{i^*}) = P^2 \tag{4.65}$$

$$\texttt{sizeOf}(\Gamma_j) = p_j \qquad \forall j \in \mathcal{J}^+ \tag{4.66}$$

$$\texttt{sizeOf}(\Sigma_j) = P^2 \qquad \forall j \in \mathcal{J}^+ \tag{4.67}$$

$$\texttt{endBeforeStart}\,(\gamma_{k^*}, \sigma_{i^*}) \qquad \forall k^* \in \mathcal{B}^{i^*} \tag{4.68}$$

$$\texttt{endBeforeStart}\,(\Gamma_j, \Sigma_j) \qquad \forall j \in \mathcal{J}^+ \tag{4.69}$$

[2]Only the optimal objective value found by the RSP is a valid lower bound, so for any instance the RSP is unable to prove optimality, we cannot add any cuts.

$$\texttt{alwaysIn}\left(\sum_{j\in\mathcal{J}^+}\texttt{pulse}(\Gamma_j,1)+\sum_{k^*\in\mathcal{B}^{i^*}}\texttt{pulse}(\gamma_{k^*},1),0,H,0,C^1\right) \tag{4.70}$$

$$\texttt{alwaysIn}\left(\sum_{j\in\mathcal{J}^+}\texttt{pulse}(\Sigma_j,s_j)+\texttt{pulse}(\sigma_{i^*},\theta_{i^*}),0,H,0,V^2\times C^2\right) \tag{4.71}$$

$$l_j \geq \texttt{endOf}(\Sigma_j)-d_j \qquad \forall j\in\mathcal{J}^+ \tag{4.72}$$

$$l_j \geq \texttt{endOf}(\gamma_{k^*})-d_j \qquad \forall (j,k^*)\in\mathcal{G} \tag{4.73}$$

$$l_j \in \{0,...|\mathcal{H}|\}\ \forall j\in\mathcal{J}$$

Constraints (4.64) and (4.65) set the processing times of the fixed batches. Constraints (4.66) and (4.67) set the processing times of jobs not in fixed batches. Constraint (4.68) sets precedence between the fixed tool and autoclave batches. Constraint (4.69) makes sure jobs are processed in the first stage before progressing to the second stage. Constraints (4.70) and (4.71) enforce the capacity of each stage. Constraints (4.72) and (4.73) define job tardiness.

Once we solve the RSP to optimality with a fixed autoclave batch i^* and its associated tool batches, $k\in\mathcal{B}^{i^*}$, the objective value is a lower bound on tardiness, designated as T_{i^*}. We need to add a cut to the master problem that says if the same grouping of jobs in i^* appears again, the tardiness term in the master problem objective must be at least T_{i^*}. The RSP needs to be solved for each active autoclave batch in the incumbent master solution. Let $j\in\mathcal{J}^{i^*}$ be the set of jobs that are in i^*, with decision variables $\{x_j \mid j\in\mathcal{J}^{i^*}\}$. Let m represent the current iteration, and let U^m represent the upper bound from iteration m. Constraints (4.74) and (4.75) form the *Dual Optimality Cuts*.

$$\left(count\left(\{x_j \mid j\in\mathcal{J}^{i^*}\},i\right)==|\mathcal{J}^{i^*}|\right)\rightarrow\left(\sum_{j\in\mathcal{J}}\alpha_j l_j\geq T_{i^*}\right) \qquad \forall i\in\mathcal{B}^2 \tag{4.74}$$

$$\sum_{j\in\mathcal{J}}\alpha_j l_j \leq U^m \tag{4.75}$$

We can now consider two LBBD approaches, where the first LBBD approach, designated as *LBBD1*, follows the loops shown in Figure 4.3 and only optimality cuts from the first approach are added after loop 2 completes. The second LBBD approach, designated as *LBBD2*, also follows the loops in Figure 4.3, however, after loop 2 completes and the approach 1 cuts are added, we enter an auxiliary loop where we repeatedly solve the RSP while fixing a single autoclave batch from the incumbent solution. This modified decomposition flow is shown in Figure 4.4.

The two LBBD models presented in this section will always converge. The number of feasible solutions to the 2BPHFSP is finite; the subproblem cuts remove feasible solutions from the search space and do not allow the master problem to revisit solutions. Therefore, given enough time, both LBBD models will exhaust the search space. However, the unidirectional flow of the LBBD models from the master problem to the subproblem does not allow the model to consider any trade-offs between tool batch partitioning decisions and tardiness. For example, suppose there are several feasible tool batch partitionings for a given m-master problem solution. The m-subproblem will find one of those partitionings and since the packing is now feasible, the model will move onto the subproblem to find the best schedule given the

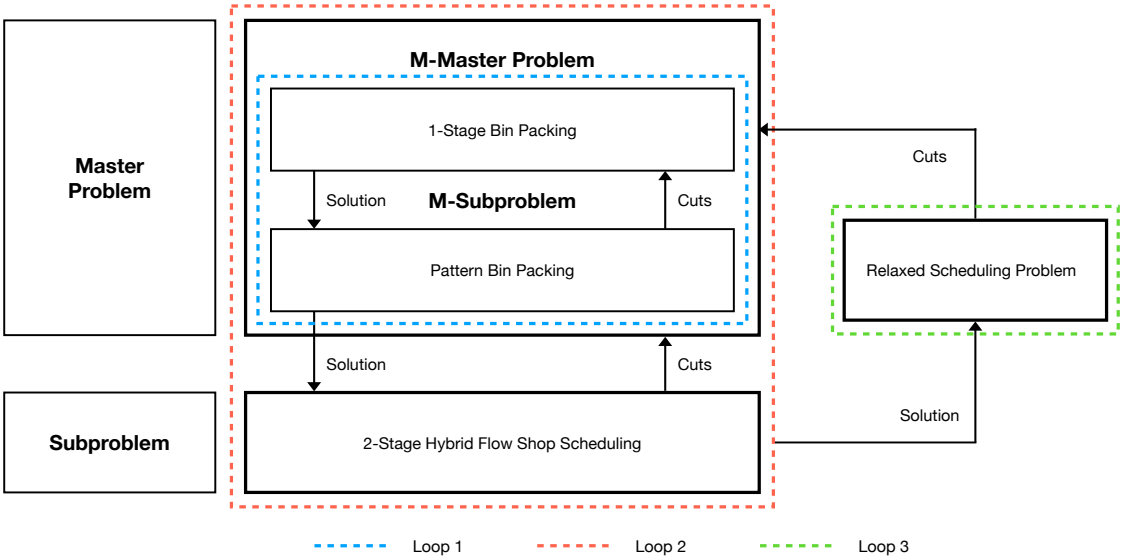


Figure 4.4: Modified decomposition flow between problems to add dual optimality cuts, loop 3 is the auxiliary loop where the RSP is solved for each autoclave batch from the incumbent solution.

incumbent packings. However, it may be the case that another one of the feasible tool batch partitionings will result in a better schedule, but the LBBD models will not backtrack to try all the partitionings. Thus, unlike the monolithic models, the LBBD models do not provide any guarantees on optimality.

Theoretical Results. To close this section, we formally prove the validity of the cuts used by the decomposition approaches. To prove a cut is valid, we must first prove the incumbent solution is removed from the search space when the cut is applied and we must then prove the cut does not remove any globally optimal solutions when applied [30, 120].

First, we added a cut from the m-subproblem into the m-master problem for each infeasible autoclave batch from the m-master problem solution. Lemma 4.2.1 proves that the infeasible solution is removed from the search space if the cut is applied.

Lemma 4.2.1. *Given a solution with an infeasible autoclave batch i^*, Cut* (4.53) *removes i^* from the search space.*

Proof. Cut (4.53) removes i^* from the search space. Suppose we have an infeasible autoclave batch i^* containing the set of jobs $\mathcal{J}^{i^*}$. The decision variables associated with jobs in i^*, $\{x_j \mid j \in \mathcal{J}^{i^*}\}$ take on values ranging from $\{1, ..., |\mathcal{B}^2|\}$, where $|\mathcal{B}^2|$ is the total number of autoclave batches available. If $\{x_j \mid j \in \mathcal{J}^{i^*}\}$ are assigned to the same autoclave batch, then, in any solution where these jobs are grouped into an autoclave batch, one of the values from $\{1, ..., |\mathcal{B}^2|\}$ appears $|\mathcal{J}^{i^*}|$ times. Cut (4.53) explicitly removes all such solutions. □

We also need to prove that the cut does not remove any globally optimal solutions from the search space, this is shown in Lemma 4.2.2.

Lemma 4.2.2. *Cut* (4.53) *does not remove any globally optimally solutions from the search space.*

Proof. Suppose we have a globally optimal solution $\mathcal{S}$ to the 2BPHFSP containing autoclave batch $i^{'}$, which contains the same jobs as infeasible autoclave batch i^*. If we let $\{x_{j'} \mid j \in \mathcal{J}\}$ be the set of decision variables that assign jobs to autoclave batches for solution $\mathcal{S}$, then all variables in the set $\{x_{j'} \mid x_j = i^*\}$ are equal to $i^{'}$. Constraint (4.42) is violated by autoclave batch i^* as $\sum_{j \in \mathcal{J} | x_j = i^*} s_j > V^2$. Similarly, the sum $\sum_{j \in \mathcal{J} | x_j = i'} s_j$ is also greater than the capacity V^2. Therefore, $\mathcal{S}$ is infeasible due to this capacity constraint violation and cannot be a globally optimal solution. □

Lemmas 4.2.1 and 4.2.2 can be combined into a single theorem, as follows.

Theorem 4.2.3. *Given a solution with an infeasible autoclave batch i^*, Cut* (4.53) *is sufficient to remove any solution containing that grouping of jobs from the search space.*

Proof. Follows directly from Lemmas 4.2.1 and 4.2.2. □

Next, let us examine the optimality cuts added from the subproblem to the master problem. Proposition 4.2.1 shows the incumbent solution will be removed from the solution space if the optimality cuts are applied.

Proposition 4.2.1. *Given a feasible solution, cuts* (4.61) *and* (4.62) *are sufficient to remove the incumbent solution from the search space.*

Proof. Suppose the incumbent solution has $|\mathcal{B}^{2*}|$ open autoclave batches, where each autoclave batch $i \in \{1, ..., |\mathcal{B}^{2*}|\}$ is associated with binary variable ω_i. If $\omega_{i^*} = 1$, autoclave batch i^* cannot appear in any subsequent solutions. If the sum of ω_i over all $i \in \mathcal{B}^{2*}$ is equal to 0, meaning $\omega_i = 0 \; \forall i \in \mathcal{B}^{2*}$, all of the disjunctions from Constraint (4.61) are inactive so no constraints are imposed on the set of decision variables $\{x_j \mid j \in \mathcal{J}\}$. Now, suppose the sum of ω_i over all $i \in \mathcal{B}^{2*}$ is equal to 1 because $\omega_{i*} = 1$. Then, Constraint (4.61) adds the following constraints to the master problem.

$$\texttt{count}(\{x_j \mid j \in \mathcal{J}^{i^*}\}, i) \leq |\mathcal{J}^{i^*}| - 1 \quad \forall i \in \mathcal{B}^2, \tag{4.76}$$

where $\mathcal{J}^{i^*}$ is the set of jobs in autoclave batch i^* for which $\omega_{i^*} = 1$. If $\mathcal{J}^{i^*}$ were to appear again in any autoclave batch of subsequent solutions, Constraint (4.76) would be violated as $\texttt{count}(\{x_j \mid j \in \mathcal{J}^{i^*}\}, i) = |\mathcal{J}^{i^*}| > |\mathcal{J}^{i^*}| - 1$. Thus, no solution containing autoclave batch i^* can appear again in subsequent solutions. Note that it is possible that exactly one autoclave batch from the incumbent solution can be changed to form a new distinct solution, as shown by example in Figure 4.5. Therefore, if we forbid at least one autoclave batch from appearing again using cuts (4.61) and (4.62), we will successfully remove the incumbent solution from the search space. □

The last set of cuts we presented in this section are the dual optimality cuts added by the subproblem to the master problem after solving the RSP. The RSP is a relaxation of the 2BPHFSP, thus, if we can solve the RSP to optimality, that optimal objective is a valid lower bound to the 2BPHFSP and can be added as optimality cuts to the master problem.

4.2.4 Earliest Due Date Heuristic

We developed a greedy heuristic based on the Earliest Due Date priority rule, designated as *EDD*, that finds feasible solutions in polynomial time. EDD is described by Algorithm 4 and involves assigning jobs to tool batches, assigning tool batches to autoclave batches, and lastly, scheduling both tool and

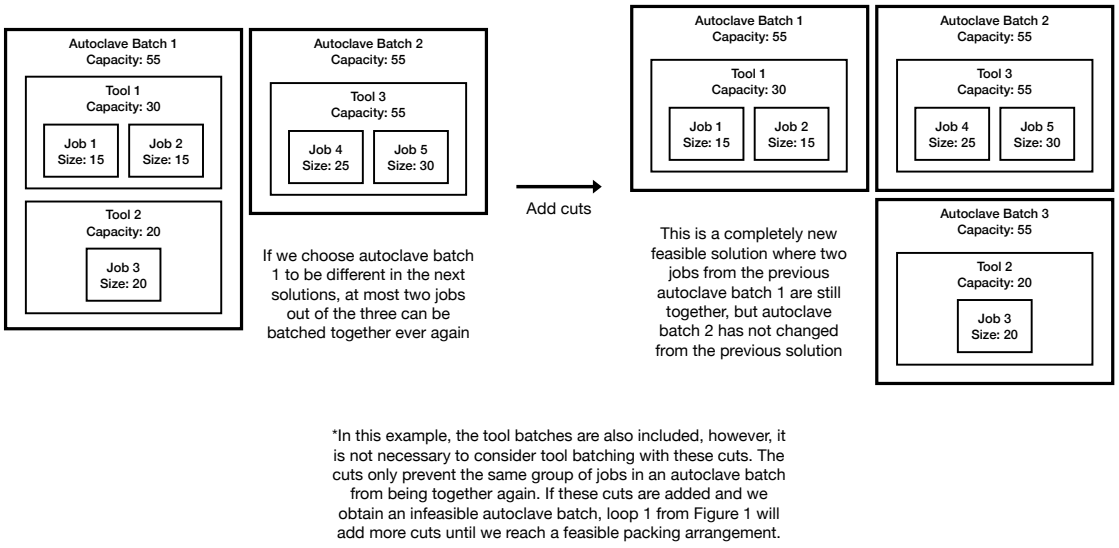


Figure 4.5: Example showing exactly one autoclave batch from the incumbent solution changing to give a new solution.

autoclave batches. For the two assignment portions of the algorithm, items (jobs or tool batches) are sorted in increasing order of due date then packed into bins (tool batches or autoclave batches) based on a first-fit decreasing (FFD) algorithm. The scheduling portion iterates through the list of packed autoclave batches in increasing order of due date and schedules the autoclave batch and its associated tool batches on the earliest available machines, while maintaining precedence between stages. Parameters used by the algorithm are presented in Table 4.7.

For both assignment portions, we iterate through a list of jobs or tool batches sorted by increasing due date. In each iteration, we remove a job or tool batch from its list and pack it into a tool batch or autoclave batch. To potentially find better solutions, instead of picking the first item from the list, we randomly selected an item from the first L items of the list. Then, we repeatedly run the algorithm for a number of iterations and pick the best solution at the end. By perturbing the selection step slightly, we will hopefully find better solutions compared to only selecting the first item every time.

Algorithm 4: EDD

Result: Feasible solution to the 2BPHFSP
for *each repetition* **do**
  sort $\mathcal{J}$ by order of increasing due date;
  while $\mathcal{J}$ *not empty* **do**
    $j \leftarrow$ randomly selected job from first L items of $\mathcal{J}$;
    if $\mathcal{B}^1$ *empty or* j *does not fit in any open batch* **then**
      create new tool batch k and add to $\mathcal{B}^1$, assign j to k;
    else
      assign j to first $k \in \mathcal{B}^1$ with enough space;
    end
  end
  sort $\mathcal{B}^1$ in order of increasing due date;
  while $\mathcal{B}^1$ *not empty* **do**
    $k \leftarrow$ randomly selected tool batch from first L items of $\mathcal{B}^1$;
    if $\mathcal{B}^2$ *empty or* k *does not fit in any open batch* **then**
      create new autoclave batch object i and add to $\mathcal{B}^2$, assign k to i;
    else
      assign k to first $i \in \mathcal{B}^2$ with enough space;
    end
  end
  sort $\mathcal{B}^2$ in order of increasing due date;
  while $\mathcal{B}^2$ *not empty* **do**
    $i \leftarrow$ randomly selected autoclave batch from first L items of $\mathcal{B}^2$;
    sort tool batches in i in order of decreasing processing time;
    for $k \in$ *sorted tool batches of* i **do**
      $m_1 \leftarrow$ `GetNextFreeMachine(`$\mathcal{M}^1$`)`;
      schedule k next on m_1;
    end
    $m_2 \leftarrow$ `GetNextFreeMachine(`$\mathcal{M}^2$`)`;
    schedule i next on m_2 after end of all k in i;
  end
  add solution to solution list;
end
pick best solution from solution list;

Table 4.7: EDD parameters.

Parameter	Description
$j \in \mathcal{J}$	Set of jobs
$k \in \mathcal{B}^1$	Set of tool batches
$i \in \mathcal{B}^2$	Set of autoclave batches
$t \in \mathcal{T}$	Set of tools
s_j	Size of job j
p_j	Processing time of job j
d_j	Due date of job j
v_t	Size of tool t
V^2	Maximum capacity of autoclave batches
P^2	Processing time of autoclave batches
C^1	Number of stage 1 machines
C^2	Number of stage 2 machines
R	Replications
L	List randomization length

4.3 Numerical Results

Our five approaches were tested on 11 sets of 30 randomly generated problem instances, ranging from 5 jobs to 100 jobs per instance. Job parameters were sampled from the following probability distributions.

- $d_j \sim \texttt{round}(\texttt{Unif}(10, \frac{\alpha * 20}{2}))$
- $p_j \sim \texttt{round}(\texttt{Unif}(5, 20))$
- $s_j \sim 50 + \texttt{round}(\texttt{Exp}(0.1) \times 30) \times 10 \mid s_j <= 300$

where α is equal to the number of jobs per instance.

The same system parameters were used across all instances: stage 1 has 4 machines, stage 2 has 2 machines, an autoclave in stage 2 has capacity 400, the curing time for an autoclave batch is 60 minutes, and there are 10 tools, each with their own size parameter ranging from 150 to 300. All CP and MIP models were implemented in Java using CPLEX Optimization Studio 12.9. Each approach was given a 60-minute time limit to solve an instance.

Time Limits on Decomposition Components. First, we tested the difference between enforcing and not enforcing a 10-minute time limit on the individual components of the decomposition models, i.e. the m-master problem, m-subproblem, and subproblem search processes are all terminated after 10 minutes if optimality is not proven yet, and the best feasible solution so far is selected. Figure 4.6 compares the number of instances for which each model was able to find a feasible solution. Figure 4.7 shows the percent difference in the best objective value the time-limited model was able to find compared to its respective non-time-limited model. Negative values indicate instances where the time-limited models gave better, i.e. lower, objective values. Figure 4.7 also shows the percent increase in the number of loop 2 iterations the time-limited models was able to complete in one hour.

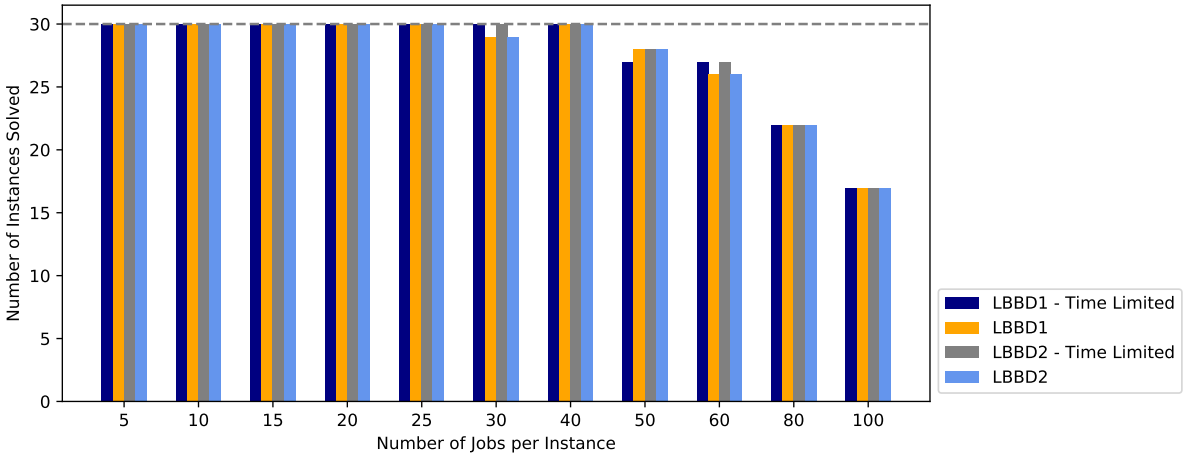


Figure 4.6: Number of instances solved using LBBD1 and LBBD2 with time-limited and non-time-limited components.

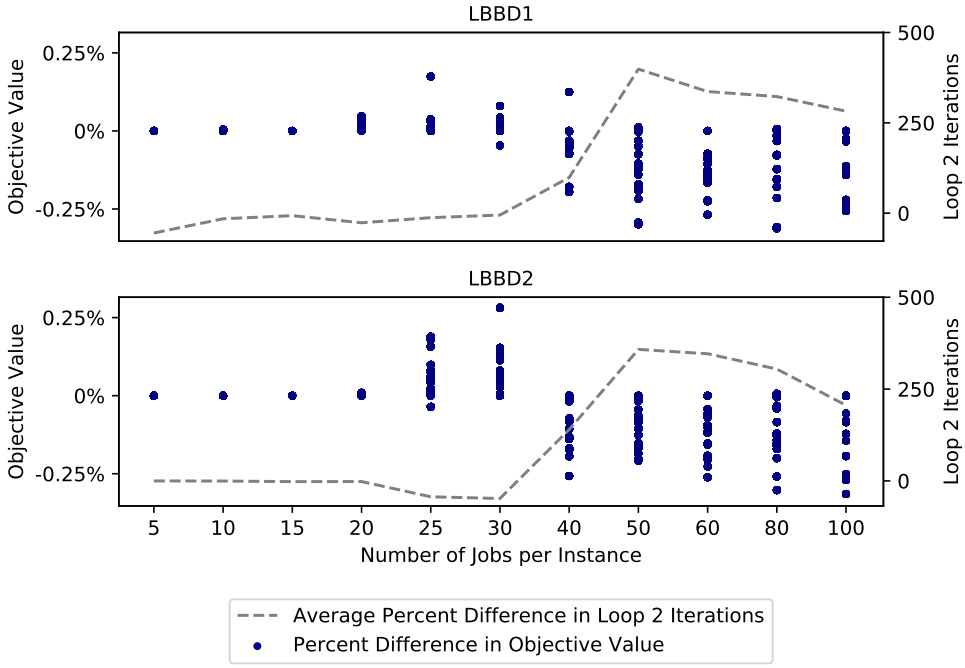


Figure 4.7: Percent differences in objective value and number of completed loop 2 iterations (i.e. number of feasible solutions found), compared to non-limited models.

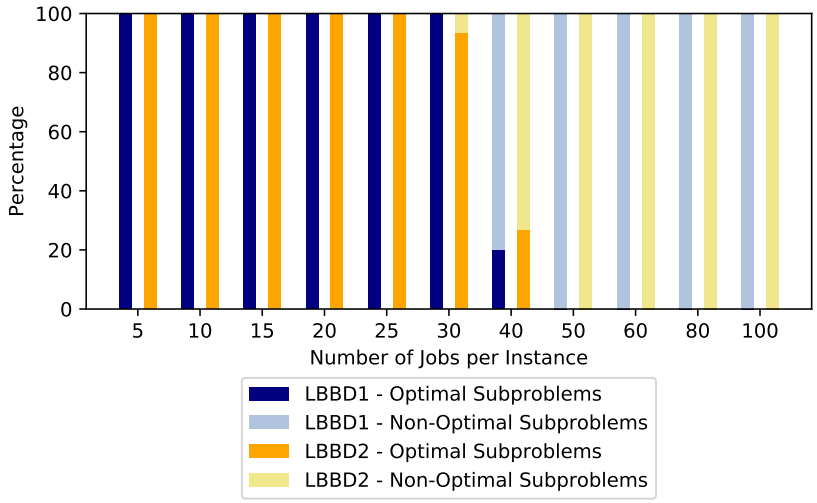


Figure 4.8: Percentage of instances where all components were solved to optimality compared to the percentage of instances where they were not all solved to optimality.

There is almost no difference in the number of instances solved between the time-limited model and the non-limited model. These results also show that LBBD1 and LBBD2 have extremely similar performance, which will be further discussed in later paragraphs. The time-limited models were able to find slightly better solutions for most instance sizes as shown in Figure 4.7. However, this is an insignificant variation in objective value, with less than 0.3% increase or decrease across all instances. If we look at instances with 40 to 80 jobs per instance, the time-limited models complete more loop 2 iterations compared to the non-limited models. Within this instance size range, the non-limited models are only able to complete one or two loop 2 iteration for many instances. However, the time-limited models can complete up to six loop 2 iterations and provide a bigger pool of solutions to choose from.

We can take a closer look at the transition point where placing a time limit on components starts to affect solution quality. Figure 4.8 shows the percentage of instances where a component was not solved to optimality for LBBD1 and LBBD2. We can clearly see that the transition point occurs at 40 jobs per instance. The subproblem is the bottleneck and it becomes difficult to prove optimality within 10 minutes starting at 40 jobs per instance. Due to the slight increase in solution quality when a time limit is added, as well as the propensity of the algorithm to time out globally within the subproblem, it is necessary to impose time limits on the components. The rest of this section will only consider the time-limited models.

Overall Comparison of Solution Approaches. CP, MIP, and EDD were tested on the same set of instances as LBBD1 and LBBD2. MIP could not find feasible solutions for any instance, therefore, MIP will no longer be considered as a solution approach. Figure 4.9 compares the number of instances solved using each approach. CP is clearly the worst performing model, scaling poorly compared to the decomposition models. EDD found feasible solutions for all instances.

Recall the RSP was defined and solved as part of LBBD2, providing a lower bound on tardiness given a single fixed autoclave packing. We can also find a global lower bound using the RSP with no fixed

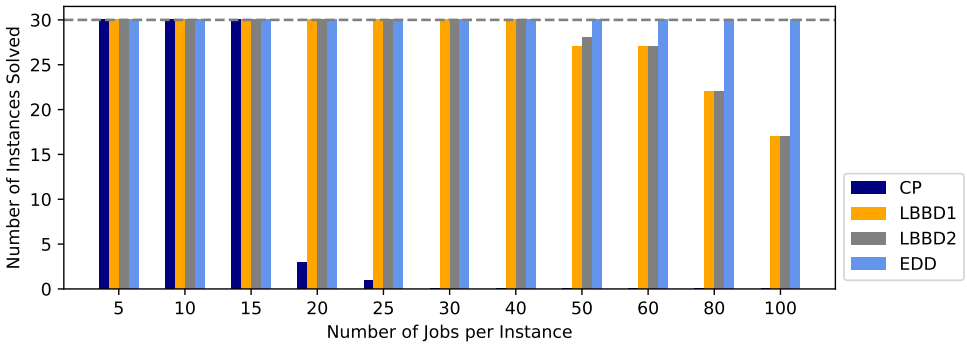


Figure 4.9: Number of instances solved using each approach.

Table 4.8: Average optimality gap for solutions found using each approach.

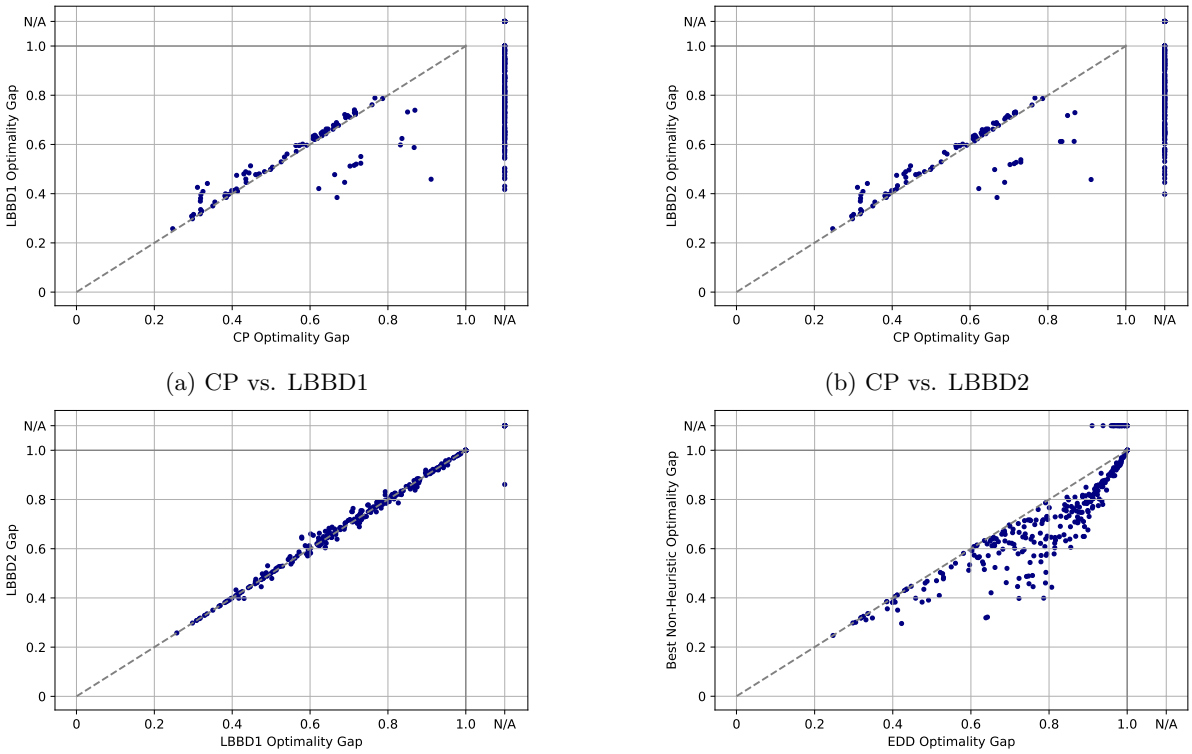
Jobs per Instance	CP	LBBD1	LBBD2	EDD
5	0.50	0.52	0.52	0.49
10	0.48	0.49	0.49	0.53
15	0.66	0.59	0.60	0.68
20	0.85	0.59	0.59	0.74
25	0.87	0.69	0.70	0.82
30	N/A	0.73	0.74	0.85
40	N/A	0.82	0.81	0.91
50	N/A	0.87	0.87	0.94
60	N/A	0.90	0.90	0.96
80	N/A	0.95	0.95	0.98
100	N/A	0.96	0.96	0.98

packings; every job is scheduled once in the first stage then once in the second stage. The optimality gap for a solution can be calculated by Equation (4.77).

$$\text{Optimality Gap} = \frac{\text{Objective Value} - \text{Optimal Objective Value of the RSP}}{\text{Optimal Objective Value of the RSP}} \tag{4.77}$$

Table 4.8 shows the average optimality gaps for each solution approach. There are a few instances for which the RSP was not able to prove optimality within one hour. These instances are not included in any average optimality gap calculations. Furthermore, only instances where the particular solution approach was able to find a solution was included in the average for that approach.

Figure 4.10 visualizes the optimality gaps of each solution approach. We can see from Figures 4.10a and 4.10b that CP is more often than not finding worse quality solutions or no solution at all compared to LBBD1 and LBBD2. Figure 4.10c further exemplifies that the two decomposition models have almost identical performance. The optimality gaps of LBBD1 and LBBD2 are very similar, implying that there is no clear benefit being provided by the dual optimality cuts. Figure 4.10d compares the EDD optimality gap against the best, i.e. lowest, non-heuristic optimality gap. Surprisingly, EDD performs well in comparison to non-heuristic methods, despite the fact that EDD is a very simple heuristic. Also, EDD found solutions within seconds for each instance, solving many instances the non-heuristic methods could not.



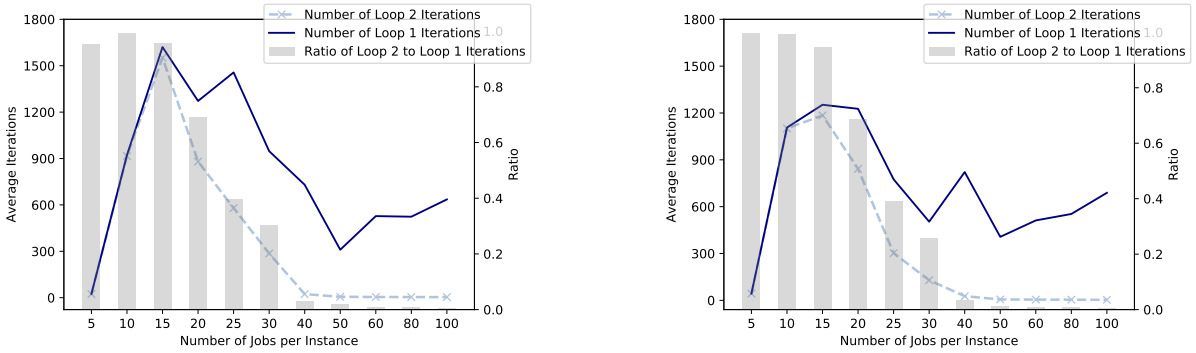
(a) CP vs. LBBD1

(b) CP vs. LBBD2

(c) LBBD1 vs. LBBD2

(d) EDD vs. Best Non-Heuristic Method

Figure 4.10: Comparison of optimality gaps.



(a) LBBD1

(b) LBBD2

Figure 4.11: Number of loop 1 and loop 2 iterations and the ratio of loop 2 to loop 1 iterations for LBBD1 and LBBD2.

Comparing LBBD1 and LBBD2. As aforementioned, the dual optimality cuts do not appear to have a clear benefit on performance. The purpose of the dual optimality cuts is to provide more information about lower bound in return for spending time solving the RSP. We can take a closer look at this trade-off by visualizing the number of loop 1 and loop 2 iterations performed by LBBD1 and LBBD2 as shown in Figure 4.11. LBBD1 and LBBD2 follow a similar pattern through the increasing instance sizes. However, LBBD2 goes through notably fewer loop 1 and loop 2 iterations before 40 jobs per instance. This result is most likely due to the time taken up by solving the RSP. The optimality gaps for LBBD1 and LBBD2 have insignificant differences before 40 jobs per instances, so we can conclude that the improvement gained by receiving lower bounds from the RSP is nullified by the significantly lower number of feasible solutions found compared to LBBD1.

After 40 jobs per instance, the number of loop 1 and loop 2 iterations are almost the same for both models. This result is most likely due to the fact that any improvement or loss from solving the RSP is overshadowed by the fact that CP is no longer able to prove optimality for the subproblem within 10 minutes. Figure 4.12 plots the normalized solution quality over time for both decompositions to further support the previous statement. Each point represents a feasible solution for one instance. The normalized solution quality with respect to the best solution for the given instance, also referred to as the relative gap, of each feasible solution is calculated by Equation (4.78).

$$\text{Relative Gap} = \frac{\text{Feasible Solution Objective Value} - \text{Best Solution Objective Value}}{\text{Best Solution Objective Value}} \tag{4.78}$$

As previously discussed, 40 jobs per instance is the transition point where the subproblem cannot be solved to optimality within 10 minutes. After 40 jobs per instance, both decompositions show a pattern of having solutions clustered around 600, 1200, 1800, 2400, and 3000 seconds. The master problem components can still solve within seconds, but the subproblem is given a time limit of 10 minutes, or 600 seconds. For each loop 2 iteration, there is only one set of dual optimality cuts added. When the model is only able to complete an average of six loop 2 iterations, the meagre number of dual optimality cuts provides negligible improvement in the search process. On the other hand, the large amount of time spent on solving the subproblem overshadows the amount of time spent on solving the RSP, creating a net zero effect of the dual optimality cuts. Thus, for the sake of simplicity, dual optimality cuts should

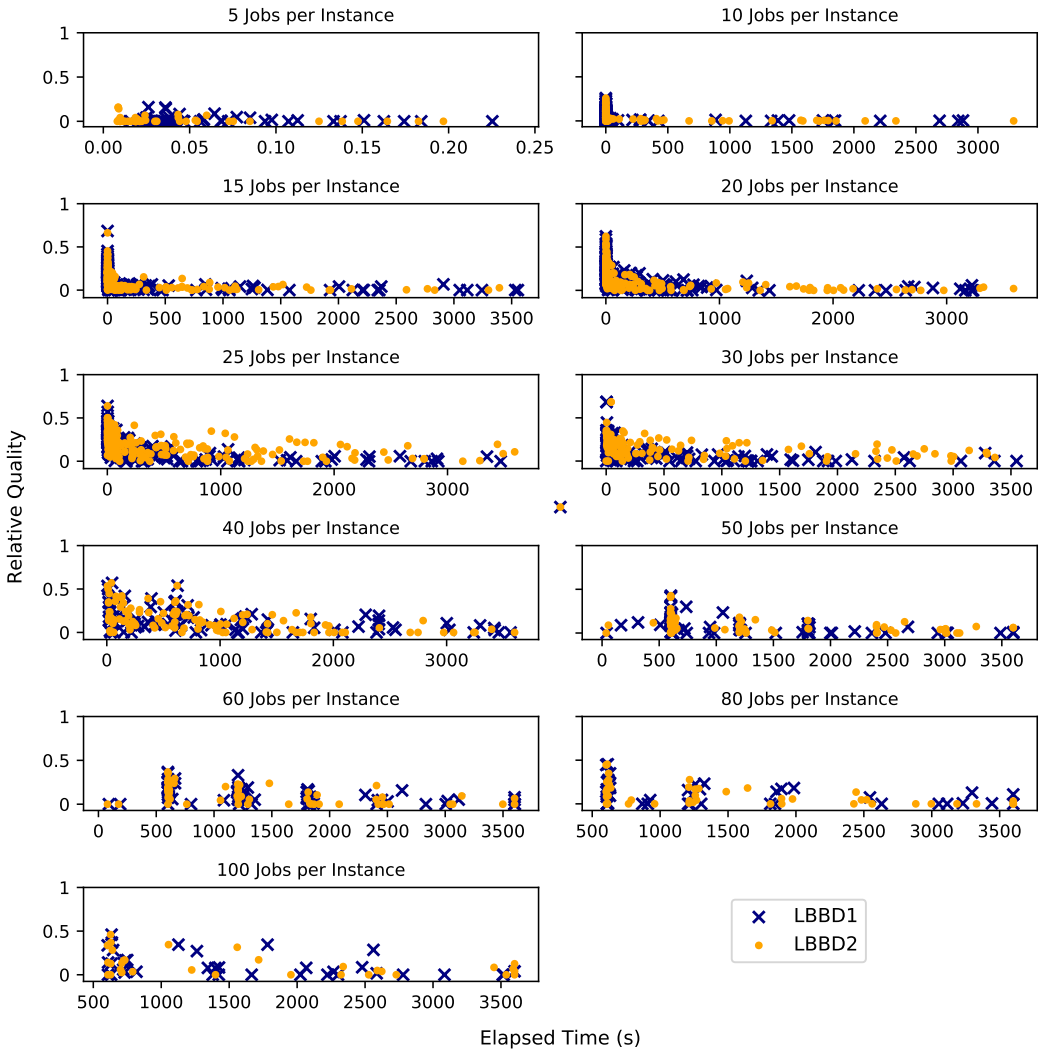


Figure 4.12: Normalized solution quality over time.

not be used when the decomposition structure developed in this chapter is applied to the full problem.

Figure 4.11 also shows the ratio of loop 2 to loop 1 iterations. LBBD1 and LBBD2 follow the same pattern through the increasing instance sizes. When problem instances are small, both models are able to find a feasible packing within loop 1 after one or two iterations. Thus, there is also a 1 to 1 ratio of the two loops. However, as instance sizes grow, the algorithm becomes more and more likely to keep iterating within loop 1 without finding a feasible packing, as evidenced by the ratio of loop 2 to loop 1 iterations following an exponential decrease. In fact, all of the instances with 40 or more jobs where either LBBD model was not able to find a feasible solution for were caused by the algorithm never being able to exit loop 1. We can conclude that as the problem size increases, the number of possible autoclave packings increases exponentially, and only a few infeasible autoclave batches are being cut from the solution space at each loop 1 iteration by the m-subproblem. This conclusion implies that this decomposition structure cannot find solutions efficiently for instance sizes larger than 100 jobs.

Table 4.9: Contingency tables summarizing counts for categories in which LBBD1 or LBBD2 found higher quality solutions.

Model	Low Size		High Size	
	Low Time	High Time	Low Time	High Time
LBBD1	56	33	2	2
LBBD2	66	56	5	3

Model	Low Time		High Time	
	Low Date	High Date	Low Date	High Date
LBBD1	27	26	19	21
LBBD2	43	33	25	29

Model	Low Size		High Size	
	Low Date	High Date	Low Date	High Date
LBBD1	53	36	2	2
LBBD2	68	53	5	5

4.3.1 Statistical Analysis

Performance Patterns from LBBD1 and LBBD2. We previously presented evidence that the dual optimality cuts do not provide any benefits to overall performance. However, previous results do not show any affinities either decomposition model may have towards certain types of problem instances. For each instance tested, we can classify it according to three factors, job size, job processing time, and job due date. For each factor, an instance can be classified as low (i.e. there exists more jobs in the instance with below average job size, processing time, or due date) or high (i.e. there exists more jobs in the instance with above average job size, processing time, or due date). Then, we can count, for each two-factor combination of classifications, how many instances LBBD1 or LBBD2 found better solutions for. Table 4.9 shows the contingency tables summarizing these counts.

χ^2 tests can be performed on the information in the contingency tables to determine if there are any significant interaction effects between factors. Each contingency table in Table 4.9 contains three factors: model type and two from job size, processing time, and due date. We perform a three-way χ^2 analysis for each table and test for complete independence between all three factors, as well as conditional independence for each pair of factors. The null hypothesis in each test is that the factors being tested are fully independent. Test results are summarized in Table 4.10. Each test for independence results in failure to reject the null hypothesis, meaning neither decomposition model has a particular affinity to solving particular types of instances. This analysis shows that the dual optimality cuts do not provide any benefits overall, but we also cannot predict situations where the cuts may be useful.

Effect of Instance Characteristics on Performance. The problem instances used so far in experiments were randomly generated. Thus, we cannot make any statements about how any of the tested approaches may perform if given hard instances. Thus, we created 12 classes of 25-job instances that contained different proportions of jobs with special parameters: high size, high processing time, and early due date. Each class has a different frequency of special jobs; class characteristics are shown in

Table 4.10: χ^2 test results.

Statistic	Complete Independence	Conditional Independence		
		Model	Size	Processing Time
χ^2	2.17	2.17	0.83	1.81
Degrees of Freedom	4	3	3	3
Critical χ^2	7.81	9.49	9.49	9.49
Conclusion	FTR	FTR	FTR	FTR
Statistic	Complete Independence	Conditional Independence		
		Model	Processing Time	Due Date
χ^2	1.67	0.46	1.62	1.49
Degrees of Freedom	4	3	3	3
Critical χ^2	7.81	9.49	9.49	9.49
Conclusion	FTR	FTR	FTR	FTR
Statistic	Complete Independence	Conditional Independence		
		Model	Size	Due Date
χ^2	1.60	1.27	1.31	0.55
Degrees of Freedom	4	3	3	3
Critical χ^2	7.81	9.49	9.49	9.49
Conclusion	FTR	FTR	FTR	FTR

Table 4.11.[3]

LBBD1 and EDD showed the most promise of all the approaches. Thus, we tested the newly generated instances with LBBD1 and EDD. The approaches performed similarly in terms of finding feasible solutions with the exception of class b2. LBBD1 was not able to find solutions for 40% of instances because it could not find a feasible packing. Only 50% of jobs in the instance have large sizes, so the m-master problem opens too few autoclave batches. Then, it becomes difficult to partition jobs onto tools in the m-subproblem such that the sum of tool sizes is within the autoclave capacity.

To test for interaction effects, we perform three three-way analysis of variance (ANOVA) tests on the objective values found using each model type on each instance class. Each test involves ignoring one aspect in the class descriptions, e.g. the first ANOVA test involves ignoring job sizes and only considering instances from class descriptions that contain either high processing times or early due dates. ANOVA results are summarized in Table 4.12, 4.13, and 4.14. The null hypothesis for an ANOVA test is that the mean is the same for all groups. Rejecting this null hypothesis means that the factor level significantly affects the group mean. Tests for which the null hypothesis is rejected are highlighted.

A clear conclusion from the ANOVA tests is that the solution approach, LBBD1 vs. EDD, contributes the least to variance in group means. This further shows that EDD is performing very well for a simple heuristic. The highest contribution to variance in all three ANOVA tests comes from the frequency of special jobs. Job size also contributes more to variance compared to processing time and due date, as

[3]Each category within the description of special jobs has two levels; the frequency of special jobs has three levels.

Table 4.11: Class characteristics of newly generated instances.

Class Name	Description of Special Jobs			Frequency of Special Jobs
	Size	Processing Time	Due Date	
a1	High	High	Normal	Low
a2	High	High	Normal	Medium
a3	High	High	Normal	High
b1	Normal	High	Early	Low
b2	Normal	High	Early	Medium
b3	Normal	High	Early	High
c1	High	Normal	Early	Low
c2	High	Normal	Early	Medium
c3	High	Normal	Early	High
d1	High	High	Early	Low
d2	High	High	Early	Medium
d3	High	High	Early	High

Table 4.12: ANOVA test results, ignoring effect of job sizes.

Factor	Degrees of Freedom	F-Value	PR(>F)
Single Factor Independence: Description of Special Jobs			
desc	2	0.67	0.52
Single Factor Independence: Frequency of Special Jobs			
freq	2	422.26	1.49e-44
Single Factor Independence: Solver Type			
solver	2	2.18	0.14
Two Factor Independence: Description + Frequency of Special Jobs			
desc	2	9.74	1.68e-04
freq	2	529.43	4.16e-46
desc:freq	4	1.93	0.11
Two Factor Independence: Description of Special Jobs + Solver Type			
desc	2	0.69	0.50
solver	1	2.18	0.14
desc:solver	2	0.0043	0.96
Two Factor Independence: Frequency of Special Jobs + Solver Type			
freq	2	609.79	1.48e-49
solver	1	37.28	3.37e-08
freq:solver	2	1.33	0.27
All Factor Independence			
desc	2	17.84	5.69e-07
freq	2	901.00	3.58e-50
solver	1	57.30	1.22e-10
desc:freq	4	3.47	1.21e-02
desc:solver	2	0.36	0.70
freq:solver	2	2.58	8.30e-02
desc:freq:solver	4	0.067	0.99

Table 4.13: ANOVA test results, ignoring effect of job processing times.

Factor	Degrees of Freedom	F-Value	PR(>F)
Single Factor Independence: Description of Special Jobs			
desc	2	11.63	3.50e-05
Single Factor Independence: Frequency of Special Jobs			
freq	2	56.83	2.46e-16
Single Factor Independence: Solver Type			
solver	2	2.67	0.11
Two Factor Independence: Description + Frequency of Special Jobs			
desc	2	124,15	5.77e-25
freq	2	332.61	6.58e-39
desc:freq	4	42.08	9.25e-19
Two Factor Independence: Description of Special Jobs + Solver Type			
desc	2	11.46	4.1e-05
solver	1	2.93	9.08e-02
desc:solver	2	0.024	0.98
Two Factor Independence: Frequency of Special Jobs + Solver Type			
freq	2	59.98	1.04e-16
solver	1	6.87	1.05e-02
freq:solver	2	0.19	0.83
All Factor Independence			
desc	2	218.35	1.43e-30
freq	2	600.08	2.34e-44
solver	1	64.69	1.67e-11
desc:freq	4	76.54	1.24e-24
desc:solver	2	0.97	0.38
freq:solver	2	1.35	2.65
desc:freq:solver	4	0.49	7.43

Table 4.14: ANOVA test results, ignoring effect of job due dates.

Factor	Degrees of Freedom	F-Value	PR(>F)
Single Factor Independence: Description of Special Jobs			
desc	2	12.45	1.80e-05
Single Factor Independence: Frequency of Special Jobs			
freq	2	59.79	5.31e-17
Single Factor Independence: Solver Type			
solver	2	2.25	0.14
Two Factor Independence: Description + Frequency of Special Jobs			
desc	2	183.15	1.38e-30
freq	2	476.41	3.65e-45
desc:freq	4	60.18	2.35e-23
Two Factor Independence: Description of Special Jobs + Solver Type			
desc	2	12.33	2.1e-05
solver	1	2.63	0.11
desc:solver	2	0.021	0.98
Two Factor Independence: Frequency of Special Jobs + Solver Type			
freq	2	61.79	3.68e-17
solver	1	5.48	2.17e-02
freq:solver	2	0.15	0.86
All Factor Independence			
desc	2	365.85	4.06e-38
freq	2	960.78	3.90e-52
solver	1	82.03	1.84e-13
desc:freq	4	121.73	5.26e-31
desc:solver	2	0.99	0.38
freq:solver	2	1.87	0.16
desc:freq:solver	4	0.63	0.64

evidenced by the failure to reject conclusion in testing for independence in the description of special jobs. Overall, there are two important conclusions from testing these new problem instances and performing ANOVA: LBBD1 and EDD have fairly consistent performance with different instance classes, and EDD has strong performance for a simple heuristic compared to a complex decomposition model.

4.3.2 Heuristic-CP Hybrid Techniques

The strong performance of EDD leads to the obvious next step of combining EDD with a CP model to try and make improvements given a feasible solution. We hereby present two heuristic-CP hybrid techniques which combine EDD with one of the models from the previous section. First, we can warm-start CP using the EDD solution, denoted as *WCP*. However, this approach may face the same scaling challenges as CP did. Therefore, we can also take the EDD solution, fix the packings, and use the CP scheduling model from the decomposition subproblem to try and improve the schedule, denoted as *EDD-CP*. It may be the case that only trying to improve the schedule will be enough to obtain good enough solutions.

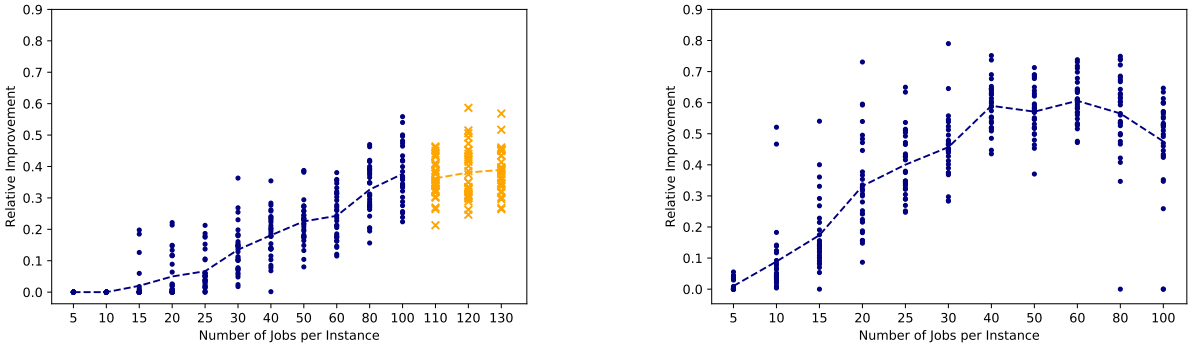
WCP and EDD-CP were tested on the same set of instances as in the original numerical results and given a one hour time limit. Both methods found feasible solutions for each given instance. We can calculate how much improvement CP was able to provide in the two hybrid approaches by using Equation (4.79), where $z(\text{EDD})$ is the EDD solution objective value and $z(\text{CP})$ is the improved objective value.

$$\text{Relative Decrease} = \frac{z(\text{EDD}) - z(\text{CP})}{z(\text{EDD})} \tag{4.79}$$

Figure 4.13 plots the relative gain in solution quality (i.e. decrease in objective value) from both methods. WCP can make large improvements to the EDD solution up to 60 jobs per instance, where its performance starts to degrade, whereas EDD-CP shows gradually improving performance from 5 to 100 jobs per instance. We generated larger problem instances of size 110, 120, and 130 to further test WCP and EDD-CP. Figure 4.13a shows that EDD-CP's performance plateaus after 100 jobs per instance. WCP was not able to solve any problems larger than 100 jobs per instance due to high memory usage. Therefore, EDD-CP is clearly the more robust and promising approach to solving the full problem.

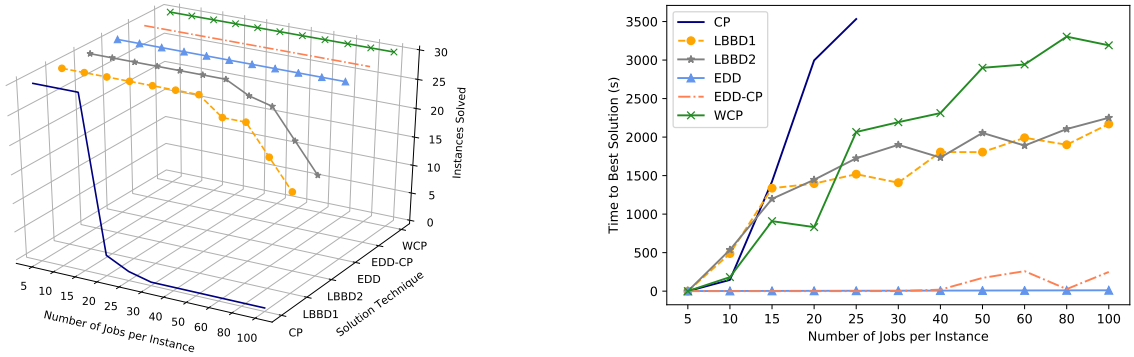
4.3.3 Summary

We can summarize the overall performance of CP, LBBD1, LBBD2, EDD, WCP, and EDD-CP in Figures 4.14 and 4.15. Figure 4.14a summarizes how many instances each solution technique was able to solve over the 11 instance sets, and Figure 4.14b summarizes the average time each solution technique spends to find the best solution within one hour. Figure 4.15 shows the average average optimality gap achieved within one hour for each solution technique, where the optimality gap is calculated using Equation (4.77).



(a) EDD-CP (b) WCP

Figure 4.13: Relative decrease in objective value after using CP to improve the heuristic solution; average decrease is shown by the dotted line. The blue data points show results from larger instances that WCP was not able to solve.



(a) Number of instances solved (b) Average time to best solution

Figure 4.14: Number of instances solved and average time to find the best solution within one hour for all tested approaches.

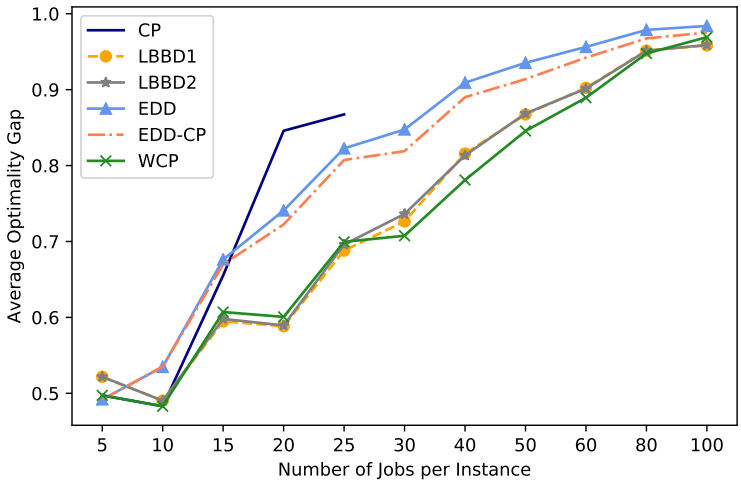


Figure 4.15: Average optimality gap for all tested approaches.

4.4 Conclusions

The Composites Manufacturing Problem (CMP) is a complex, real-world problem with multiple layers of decisions. In this chapter, we formally defined an abstraction of the CMP called the Two-Stage Bin Packing and Hybrid Flowshop Scheduling Problem (2BPHFSP). Through modelling, analysis, and experimentation, the 2BPHFSP helped us to better understand some of the core complexities of the CMP as well as how to partition the problem into more manageable pieces. This knowledge will be essential to developing accurate and lightweight solution techniques to the CMP in the following chapters.

We developed five different approaches to solve the 2BPHFSP: Mixed Integer Programming (MIP), Constraint Programming (CP), two variations of Logic-based Benders Decomposition (LBBD1, LBBD2), and an Earliest Due Date heuristic (EDD). Each approach was tested on 11 sets of 30 instances, ranging from 5 to 100 jobs per instance. CP was able to find solutions for the smaller instances, but scaled poorly, while MIP was not able to find solutions for any instances. LBBD1 and LBBD2 performed the best, with insignificant differences in performance between the two variations. The LBBD approaches scaled better than CP, but still did not find feasible solutions for certain instances with 40 or more jobs. EDD was able to find a feasible solution within seconds for every instance, and often found solutions of comparable quality to LBBD1 and LBBD2, especially for larger instances.

To further investigate the high performance of EDD, two heuristic-CP approaches were tested on the same sets of problem instances. First, the heuristic solution was used to warm-start the CP model (WCP), and second, the packing from the heuristic solution was fixed and the scheduling subproblem from the LBBD models was used to improve the heuristic solution schedule (EDD-CP). Both methods were able to find feasible solutions for every instance. WCP performed better than EDD-CP within the range of problem instances tested, but showed signs of decreasing performance at around 50 to 60 jobs per instance. EDD-CP does not appear to have decreased performance up to 130 jobs per instance.

Overall, the most interesting conclusion is the comparable performance of heuristic techniques to the mathematical formulations, especially with larger instances. The complexity of the problem lends itself to CP, but scaling is an issue for complete approaches, especially as real-world instances contain 4000 jobs on average. The next two chapters will focus on considering some complexities of the original problem that were abstracted away, and increasing instance sizes to match the real world. The success shown by EDD and the hybrid heuristic-CP approaches suggests that complex problems such as the CMP pose a challenge for the development of robust and efficient exact methods. Thus, we will focus on developing heuristic-based approaches.

Chapter 5

Scaling Up: Complexity

The next step in our investigation of for the Composites Manufacturing Problem (CMP) is to include the key complexities that were relaxed in the abstracted problem. Thus, all models presented in this chapter solve the full problem defined in Chapter 2. We present the full models/algorithms for four solution techniques: Mixed Integer Programming (MIP) packing with Constraint Programming (CP) scheduling, CP packing with CP scheduling, Logic-based Benders Decomposition (LBBD), and an Earliest Due Date (EDD) packing heuristic with CP scheduling. Numerical results and a discussion of the results close the chapter.

5.1 Solution Approaches

Unlike the abstracted problem, the full problem as defined in Chapter 2 is too large and complex to be modelled using a single model. Thus, we split the problem into its packing and scheduling components. Separating the problem results in a loss of guarantee of optimality. However, based on the results of Chapter 4, optimality does not seem to be a realistic goal. We extend the packing portions of the MIP, CP, and EDD approaches defined in Chapter 4 and designate these packing models as *MIP-pack*, *CP-pack*, and *EDD-pack*.

For the scheduling component, numerical results from Chapter 4 showed that a MIP scheduling model is very unlikely too perform competitively with other approaches. A time-indexed MIP model is too large to solve such a complex problem. Also, the scheduling portion of the EDD heuristic from Chapter 4 cannot be extended as it is based on single-machine stages. Thus, we can only extend the scheduling portion of the CP model from Chapter 4 to be a scheduling model for the CMP, designated as *CP-sched*.

As we have a CP packing model and a CP scheduling model, an LBBD model with a similar structure and similar optimality cuts to the LBBD1 model from Chapter 4 can be formulated for the full problem, designated as *LBBD*. Adding cuts between the packing and scheduling models theoretically allows us to prove optimality, given enough time. However, the results of Chapter 4 imply that we most likely need to add time limits on the subproblems, which again removes the guarantee of optimality. In conclusion, we can create four different combinations of packing and scheduling components, labelled as Models 0 to 3 and shown in Table 5.1.

Table 5.1: Overview of models and algorithms (Models 0 to 3).

		Scheduling
		Constraint Programming **(CP-sched)**
Packing	Constraint Programming **(CP-pack)**	Model 0 (with LBBD cuts); Model 1
	Mixed Integer Programming **(MIP-pack)**	Model 2
	Earliest Due Date Packing Heuristic **(EDD)**	Model 3

There are an exponential number of job-to-batch packings that result in the same number of autoclave batches being formed. In this chapter, we are separately solving the packing and scheduling aspects of the problem. Thus, two unique packing solutions with the same number of autoclave batches may have different sums of tardiness after being scheduled, as jobs require multiple resources during processing in each stage and some resources have time-varying capacities. Figure 5.1 shows an example of how tardiness is affected by batch assignments and how start times can be delayed as a result of how jobs are assigned to batches.

We need a method that allows us to predict the "goodness" of a packing with respect to its eventual tardiness without having to schedule the packing. A natural idea is to push the packing model towards finding packings where jobs in a batch have similar due dates. However, based on the data, job due dates in production are very concentrated around specific dates (see Section 2.2). Thus, there may be hundreds of jobs with the same due date. An alternative measure of job closeness needs to be defined.

A relaxed CP scheduling problem was defined in Chapter 4 to find a global lower bound on tardiness. We can define a similar relaxed scheduling problem, designated as *RSP*, for the full problem to use as a packing guide. If two jobs are scheduled to end close together by the RSP, we can anticipate that they are likely to be scheduled close together in a full schedule. Thus, it is heuristically advantageous to batch them together. We first solve each problem instance using the RSP to obtain an ordering of jobs based on their RSP end times, then use that ordering to guide the packing models. This procedure is illustrated in Figure 5.2.

5.1.1 Relaxed Scheduling Problem

In the RSP, all packing constraints are relaxed and jobs are scheduled individually. Each job is assigned a bottom tool to be used from the beginning of tool preparation to the end of demould, and assigned one machine and one labour team for each relevant stage. In tool preparation, layup, and demould, each job uses one machine and its required number of labour teams throughout its processing time. These machines and labour teams are constrained by their availabilities over the schedule horizon. In the full problem, jobs are processed in autoclaves simultaneously with other jobs as part of an autoclave batch. In the RSP, each job takes up a certain amount of space in an autoclave. There is one autoclave ma-

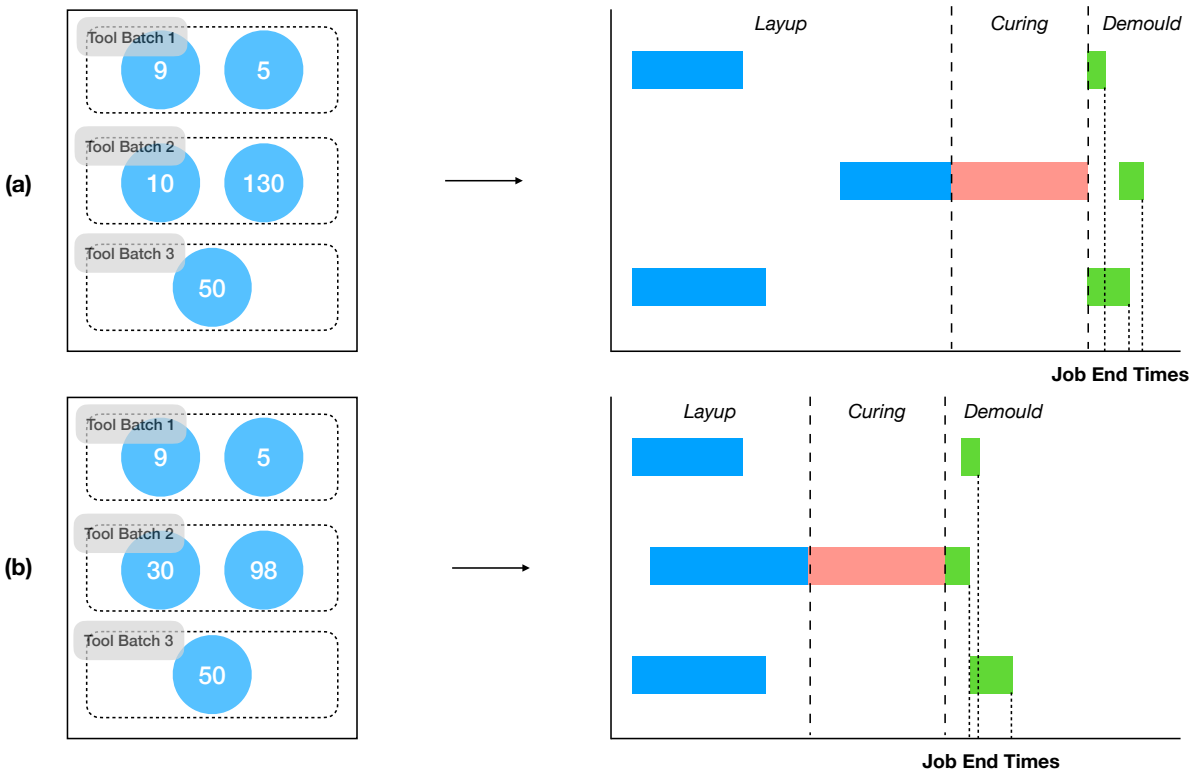


Figure 5.1: An example of the impact of packing decisions on scheduling decisions. In (a), we see an autoclave batch containing three tool batches with its best possible schedule. In (b), we see the same autoclave batch except tool batch 2 now contains jobs 30 and 98. All five jobs now have earlier end times as the new tool batch 2 can be scheduled earlier in the layup stage due to its different resource constraints.

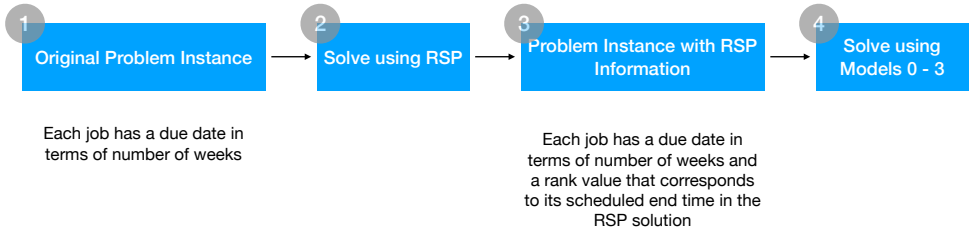


Figure 5.2: Using the RSP to obtain an ordering of jobs to be used during packing.

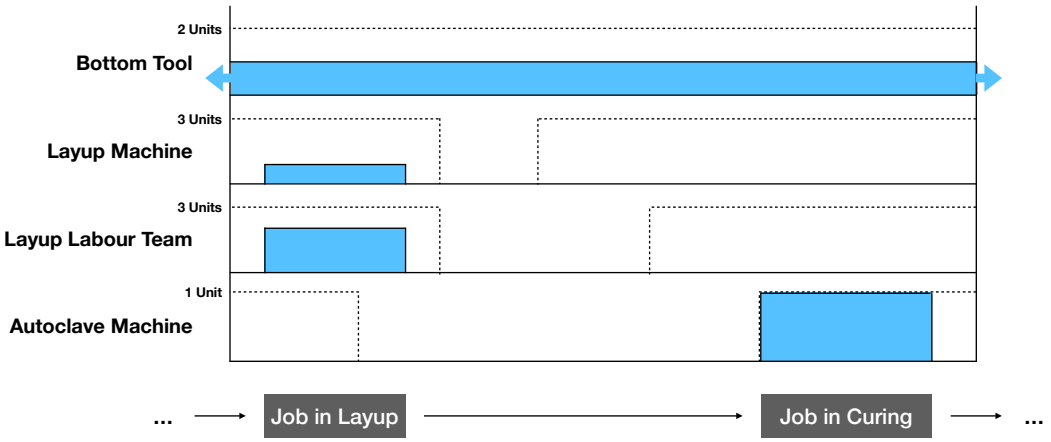


Figure 5.3: Sample scheduled job by the RSP in layup and curing alongside resource usage profiles; the bottom tool is still in use before layup and after curing. Dotted lines represent the maximum capacity of the resource.

chine available for each autoclave cycle type so the machines are constrained by their available volume throughout the schedule horizon. Figure 5.3 shows an example of how a job might be scheduled through the four stages and how that corresponds to resource usage.

We model and solve the RSP using a two-stage procedure:

1. Jobs are assigned to tool combinations based on a priority selection process to determine which machines and labour teams are required for processing. Each job j has a list of tool combinations. We calculate the ratio of bottom tool size over the total number of job slots present in any tool batch using that combination. From the allowed tool combinations of j, we choose the combination with the lowest size-to-slot ratio; if any ties exist, the one with the highest quantity of bottom tools available is chosen to break the tie. After the assignment, we now also know the machines and labour teams required as each tool combination comes with its own set of required resources for each stage.

2. The remaining scheduling problem is modelled using CP. There are four sets of interval variables corresponding to a job in each of the four stages. Each job is associated with an interval variable that spans its stage-specific interval variables. The objective of this model is to find the schedule with the minimum sum of job tardiness. Model variables and parameters are shown in Table 5.2.

Next, the model is presented in its entirety.

Table 5.2: RSP parameters and variables.

Parameter	Description
$j \in \mathcal{J}$	Set of jobs
$b \in \mathcal{N}$	Set of bottom tools
$m \in \mathcal{M}^{prep}$	Set of tool preparation machines
$m \in \mathcal{M}^{layup}$	Set of layup machines
$m \in \mathcal{M}^{cure}$	Set of curing machines
$m \in \mathcal{M}^{demould}$	Set of demould machines
$l \in \mathcal{L}^{prep}$	Set of tool preparation labour teams
$l \in \mathcal{L}^{layup}$	Set of layup labour teams
$l \in \mathcal{L}^{demould}$	Set of demould labour teams
$n \in \mathcal{H}$	Set of time periods
s_j	Size of job j
d_j	Due date of job j
b_j	Bottom tool of tool combination assigned to job j
cap_b	Capacity of bottom tool b, i.e. maximum number of jobs that can fit on tool
q_b	Quantity of bottom tool b available
m_j^{prep}	Tool preparation machine based on tool combination assigned to job j
m_j^{layup}	Layup machine based on tool combination assigned to job j
m_j^{cure}	Curing machine based on tool combination assigned to job j
$m_j^{demould}$	Demould machine of job j
l_j^{prep}	Tool preparation labour team of job j
l_j^{layup}	Layup labour team of job j
$l_j^{demould}$	Demould labour team of job j
lq_j^{prep}	Quantity of tool preparation labour teams required to process job j
lq_j^{layup}	Quantity of layup labour teams required to process job j
$lq_j^{demould}$	Quantity of demould labour teams required to process job j
q_n^m	Quantity (or volume if autoclave) of machine m available at time period n
q_n^l	Quantity of labour team l available at time period n

Variable	Description
$prep_j$	Interval variable of job j in tool preparation
$layup_j$	Interval variable of job j in layup
$cure_j$	Interval variable of job j in curing
$demould_j$	Interval variable of job j in demould
tot_j	Interval variable spanning all interval variables of job j
t_j	Tardiness of job j

$$\min \quad \sum_{j \in \mathcal{J}} t_j \tag{5.1}$$

$$\text{s.t.} \quad \texttt{endBeforeStart}(prep_j, layup_j) \qquad \forall j \in \mathcal{J} \tag{5.2}$$

$$\texttt{endBeforeStart}(layup_j, cure_j) \qquad \forall j \in \mathcal{J} \tag{5.3}$$

$$\texttt{endBeforeStart}(cure_j, demould_j) \qquad \forall j \in \mathcal{J} \tag{5.4}$$

$$\texttt{span}(tot_j, \{prep_j, demould_j\}) \qquad \forall j \in \mathcal{J} \tag{5.5}$$

$$\texttt{alwaysIn}\left(\sum_{j \in \{\mathcal{J} | b_j = b\}} \texttt{pulse}(tot_j, 1), 0, |\mathcal{H}|, 0, cap_b \times q_b\right) \qquad \forall b \in \mathcal{N} \tag{5.6}$$

$$\texttt{alwaysIn}\left(\sum_{j \in \{\mathcal{J} | m_j^{prep} = m\}} \texttt{pulse}(prep_j, 1), n, 0, q_n^m\right) \qquad \forall m \in \mathcal{M}^{prep}, \quad \forall n \in \mathcal{H} \tag{5.7}$$

$$\texttt{alwaysIn}\left(\sum_{j \in \{\mathcal{J} | m_j^{layup} = m\}} \texttt{pulse}(layup_j, 1), n, 0, q_n^m\right) \qquad \forall m \in \mathcal{M}^{layup}, \quad \forall n \in \mathcal{H} \tag{5.8}$$

$$\texttt{alwaysIn}\left(\sum_{j \in \{\mathcal{J} | m_j^{cure} = m\}} \texttt{pulse}(auto_j, s_j), n, 0, q_n^m\right) \qquad \forall m \in \mathcal{M}^{cure}, \quad \forall n \in \mathcal{H} \tag{5.9}$$

$$\texttt{alwaysIn}\left(\sum_{j \in \{\mathcal{J} | m_j^{demould} = m\}} \texttt{pulse}(demould_j, 1), n, 0, q_n^m\right) \qquad \forall m \in \mathcal{M}^{demould}, \quad \forall n \in \mathcal{H} \tag{5.10}$$

$$\texttt{alwaysIn}\left(\sum_{j \in \{\mathcal{J} | l_j^{prep} = l\}} \texttt{pulse}(prep_j, lq_j^{prep}), n, 0, q_n^l\right) \qquad \forall l \in \mathcal{L}^{prep}, \quad \forall n \in \mathcal{H} \tag{5.11}$$

$$\texttt{alwaysIn}\left(\sum_{j \in \{\mathcal{J} | l_j^{layup} = l\}} \texttt{pulse}(layup_j, lq_j^{layup}), n, 0, q_n^l\right) \qquad \forall l \in \mathcal{L}^{layup}, \quad \forall n \in \mathcal{H} \tag{5.12}$$

$$\texttt{alwaysIn}\left(\sum_{j \in \{\mathcal{J} | l_j^{demould} = l\}} \texttt{pulse}(demould_j, lq_j^{demould}), n, 0, q_n^l\right) \qquad \forall l \in \mathcal{L}^{demould}, \quad \forall n \in \mathcal{H} \tag{5.13}$$

$$t_j \geq \texttt{endOf}(demould_j) - d_j \qquad \forall j \in \mathcal{J} \tag{5.14}$$

$$t_j \in \{0, ...|\mathcal{H}|\} \ \forall j \in \mathcal{J}$$

Constraints (5.2) to (5.4) enforce sequencing constraints on the stage-specific interval variables. Tool preparation before layup, layup before curing, and curing before demould. Constraint (5.5) connects the stage-specific interval variables to the spanning interval variable for each job. This connection is required by Constraint (5.6) which uses one slot of a bottom tool for the entire time a job is being processed. Constraint (5.6) also enforces that only the available quantity of tool slots can be used at any point in time. Constraints (5.7) to (5.13) enforce resource availabilities for machines and labour teams. With the exception of Constraint (5.9), which deals with autoclave machines, all the constraints enforce that only the available quantity of machines or labour teams can be used at any point in time. This availability changes throughout the horizon due to predefined shift changes and machine downtime. Constraint (5.9) is slightly different as it enforces that only the available capacity, rather than quantity, of each autoclave machine can be used at any point in time. Constraint (5.14) defines job tardiness.

Once jobs are scheduled using the RSP, we define rsp_j to be the RSP rank of job j and equate rsp_j to the end time of job j in the demould stage. An ordered list of jobs according to their RSP ranks $rsp_{j_0} \leq rsp_{j_1} \leq rsp_{j_2} \leq ...$ would then be equal to $\{j_0, j_1, j_2, ...\}$.

5.1.2 Constraint Programming Packing

First, let us describe the packing problem being solved by CP-pack. The first step is to assign jobs to tool combinations. Each tool combination can fit a predetermined number of jobs, so we need to calculate how many tool combinations need to be "opened" to fit all of its assigned jobs. The eventual objective of the packing component is to minimize the number of autoclave batches so we should choose tool combinations with smaller sizes if possible. Each open tool combination sets the foundation for one tool batch and jobs should be partitioned into tool batches such that jobs with similar rsp_j values are batched together. A tool batch k then inherits the average RSP rank of its jobs, $rsp_k = \frac{1}{|\{j \text{ in } k\}|} \sum_{j \in \{j \text{ in } k\}} rsp_j$. Next, tool batches with the same autoclave cycle need to be assigned to autoclave batches. As aforementioned, the main objective is to minimize the number of autoclave batches, but we also have a secondary objective of minimizing the distribution of tool batch RSP ranks within each autoclave batch.

This complex packing is difficult to express in a monolithic model, so we defined CP-pack to be the following sequence that takes jobs as an input and outputs autoclave batches:

1. Pack all jobs with only one mapped tool combination that contains one job slot, creating a set of single-job tool batches. These assignment are not heuristically determined. Any job j with only one tool combination option needs to be assigned to that tool combination. Furthermore, as the tool combination only has room for one job, there is no other option for j aside from being batched into a tool batch by itself. Thus, these single-job tool batches will exist in any feasible solution.

2. Assign remaining jobs to tool combinations using CP with the objective of minimizing the ratios of bottom tool size to the total number of job slots over the assigned combinations.

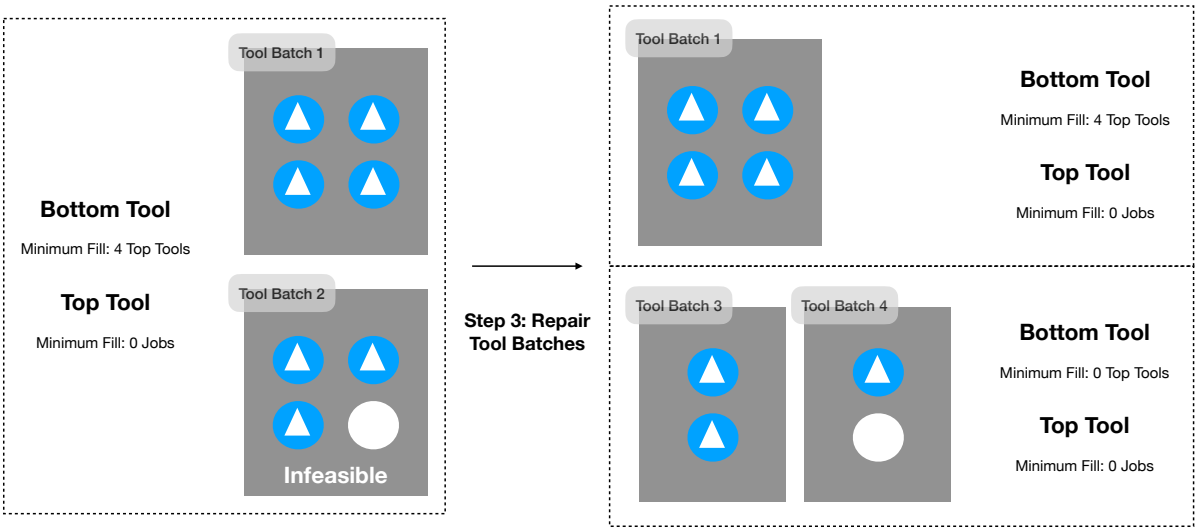


Figure 5.4: Fixing tool batch infeasibility arising from minimum fill requirements. The tool batches on the left are formed as a result of step 2 in the packing sequence. However, the bottom tool has a minimum fill requirement that is not satisfied by tool batch 2. After repairing the batches with step 3 in the packing sequence, the top tool batch remains unchanged, but the jobs in tool batch 2 have been reassigned to a different tool combination, forming tool batches 3 and 4 which are feasible.

3. Repair tool packing. We relaxed the minimum tool fill requirement in step 2 so there may be tool combinations where we have assigned jobs that cannot fill the minimum number of slots required to open a new tool batch. Based on the characteristics of job-to-tool-combination mappings in the data, we know that if a job has more than one possible tool combination it can be assigned to, it is guaranteed that a least one of those combinations either does not have a minimum fill requirement or is a single-job combination (i.e. only has one slot available). Thus, if we end up with a set of jobs that cannot fill the chosen tool combination to its minimum level, we are able to choose the alternate combination. This step guarantees that all tool batches will be feasible. Figure 5.4 explains how this repair process happens.

4. Assign tool batches to autoclave batches using CP with a primary objective of minimizing the number of autoclave batches and a secondary objective of minimizing the distribution of tool batch RSP rankings within each autoclave batch.

The model variables and parameters for step 2, the job-to-tool-combination packing problem, are shown in Table 5.3.

$$\min \quad \sum_{c \in \mathcal{C}} r_c \times \lambda_c \tag{5.15}$$

$$\text{s.t.} \quad \texttt{allowedAssignments}(c_j, \{c \mid \mathcal{C}^j\}) \qquad \forall j \in \mathcal{J} \tag{5.16}$$

$$\texttt{count}(x_j, c) \leq cap_b^c \times cap_o^c \times \lambda_c \qquad \forall c \in \mathcal{C} \tag{5.17}$$

$$x_j \in \{0, ...|\mathcal{C}|\}\ \forall j \in \mathcal{J};\ \lambda_c \in \{0, ...|\mathcal{J}|\}\ \forall c \in \mathcal{C}$$

Constraint (5.16) makes sure each job can only be assigned to one of its mapped tool combinations. Constraint (5.17) determines the number of tool batches that have to be opened for each tool combina-

Table 5.3: CP job-to-tool-combination packing parameters and variables.

Parameter	Description
$j \in \mathcal{J}$	Set of jobs
$c \in \mathcal{C}$	Set of tool combinations
$c \in \mathcal{C}^j$	Set of tool combinations mapped to job j
r_c	Ratio of bottom tool size to number of available slots for tool combination c
cap_b^c	Number of available slots in bottom tool b of combination $c = (b, o)$
cap_o^c	Number of available slots in top tool o of combination $c = (b, o)$

Variable	Description
x_j	Tool combination that job j is assigned to
λ_c	Number of tool combinations c that were opened

Table 5.4: CP tool-batch-to-autoclave-batch packing parameters and variables.

Parameter	Description
$k \in \mathcal{B}^{1*}$	Set of packed tool batches
$i \in \mathcal{B}^2$	Set of autoclave batches
$b \in \mathcal{N}^*$	Set of bottom tools used by tool batches
σ_k	Size of tool batch k
b_k	Bottom tool of tool batch k
rsp_k	RSP rank of tool batch k
ac_k	Autoclave cycle type required by tool batch k
ac_i	Predefined autoclave cycle type of autoclave batch i
q_b	Quantity of bottom tool b available

Variable	Description
y_k	Autoclave batch that tool batch k is assigned to
v_i	Size of autoclave batch i
ϕ_i	Lowest RSP rank of all tool batches in autoclave batch i
π_i	Highest RSP rank of all tool batches in autoclave batch i
a	Number of autoclave batches that were opened

tion. The `count` constraint counts the number of jobs have been assigned to a combination c and the expression $cap_b^c \times cap_o^c$ gives us the number of available slots on a tool batch using combination c. Thus, we constrain $cap_b^c \times cap_o^c \times \lambda_c$ to be greater than the number of jobs assigned to c so λ_c then gives us the minimum number of tool batches using c that have to be opened.

Model variables and parameters for step 4, the tool-batch-to-autoclave-batch packing problem, are shown in Table 5.4.

Table 5.5: MIP job-to-tool-combination packing variables.

Variable	Description
$x_{j,c}$	Equal to 1 if job j is assigned to tool combination c, otherwise equal to 0
λ_c	Number of tool combination c that were opened

$$\min \quad a + \sum_{i \in \mathcal{B}^2} (\pi_i - \phi_i) \tag{5.18}$$

$$\text{s.t.} \quad \texttt{pack}\left(\{v_i \mid i \in \mathcal{B}^2, \{y_k \mid k \in \mathcal{B}^{1*}\}, \{\sigma_k | k \in \mathcal{B}^{1*}\}\right) \tag{5.19}$$

$$\texttt{allowedAssignments}\left(y_k, \{i \in \mathcal{B}^2 \mid ac_k = ac_i\}\right) \quad \forall k \in \mathcal{B}^{1*} \tag{5.20}$$

$$a = |\mathcal{B}^{1*}| - |\{\mathcal{B}^2 \mid v_i > 0\}| \tag{5.21}$$

$$\texttt{GCC}\left(\{0, ..., q_b\}, \{0, ..., |\mathcal{B}^2|\}, \{y_k \mid b = b_k\}\right) \quad \forall b \in \mathcal{N}^* \tag{5.22}$$

$$\texttt{element}\left(\{\phi_i \mid i \in \mathcal{B}^2\}, y_k\right) \leq rsp_k \quad \forall k \in \mathcal{B}^{1*} \tag{5.23}$$

$$\texttt{element}\left(\{\pi_i \mid i \in \mathcal{B}^2\}, y_k\right) \geq rsp_k \quad \forall k \in \mathcal{B}^{1*} \tag{5.24}$$

$$y_k \in \{0, ...|\mathcal{B}^2|\}; \; v_i \in \{0, ...\texttt{capacity of } ac_i\}$$

$$\phi_i, \pi_i \in \{rsp_k \mid k \in \mathcal{B}^{1*}\} \; \forall i \in \mathcal{B}^2; \; a \in \{0, ...|\mathcal{B}^2|\}$$

Constraint (5.19) packs tool batches into autoclave batches while enforcing autoclave capacity limits. Constraint (5.20) only allows tool batches to be assigned to autoclave batches with the same cycle type.[1] Constraint (5.21) defines the number of open autoclave batches. Constraint (5.22) makes sure only the available number of bottom tools are present in any autoclave batch. Constraints (5.23) and (5.24) define the minimum and maximum tool batch RSP rankings in an autoclave batch.

5.1.3 Mixed Integer Programming Packing

MIP-pack follows the same packing sequence as described in Section 5.1.2, with the two mathematical models formulated using MIP.

Model variables for step 2, the job-to-tool-combination packing problem, are shown in Table 5.5; see Table 5.3 for model parameters.

$$\min \quad \sum_{c \in \mathcal{C}} r_c \times \lambda_c \tag{5.25}$$

$$\text{s.t.} \quad \sum_{c \in \mathcal{C}} x_{j,c} = 1 \quad \forall j \in \mathcal{J} \tag{5.26}$$

$$\sum_{\{c \in \mathcal{C} | c \notin \mathcal{C}^j\}} x_{j,c} = 0 \quad \forall j \in \mathcal{J} \tag{5.27}$$

$$\sum_{j \in \mathcal{J}} x_{j,c} \leq cap_b^c \times cap_o^c \times \lambda_c \quad \forall c \in \mathcal{C} \tag{5.28}$$

$$x_{j,c} \in \{0, 1\} \; \forall j \in \mathcal{J}, \; \forall c \in \mathcal{C}; \; \lambda_c \in \{0, ...|\mathcal{J}|\} \; \forall c \in \mathcal{C}$$

[1] The initial set of empty autoclave batches is the same size as the set of packed tool batches as the worst case scenario has each tool batch assigned to its own autoclave batch. Thus, each empty autoclave batch corresponds to a tool batch and inherits its autoclave cycle type.

Table 5.6: MIP tool-batch-to-autoclave-batch packing variables.

Variable	Description
$x_{k,i}$	Equal to 1 if tool batch k is assigned to autoclave batch i, otherwise equal to 0
γ_i	Equal to 1 if autoclave batch i is open, otherwise equal to 0
v_i	Size of autoclave batch i
ϕ_i	Lowest RSP rank of all tool batches in autoclave batch i
π_i	Highest RSP rank of all tool batches in autoclave batch i
a	Number of open autoclave batches

Constraint (5.26) assigns each job to one tool combination. Constraint (5.27) makes sure jobs are only assigned to mapped tool combinations. Constraint (5.28) counts the number of tool batches that have to be opened based on how many jobs have been assigned to each tool combination.

Model variables and parameters for step 4, the tool-batch-to-autoclave-batch packing problem, are shown in Table 5.6; see Table 5.4 for model parameters.

$$\min \quad a + \sum_{i \in \mathcal{B}^2} (\pi_i - \phi_i) \tag{5.29}$$

$$\text{s.t.} \quad \sum_{i \in \mathcal{B}^2} x_{k,i} = 1 \qquad \forall k \in \mathcal{B}^{1*} \tag{5.30}$$

$$\sum_{\{i \in \mathcal{B}^2 | ac_k \neq ac_i\}} x_{k,i} = 0 \qquad \forall k \in \mathcal{B}^{1*} \tag{5.31}$$

$$\gamma_i \geq x_{k,i} \qquad \forall k \in \mathcal{B}^{1*} \quad \forall i \in \mathcal{B}^2 \tag{5.32}$$

$$a \geq \sum_{i \in \mathcal{B}^2} \gamma_i \tag{5.33}$$

$$v_i = \sum_{k \in \mathcal{B}^{1*}} \sigma_k \times x_{k,i} \qquad \forall i \in \mathcal{B}^2 \tag{5.34}$$

$$\sum_{\{k \in \mathcal{B}^{1*} | b_k = b\}} x_{k',i} \leq q_b \qquad \forall b \in \mathcal{N}^*, \quad \forall i \in \mathcal{B}^2 \tag{5.35}$$

$$\phi_i \leq rsp_k + M\,(1 - x_{k,i}) \qquad \forall k \in \mathcal{B}^{1*}, \quad \forall i \in \mathcal{B}^2 \tag{5.36}$$

$$\pi_i \geq rsp_k - M\,(1 - x_{k,i}) \qquad \forall k \in \mathcal{B}^{1*}, \quad \forall i \in \mathcal{B}^2 \tag{5.37}$$

$$x_{k,i} \in \{0,1\}\ \forall k \in \mathcal{B}^1,\ \forall i \in \mathcal{B}^2;\ \gamma_i \in \{0,1\}\ \forall i \in \mathcal{B}^2;\ a \in \{0, ...|\mathcal{B}^2|\}$$

$$v_i \in \{0, ...\text{capacity of } ac_i\}\ \forall i \in \mathcal{B}^2;\ \phi_i, \pi_i \in \{rsp_k \mid k \in \mathcal{B}^{1*}\}\ \forall i \in \mathcal{B}^2$$

Constraint (5.30) assigns each tool batch to one autoclave batch. Constraint (5.31) makes sure tool batches are only assigned to an autoclave batch with the same cycle type. Constraint (5.32) defines the indicator variable that equals 1 if an autoclave batch is open. Constraint (5.33) defines the number of open autoclave batches. Constraint (5.34) defines the size of each autoclave batch. Constraint (5.35) makes sure only the available quantity of bottom tools is used in an autoclave batch. Constraints (5.36) and (5.37) define the minimum and maximum tool batch RSP ranks within an autoclave batch.

5.1.4 Earliest Due Date Packing

The EDD packing algorithm follows a similar sequence as the packing portion of the EDD algorithm used to solve the abstracted problem (see Section 4.2.4). First, jobs are packed into tool batches. Jobs are sorted by increasing due date and then, within each block of jobs with the same due date, by RSP ranking. To form a tool batch, we first randomly select a job j from the first L items of $\mathcal{J}$, where L is a predetermined length of randomization. Let c be a tool combinations from $\mathcal{C}^j$, the set of mapped tool combinations to j, where $cap_b^c \times cap_o^c$ is the total number of slots in a tool batch using c and s_b is the size of the bottom tool in c. We define the bottom tool size per slot ratio as equal to $\frac{s_b}{cap_b^c \times cap_o^c}$ and assign j to the tool combination with the smallest bottom tool size per slot from $\mathcal{C}^j$. If any ties exist, we select the tool combination with the highest quantity of bottom tools available. The selected tool combination forms the foundation of the current tool batch k. We iterate through the remaining jobs in order and add any jobs that can fit into empty slots in k. Once we are no longer able to add jobs, if k does not meet the minimum fill requirement, all jobs from k are added back into $\mathcal{J}$ and the tool combination is removed from $\mathcal{C}^j$. Packing proceeds until all jobs are assigned to a tool batch. Each job j is guaranteed to have a tool combination with no minimum fill requirement in $\mathcal{C}^j$, thus, we are guaranteed to be able to pack all jobs into tool batches using this procedure.

The list of tool batches sorted by increasing RSP rank is denoted by $\mathcal{B}^1$. To form an autoclave batch, we first randomly select a tool batch k from the first L items of $\mathcal{B}^1$. The autoclave cycle type of k determines the cycle type of the current autoclave batch. Then, we iterate through the remaining tool batches and add any tool batches with a matching cycle type to the current autoclave batch until either the autoclave batch is full or there are no more eligible tool batches remaining. Note that tool batches are also added to an autoclave batch only if adding the tool batch does not exceed any available tool quantities. Packing proceeds until all tool batches are assigned to an autoclave batch. We are guaranteed to be able to pack all tool batches as in the worst case scenario, a tool batch can be assigned to its own autoclave batch. Algorithm 5 formally describes this packing procedure.

5.1.5 Constraint Programming Scheduling

Given the tool batches and autoclave batches produced by a packing approach, we now need to schedule these batches in the four stages of the CMP. Each tool batch needs to be scheduled once in tool preparation, layup, and demould, and each autoclave batch needs to be scheduled once in curing. The objective of the scheduling problem is to minimize the sum of job tardiness. We chose this objective due to the fact that reducing tardiness can be directly correlated to reducing cost. Model variables and parameters are shown in Table 5.7.

Algorithm 5: EDD Packing

Result: Feasible packing
for *each repetition* **do**
  while $\mathcal{J}$ *not empty* **do**
    sort $\mathcal{J}$ by increasing due date;
    $j \leftarrow$ randomly selected job from first L items of $\mathcal{J}$;
    create new tool batch k and add to $\mathcal{B}^1$;
    pick the tool combination c with the smallest bottom tool size per slot ratio out of the set $c \in \mathcal{C}^j$;
    assign tool batch k to use combination c;
    remove j from $\mathcal{J}$ and assign to k;
    iterate through $\mathcal{J}$ and add jobs that are also mapped to c to k until all job slots in k are filled or no more eligible jobs are left
    remove all jobs added to k from $\mathcal{J}$
    if k *does not meet minimum fill* **then**
      add all jobs from k back into $\mathcal{J}$;
      remove tool combination c from $\mathcal{C}^j$;
    end
  end
  while $\mathcal{B}^1$ *not empty* **do**
    sort $\mathcal{B}^1$ by increasing RSP rank;
    $k \leftarrow$ randomly selected tool batch from first L items of $\mathcal{B}^1$;
    create new autoclave batch i and add to $\mathcal{B}^2$;
    set the cycle type of autoclave batch i to be equal to the cycle type required by tool batch k;
    remove k from $\mathcal{B}^1$ and assign to i;
    iterate through $\mathcal{B}^1$, if a tool batch k^* has the same cycle type as i and adding k^* will not exceed any available tool quantities in i, add k^* to i and remove from $\mathcal{B}^1$;
  end
  add packing solution to solution list;
end
pick best packing solution from solution list[2];

Table 5.7: CP scheduling parameters and variables.

Parameter	Description
$j \in \mathcal{J}$	Set of jobs
$k \in \mathcal{B}^1$	Set of tool batches
$i \in \mathcal{B}^2$	Set of autoclave batches
$b \in \mathcal{N}$	Set of bottom tools
$m \in \mathcal{M}^{prep}$	Set of tool preparation machines
$m \in \mathcal{M}^{layup}$	Set of layup machines
$m \in \mathcal{M}^{cure}$	Set of curing machines
$m \in \mathcal{M}^{demould}$	Set of demould machines
$l \in \mathcal{L}^{prep}$	Set of tool preparation labour teams
$l \in \mathcal{L}^{layup}$	Set of layup labour teams
$l \in \mathcal{L}^{demould}$	Set of demould labour teams
$n \in \mathcal{H}$	Set of time periods
d_j	Due date of job j
k_j	Tool batch that job j is assigned to
b_k	Bottom tool of tool batch k
i_k	Autoclave batch that tool batch k is assigned to
q_b	Quantity of bottom tool b available
m_k^{prep}	Tool preparation machine of tool batch k
m_k^{layup}	Layup machine of tool batch k
m_i^{cure}	Curing machine of autoclave batch i
$m_k^{demould}$	Demould machine of tool batch k
l_k^{prep}	Tool preparation labour team of tool batch k
l_k^{layup}	Layup labour team of tool batch k
$l_k^{demould}$	Demould labour team of tool batch k
lq_k^{prep}	Quantity of tool preparation labour teams required to process tool batch k
lq_k^{layup}	Quantity of layup labour teams required to process tool batch k
$lq_k^{demould}$	Quantity of demould labour teams required to process tool batch k
q_n^m	Quantity of machine m available at time period n
q_n^l	Quantity of labour team l available at time period n

Variable	Description
$prep_k$	Interval variable for tool batch k in tool preparation
$layup_k$	Interval variable for tool batch k in layup
$cure_i$	Interval variable for curing common to all tool batches in autoclave batch i
$demould_k$	Interval variable for tool batch k in demould
tot_k	Interval variable spanning all interval variables of tool batch k
t_j	Tardiness of job j

$$\min \quad \sum_{j \in \mathcal{J}} t_j \tag{5.38}$$

$$\text{s.t.} \quad \texttt{endBeforeStart}(prep_k, layup_k) \qquad \forall k \in \mathcal{B}^1 \tag{5.39}$$

$$\texttt{endBeforeStart}(layup_k, \{cure_i \mid i = i_k\}) \qquad \forall k \in \mathcal{B}^1 \tag{5.40}$$

$$\texttt{endBeforeStart}(\{cure_i \mid i = i_k\}, demould_k) \qquad \forall k \in \mathcal{B}^1 \tag{5.41}$$

$$\texttt{alwaysIn}\left(\sum_{k \in \{\mathcal{B}^1 | b = b_k\}} pulse(tot_k, 1), 0, |\mathcal{H}|, 0, q^b\right) \qquad \forall b \in \mathcal{N} \tag{5.42}$$

$$\texttt{alwaysIn}\left(\sum_{k \in \{\mathcal{B}^1 | m = m_k^{prep}\}} \texttt{pulse}(prep_k, 1), n, 0, q_n^m\right) \qquad \forall m \in \mathcal{M}^{prep}, \quad \forall n \in \mathcal{H} \tag{5.43}$$

$$\texttt{alwaysIn}\left(\sum_{k \in \{\mathcal{B}^1 | m = m_k^{layup}\}} \texttt{pulse}(layup_k, 1), n, 0, q_n^m\right) \qquad \forall m \in \mathcal{M}^{layup}, \quad \forall n \in \mathcal{H} \tag{5.44}$$

$$\texttt{alwaysIn}\left(\sum_{i \in \{\mathcal{B}^2 | m = m_i^{cure}\}} \texttt{pulse}(auto_k, 1), n, 0, q_n^m\right) \qquad \forall m \in \mathcal{M}^{cure}, \quad \forall n \in \mathcal{H} \tag{5.45}$$

$$\texttt{alwaysIn}\left(\sum_{k \in \{\mathcal{B}^1 | m = m_k^{demould}\}} \texttt{pulse}(demould_k, 1), n, 0, q_n^m\right) \qquad \forall m \in \mathcal{M}^{demould}, \quad \forall n \in \mathcal{H} \tag{5.46}$$

$$\texttt{alwaysIn}\left(\sum_{k \in \{\mathcal{B}^1 | l = l_k^{prep}\}} \texttt{pulse}(prep_k, lq_k^{prep}), n, 0, q_n^l\right) \qquad \forall l \in \mathcal{L}^{prep}, \quad \forall n \in \mathcal{H} \tag{5.47}$$

$$\texttt{alwaysIn}\left(\sum_{k \in \{\mathcal{B}^1 | l = l_k^{layup}\}} \texttt{pulse}(layup_k, lq_k^{layup}), n, 0, q_n^l\right) \qquad \forall l \in \mathcal{L}^{layup}, \quad \forall n \in \mathcal{H} \tag{5.48}$$

$$\texttt{alwaysIn}\left(\sum_{k \in \{\mathcal{B}^1 | l = l_k^{demould}\}} \texttt{pulse}(demould_k, lq_k^{demould}), n, 0, q_n^l\right) \qquad \forall l \in \mathcal{L}^{demould}, \quad \forall n \in \mathcal{H} \tag{5.49}$$

$$t_j \geq \texttt{endOf}(\{demould_k \mid k = k_j\}) - d_j \qquad \forall j \in \mathcal{B}^1 \tag{5.50}$$

$$t_j \in \{0, ...|\mathcal{H}|\} \; \forall j \in \mathcal{J}$$

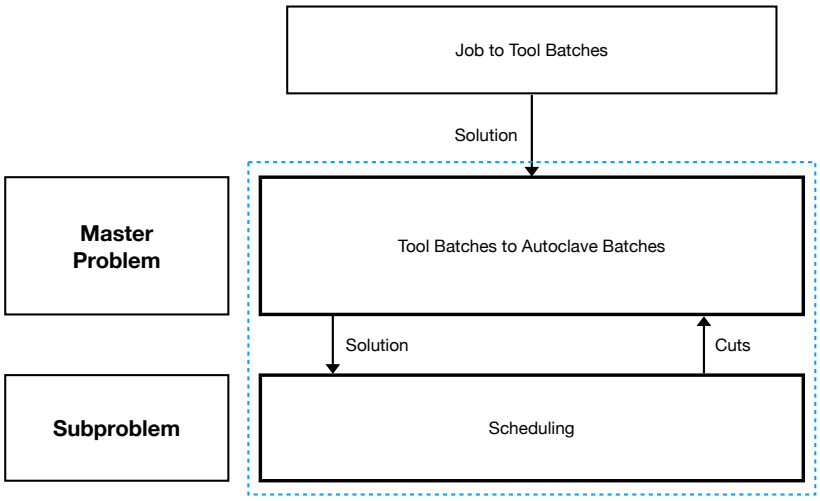


Figure 5.5: Two-stage decomposition flow between problems.

Constraints (5.39) to (5.41) enforce sequencing between tool preparation, layup, curing, and demould activities. Constraint (5.42) makes sure only the available quantity of bottom tools are used at any point in time. Constraints (5.43) to (5.49) make sure only the available quantity of machines and labour teams within a time period is used during that time period. Constraint (5.50) defines job tardiness.

Logic-based Benders Decomposition Cuts. We can use the CP packing model presented in Section 5.1.2 with the above CP scheduling model to form a two-stage Logic-based Benders Decomposition (LBBD) where the master problem consists of the tool-batch-to-autoclave-batch packing and the subproblem consists of the entire scheduling problem. Figure 5.5 shows the decomposition flow.

After the packing and scheduling is done, we can add optimality cuts of a similar form as the no-good optimality cuts defined for the abstracted problem to re-solve the problem from the master problem (see Section 4.2.3). Note that results from Chapter 4 show the dual bounds of Section 4.2.3 do not improve the search process in any meaningful way, so we only use the no-good optimality cuts for this LBBD model. We introduce a new set of binary variables $\omega_{i^*} \in \{0,1\}$ where $i^* \in \mathcal{B}^{2*}$ is the set of open autoclave batches. In the abstracted problem, we cut job to autoclave batch assignments after every cycle. But since we are now assigning tool batches to autoclave batches, we need to modify the cuts accordingly. We set $\omega_{i*} = 1$ if autoclave batch i^* is required to be different and then enforce that only strict subsets of tool batches in that autoclave batch can appear together again. At least one $\{\omega_{i^*} | i^* \in \mathcal{B}^{2*}\}$ is required to be equal to 1. Constraints (5.51) and (5.52) form the no-good optimality cuts.

$$(\omega_{i^*} == 1) \rightarrow \Big(\texttt{count}(\{y_k \mid k \in \mathcal{B}^{1,i^*}\}, i) \leq |\mathcal{B}^{1,i^*} - 1|\Big) \qquad \forall i \in \mathcal{B}^2, \quad \forall i^* \in \mathcal{B}^{2*} \tag{5.51}$$

$$\sum_{i^* \in \mathcal{B}^{2*}} \omega_{i^*} \geq 1 \tag{5.52}$$

$$\omega_{i^*} \in \{0,1\} \; \forall i^* \in \mathcal{B}^{2*}$$

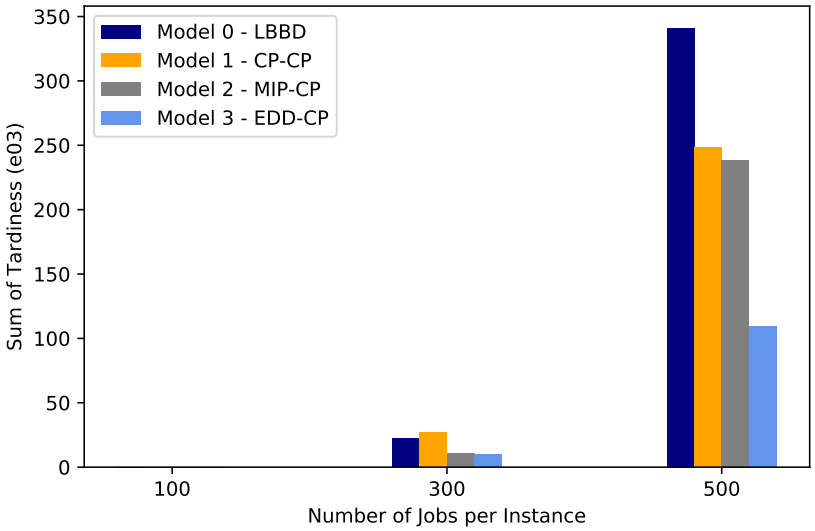


Figure 5.6: Comparison of average sum of tardiness between solution techniques.

5.2 Numerical Results

The four solution techniques presented in this chapter were tested on 3 sets of 30 problem instances with 100, 300 and 500 jobs per instance, respectively. Problem instances were sampled using the technique described in Section 2.2.1. All models and algorithms were implemented in Java and all mathematical models were implemented using CPLEX Optimization Studio 12.9. Models 1 to 3 were given a thirty minute time limit for packing and a thirty minute time limit for scheduling to solve an instance. Model 0, the LBBD approach, was given a sixty minute total time limit and the master problem and subproblem were individually given two minute time limits.

Overall Comparison of Solution Approaches. All four solution techniques were able to find feasible solution for all instances. Figure 5.6 compares the average sum of tardiness found by the different techniques..

It is difficult to compare solution techniques by looking at the 100 job and 300 job instances as the tardiness values are so low. However, when we get to the 500 job instances, we can see that Model 0 has the worst performance, followed by Models 1 and 2, then Model 3, which has the best peformance. Model 2 finds slightly better schedules with MIP-pack and CP-sched than Model 1 which uses CP-pack and CP-sched; this result weakly implies that MIP may be more capable of finding better solutions to complex packing problems than CP. However, the discrepancy is small and ultimately does not motivate any investigation into the differences between MIP and CP. The more interesting result from Figure 5.6 is the impressive performance of Model 3. The average sum of tardiness for instances with 500 jobs is significantly lower for Model 3 than all other techniques. We can hypothesize that because the number of open autoclave batches still falls within the same general range for Models 1, 2, and 3, the distribution of jobs amongst batches, which we denote as packing quality, is most likely accounting for the difference in tardiness.

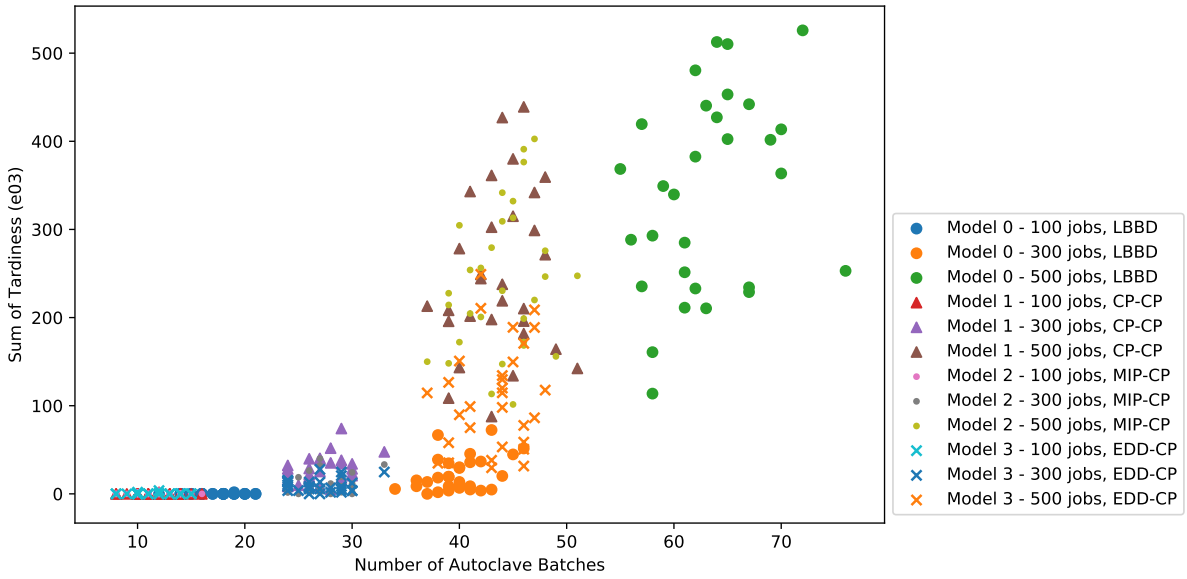


Figure 5.7: Connection between number of autoclave batches and sum of tardiness.

Figure 5.7 gives a more detailed look at the number of autoclave batches compared to tardiness for each instance. The tardiness of Model 0 solutions falls within the same range as Model 1 and 2 solution tardiness despite the larger number of open autoclave batches. Model 0 cycles between the master problem and subproblem 15 times within the one hour limit to solve an instance. Figure 5.8 shows how much tardiness varies throughout these 15 iterations for a few sample instances. This variance supports our previous hypothesis about the packing quality also having an impact on tardiness. Model 0 is able to find different distributions of jobs across batches by iterating between the master problem and subproblem, thus finding better schedules compared to other models that open same number of open autoclave batches. However, the large number of autoclave batches opened by Model 0 offsets this ability to improve tardiness as the number of autoclave batches is still positively correlated with tardiness, which is why the average tardiness for Model 0 in Figure 5.6 is the highest. Therefore, Model 0 is still considered to be the worst-performing model.

In conclusion, we have a clear ranking of solutions: Model 3, using EDD-pack and CP-sched, is the best technique so far while Model 0, the LBBD model, is the worst. Models 1, using CP-pack and CP-sched, and 2, using MIP-pack and CP-sched, are almost indistinguishable from each other and perform significantly worse than Model 3.

Quality over Time. Figure 5.9 shows schedule quality, i.e. sum of tardiness, over time. Each point represents a time where a better quality solution was found using CP-sched for Models 1, 2, and 3. Model 0 was not included due to its smaller, two minute time limit on CP-sched. Figure 5.9 supports the hypothesis of CP-sched stagnating almost immediately after starting. There are still minor improvements in tardiness being made throughout the thirty minutes, however, these improvements appear to be negligible.

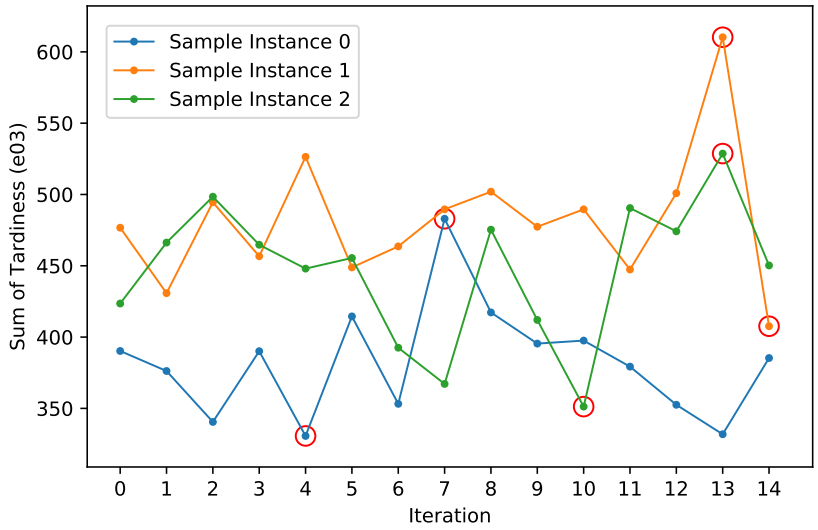


Figure 5.8: Variance of tardiness from different schedules found by Model 0 within one hour.

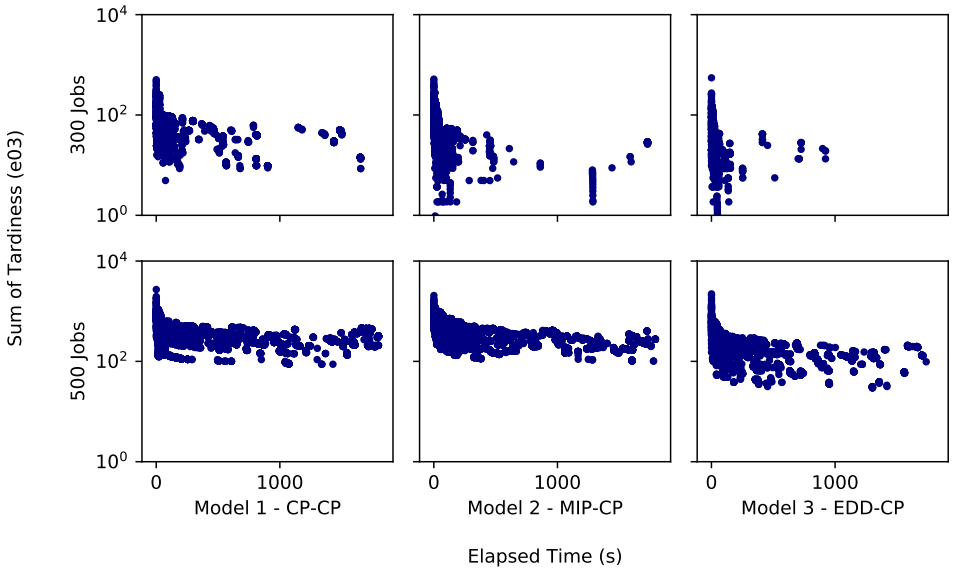


Figure 5.9: Schedule quality over time with CP-sched. Note that solutions to instances with 100 jobs were solved within a few seconds so graphs for the 100 job instances were not includes in this figure.

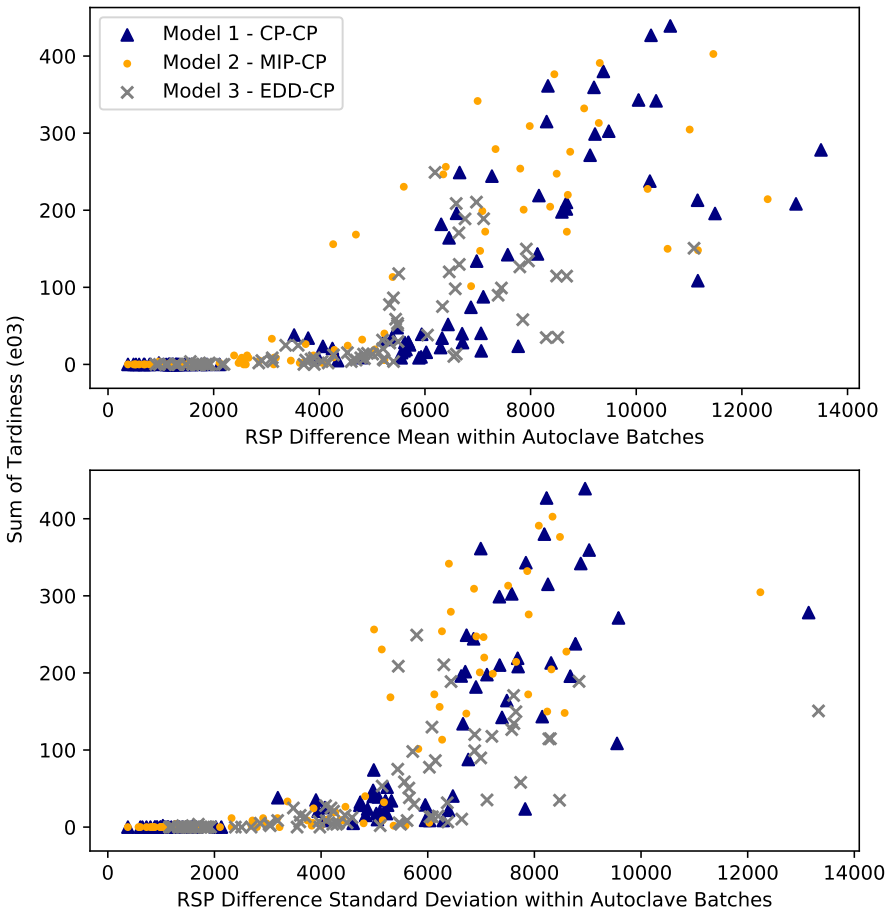


Figure 5.10: Connection between the distribution of RSP ranks within autoclave batches and sum of tardiness for Models 1, 2, and 3.

Packing Quality. We have mentioned that packing quality may be a contributing factor to schedule tardiness. One characterization of packing quality is the distribution of RSP rankings within autoclave batches. The RSP distribution of an autoclave batch is the difference between the highest and lowest job RSP rankings amongst jobs in the batch. If an autoclave batch only contains one job, then the RSP distribution is not counted in calculations. Figure 5.10 shows the distribution of RSP ranks within autoclave batches between Models 1, 2, and 3 and its impact on sum of tardiness. We can see that packing quality with respect to RSP distributions is positively correlated with schedule tardiness. EDD-pack is able to find packings with lower RSP distributions and CP-sched is subsequently able to find schedules with lower tardiness. MIP-pack and CP-pack appear to be packing batches with higher RSP distributions, which correlates with increased tardiness in schedules from CP-sched. The clusters of Model 3 solutions have a lower average and standard deviation of RSP distributions compared to Models 1 and 2.

In conclusion, we can see that both the number of autoclave batches and the quality of autoclave batches opened by a packing model contribute to eventual schedule tardiness. The instances tested in this chapter are too small for us to provide any conclusive statements, so this connection between autoclave batches and tardiness will be explored more deeply in the next chapter.

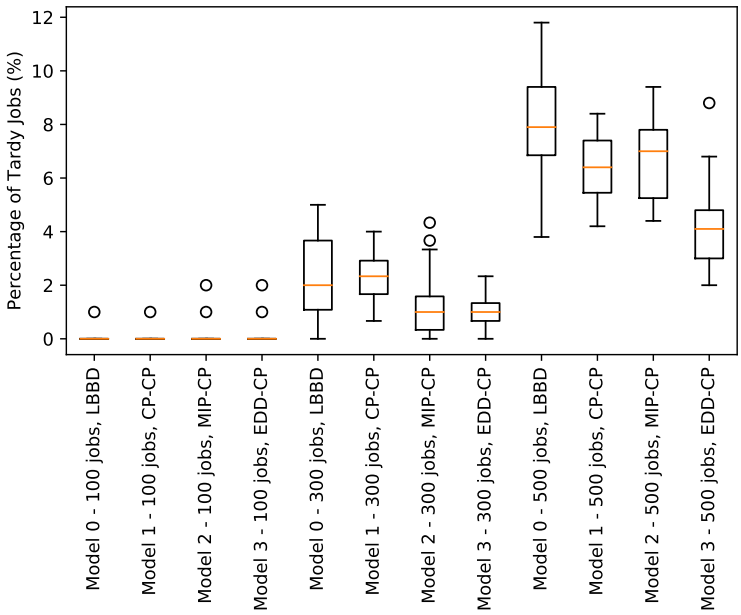


Figure 5.11: Percentage of tardy jobs per instance found by all solution techniques.

Job Tardiness. Lastly, we look at the percentage of jobs that are tardy across the different solution techniques in Figure 5.11. We can see that as the range of percentages only differs slightly between techniques, and Model 3 outperforms the other models again. The data presented in Figure 5.11 does not have a clear enough correlation such that we can predict how many jobs might be tardy given a certain instance size.

5.3 Conclusions

This chapter explored the efficacy of four solution techniques extended from models developed in Chapter 4. Each solution technique is the combination of one packing model and one scheduling model: a Logic-based Benders Decomposition (LBBD), Mixed Integer Programming packing (MIP-pack) with Constraint Programming scheduling (CP-sched), Constraint Programming packing (CP-pack) with CP-sched, an Earliest Due Date packing heuristic (EDD-pack) with CP-sched. These four approaches, along with their variables, parameters, and formulations were presented in this chapter. Each approach was tested on 3 sets of 30 instances containing 100, 300, and 500 jobs per instance, respectively. EDD-pack with CP-sched performed the best while LBBD performed the worst. MIP-pack with CP-sched performed slightly better than CP-pack with CP-sched but both were eclipsed by EDD-pack with CP-sched.

The connection between autoclave batch characteristics and schedule tardiness was explored. Data showed a strong connection between schedule tardiness and both the number of autoclave batches and the quality of autoclave batches found by packing. It was shown that the distribution of job RSP rankings within autoclave batches had a positive correlation with sum of tardiness in the eventual schedule. This correlation is further explored in the next chapter with larger problem instances. Another characterization of the results showed that the percentage of tardy jobs within a given instance does not vary

significantly between solution techniques. EDD-pack with CP-sched performs slightly better than the other techniques, with 0% to 9% of jobs being tardy for instances with 500 or fewer jobs.

Overall, we can conclude from this chapter that EDD-pack with CP-sched is the most likely candidate so far in producing high quality production-scale schedules. Although the other solution techniques were able to solve all the instances in this chapter, they exhibited worse performance and scalability than EDD-pack with CP-sched. This result is similar to what we found in Chapter 4, where the hybrid heuristic-CP approach that uses a CP scheduling model to improve a feasible EDD solution performed well and showed the greatest scalability. Thus, EDD-pack with CP-sched will be the only model from this chapter tested on production-scale instances in the next chapter.

Chapter 6

Scaling Up: Size

The instances solved in Chapter 5 are much smaller than the problems solved in practice: 4000 job instances with due dates from 1 to 4 weeks. In this chapter, we test the best solution technique from Chapter 5, an Earliest Due Date (EDD) packing heuristic with Constraint Programming (CP) scheduling on the full size instances. We also present a new packing algorithm, size-constrained clustering, and two new scheduling algorithms, a parallel scheduling algorithm and a genetic algorithm. The numerical results presented in this chapter are for problems of the same complexity and scale as real-world production instances.

6.1 Solution Approaches

After representing the full complexity of the problem in Chapter 5, our next step is to investigate solution techniques for increasingly larger problem instances. As demonstrated in the results of Chapter 5, the EDD packing algorithm consistently outperforms the Mixed Integer Programming and Constraint Programming (CP) packing models. Thus, we only test the EDD packing and CP scheduling models for scalability in this chapter. Due to the efficacy of a greedy heuristic against exact methods, we will also introduce and test heuristic algorithms for packing and scheduling.

We develop a learning-based packing technique by using techniques from the constrained clustering community. If each autoclave batch is thought of as a cluster of tool batches, size-constrained clustering [130] (SCC) can be used to enforce autoclave capacities as well as the frequency of bottom tools within clusters (autoclave batches). We designate this packing technique as *SCC-pack*.

The scheduling problem is similar to a problem within project scheduling literature, known as the *Time-Varying Resource-Constrained Project Scheduling Problem* [71]. We implement a modified version of a heuristic technique for this problem, known as the parallel scheduling scheme [14]. There has also been success in project scheduling literature in using a genetic algorithm to determine the activity list that is taken as an input by the parallel scheduling scheme [55]. Thus, we will test two new scheduling techniques, designated as *PS-sched* and *PS-GA-sched*.

Table 6.1: Overview of models and algorithms.

		Scheduling		
		Constraint Programming **(CP)**	Parallel Scheduling Scheme **(PS)**	Genetic Algorithm **(PS-GA)**
Packing	Earliest Due Date Packing Heuristic **(EDD)**	Model 3	Model 4	Model 5
	Constrained Clustering **(SCC)**	Model 6	Model 7	Model 8

Therefore, in this chapter we have two packing techniques, EDD-pack and SCC-pack, and three scheduling techniques, CP-sched, PS-sched, and PS-GA-sched. We test every combination of packing and scheduling techniques, resulting in a total of six distinct models, numbered as Models 3 to 8[1], that take an input of jobs and produce an output of scheduled batches. Table 6.1 shows an overview of these models and their techniques.

The purpose of this chapter is to compare the performance of our best solution approach so far, EDD-pack with CP-sched, against several non-exact methods from different bodies of literatures. In Chapter 5, we showed the superiority of EDD-pack with CP-sched against other combinations using exact modelling techniques, but this approach has not yet been tested against other non-exact approaches.

6.1.1 Size-Constrained Cluster Packing

SCC-pack is inspired by the KmeansS algorithm [130] for size-constrained clustering (see Section 3.2.4). Before clustering, we batch jobs into tool batches. Thus, we re-use the first section of Algorithm 5 (Section 5.1.4) to obtain a set of feasible tool batches. Each tool batch then represents one point, and an autoclave batch subsequently represents a cluster of points. The location of a point is given by the RSP rank of the point's associated tool batch (see Section 5.1.1). Due to the fact that packing is completely disjoint for each distinct autoclave cycle type, the following clustering algorithm is performed independently for each cycle type.

As with any k-means based algorithm, we need to first determine a starting value of k and obtain an initial clustering of points with k centroids. The minimum number of clusters we can have is bounded by the number of bottom tools that exist. Given a bottom tool b with quantity 1, if 10 tool batches are formed using b, we need to have at least 10 autoclave batches to feasibly pack those tool batches without using more than the existing number of bottom tools in any autoclave batch. Thus, the lower bound on the number of autoclave batch is calculated using the following equation.

$$LB = max\left(\left\{\frac{|\{k \in \mathcal{B}^1 \mid b_k = b\}|}{q_b} \mid \forall b \in \mathcal{N}\right\}\right) \tag{6.1}$$

[1]We start numbering the models at 3 to remain consistent with the numbering of models in Chapter 5, where Model 3 refers to the model with EDD-pack and CP-sched

where $k \in \mathcal{B}^1$ is the set of tool batches, b_k is the tool batch being used by tool batch k, and q_b is the available quantity of bottom tool b.

To obtain an initial clustering of points, we use the classic k-means algorithm [86] to find a clustering with k equal to the lower bound. Since k is the lower bound, all k clusters are guaranteed to be non-empty in any feasible solution. Thus, we force each cluster to contain at least one point. Once an initial clustering is obtained, we iterate through the clusters, modify centroids, and then reassign points to their new closest centroids. To satisfy the limit on the quantities of bottom tools, we reassign each point to their closest centroid with less than the available quantity of the point's bottom tool. For example, we may have a point p representing a tool batch with bottom tool b but point p's two closest clusters already have two points each with bottom tool b and there are only two units of b in inventory. Thus, point p needs to be assigned to its third closest cluster.

When a cluster c is infeasible, i.e. larger than its constrained size, there exists a collection of sets of points $\{\mathcal{P}^i \mid i = 0, ..., n\}$ where for each set $\mathcal{P}^i$, if points in the set are removed from the cluster, the cluster becomes feasible. We need to find the value i^* such that points in $\mathcal{P}^{i^*}$ are further away from the centroid of c than points in any other set $\mathcal{P}^i, i \neq i^*$. When we find our desired set of points $\mathcal{P}^{i^*}$, we need to adjust the centroid of c to move away from the points in $\mathcal{P}^{i^*}$. At each iteration, if a cluster is infeasible, the centroid needs to be modified. We stop the clustering algorithm when cluster centroids stay constant over multiple iterations. Thus, we are guaranteed one of two options: if k clusters are not enough to feasibly contain all points, we are left with an incomplete clustering and need to repeat the entire process with $k + 1$ clusters; if cluster centroids do not change over multiple iterations, the clustering is feasible with respect to all size constraints. We can always find a feasible solution as in the worst case, k is equal to the number of tool batches and each point is assigned to its own cluster.

Now, let us formally define the centroid modification procedure as presented in Algorithm 6.[2] Given a cluster (autoclave batch), an in-class point refers to any point (tool batch) within that cluster. Point coordinates refer to its tool batch's RSP rank, so the centroid of a cluster is the average of all tool batch RSP ranks assigned to the autoclave batch represented by the cluster. If a cluster contains more volume than its capacity, then the farthest points from the cluster centroid which, when removed, make the cluster feasible are labelled as the farthest in-class (FI) points. Let the set of points $\mathcal{P}^{i^*}$ be the farthest in-class (FI) points of a cluster. Each FI point is paired with a virtual point called a virtual FI (VFI) point. A VFI point is a dummy point that adds weight in the opposite direction as its FI point with respect to the cluster centroid. Thus, when we update the centroid location, we use the VFI points to move the centroid away from the cluster's FI points. If $y \in \mathcal{FI}(\mathcal{X}_h)$ is the set of FI points in cluster X_h and $x \in \mathcal{X}_h$ is the set of all points in cluster X_h, the following equation defines the VFI point z for any FI point y.

$$z = 2 \times \frac{\sum_{x \in \{\mathcal{X}_h \setminus \mathcal{FI}(\mathcal{X}_h)\}} x}{|\mathcal{X}_h \setminus \mathcal{FI}(\mathcal{X}_h)|} - y \quad \forall y \in \mathcal{FI}(\mathcal{X}_h) \tag{6.2}$$

If a cluster contains VFI points $z \in \mathcal{VFI}(\mathcal{X}_h)$, its centroid can be updated with the following equation

[2]This centroid modification procedure with equations (6.2) and (6.3) was presented by Zhang et al. [130].

Algorithm 6: SCC Packing

```
Result: Feasible packing using SCC-pack
let 𝒞 hold future clusters;
for each autoclave cycle do
    𝒫 ← points (tool batches) to be clustered;
    k ← lower bound on number of clusters (autoclave batches);
    while feasible clusterings not found do
        perform regular k-means to initialize clusters, ensure that each cluster out of k has at
         least one point;
        while iterations ≤ maxIter do
            add points from 𝒫 to their closest cluster with less than the available number of
             tools, ensure that each cluster out of k has at least one point;;
            for each cluster do
                detect FI points;
                for each FI point do
                    calculate VFI point using Equation (6.2);
                end
            end
            if no FI points detected then
                feasible clustering have been found;
                add all clusters to 𝒞;
            else
                for each cluster do
                    reset cluster centroids using Equation (6.3);
                end
            end
        end
        if iterations = maxIter and clusters not feasible then
            k ← k + 1;
        end
    end
end
return 𝒞 as packed autoclave batches;
```

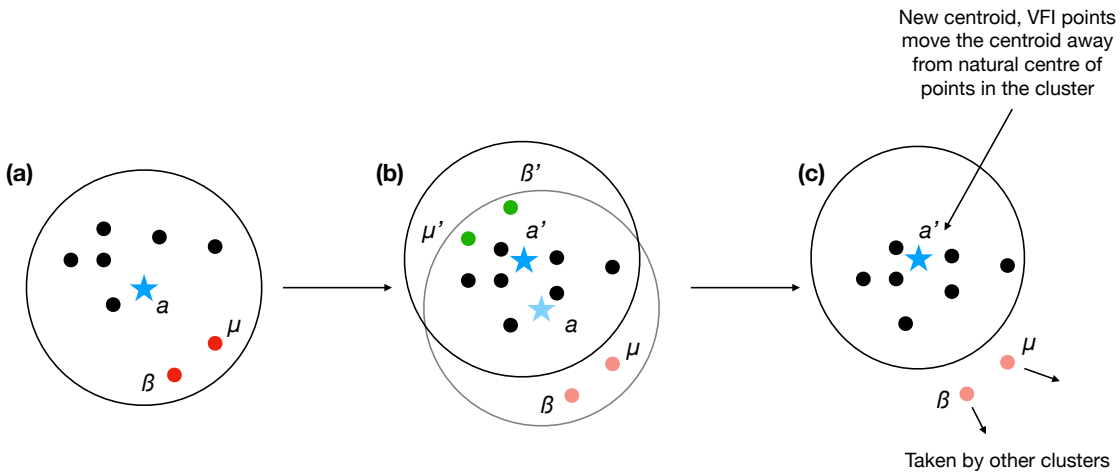


Figure 6.1: FI and VFI points in a cluster. In (a), the points β and μ are the farthest points in the cluster defined by centroid a which if removed, will make the cluster feasible. Thus, in (b), we see the VFI points $\mu^{'}$ and $\beta^{'}$ as calculated by Equation (6.2) are on the other side of the cluster as the original FI points. Thus, when the centroid is updated with Equation (6.3), the centroid moves towards the VFI points and away from the FI points. Then, we see that in (c), the two original FI points are now closer to other cluster centroids and have thus been removed from their original cluster. Then, the cluster with new centroid $a^{'}$ is now feasible.

where γ is the penalty factor for modifying cluster centroids. Note that even though the set of VFI points $z \in \mathcal{VFI}$ is calculated by finding the VFI point for every FI point $y \in \mathcal{FI}(\mathcal{X}_h)$, we do not care which VFI point is associated with which FI point.

$$\mu_h = \frac{\sum_{x \in \mathcal{X}_h} x + \gamma \sum_{z \in \mathcal{VFI}(\mathcal{X}_h)} z}{|\mathcal{X}_h| + \gamma |\mathcal{VFI}(\mathcal{X}_h)|} \tag{6.3}$$

Figure 6.1 shows how FI and VFI points are detected and calculated as well as how they contribute to updating the cluster centroid.

Zhang et al. [130] detected FI points by cluster size in terms of number of points in a cluster. However, we need to introduce new detection criteria as our clusters, representing autoclave batches, are not constrained by the number of points but by the total size of the points added together. Thus, for each cluster, we calculate the sum of point sizes (i.e. sum of tool batch sizes) and determine if it is larger than the autoclave capacity of the current cycle type. For example, if the autoclave capacity is 1000 and the sum of point sizes is 1200, then the farthest points from the centroid that add up to at least 200 are labelled as FI points. Then, we use Equation 6.2 to calculate the VFI points associated with our labelled FI points and Equation 6.3 to update the centroid location.

This process of finding FI/VFI points and updating centroids is repeated until either no clusters have FI points or we reach a threshold of iterations. If the maximum number of iterations is reached without finding a solution with only feasible clusters, we increase k by one and repeat the process. In the worst case scenario, we assign one point (tool batch) to one cluster (autoclave batch), so this algorithm will always converge.

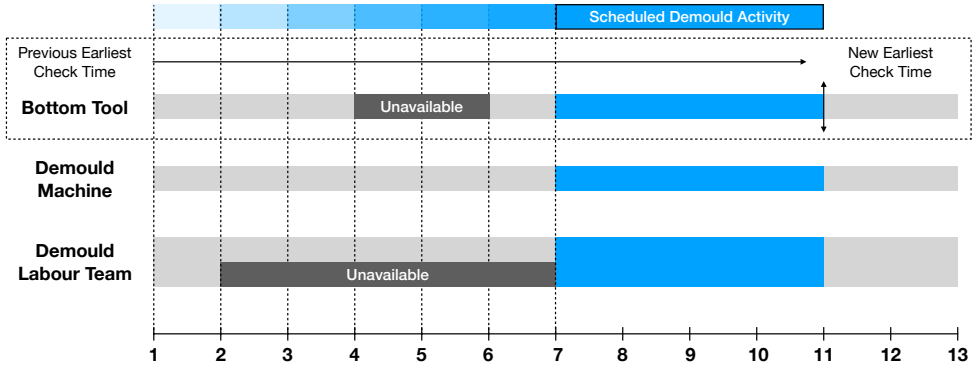


Figure 6.2: Checking and scheduling activities on resource horizons. When the demould activity is scheduled, the earliest time for which a future activity is scheduled on this bottom tool resource is moved to the end of the scheduled activity.

6.1.2 Parallel Scheduling

The PS-sched algorithm is inspired by the parallel scheduling scheme from project scheduling literature [34] (see Section 3.2.3). Similar to the interval variables used in CP-sched, we define an activity element that corresponds to a batch being processed in one stage. Thus, each tool batch has three activities for tool preparation, layup, and demould, respectively, and each autoclave batch has one activity for curing. We know the processing time of these activities and the required resources from the packing decisions.

First, we sort the set of autoclave batches by RSP rank, where an autoclave batch's RSP rank is the average of its jobs' RSP ranks. In the classic parallel scheduling scheme, at the start of each iteration, all activities with fulfilled precedence are added to the queue and scheduled with a certain priority. Then, the schedule horizon is updated to the earliest finish time of the recently scheduled activities. The PS-sched algorithm is motivated by the idea of choosing a set of activities then scheduling them without regard for activities not in the set.

Thus, at each iteration, we randomly select an autoclave batch i from the first L items of $\mathcal{B}^2$. We schedule all tool preparation, layup, curing, and demould activities for that autoclave batch without considering activities from other batches. Because there is precedence between stages, we have to create two lists. One to initially hold all tool prep activities associated with tool batches in i, and the second to hold the remaining layup, curing, and demould activities. For each activity in the first list, we find the earliest time that the activity can be scheduled while taking into account all of its required resources and schedule the activity to start at that time. Figure 6.2 clarifies this check and schedule procedure. Then, we look through the second list and add any activities with now fulfilled precedence to the first list. These actions are repeated until all activities for autoclave batch i have been scheduled and we reach the end of one iteration. We repeat this iterative process until the activities for all autoclave batches have been scheduled. Algorithm 7 formally describes this scheduling procedure.

Algorithm 7: PS Scheduling

```
Result: Feasible schedule using PS-sched
given a set of packed autoclave batches B^2;
for each repetition do
    reset all horizons of bottom tools, machines, and labour teams used by any batch;
    let I contain activities that are able to be scheduled;
    let A contain successors of activities in I;
    sort B^2 by RSP rank;
    while B^2 not empty do
        i ← randomly selected autoclave batch from first L items of B^2;
        add all tool preparation activities to I;
        add all layup, curing, and demould activities to A;
        remove i from B^2;
        while I not empty do
            a ← randomly selected activity from first L items of I;
            e ← earliest time where all resource horizons used by a is available;
            while a is not scheduled do
                if associated horizons of a can be scheduled from e then
                    schedule a with start time of e;
                    if a is demould activity then
                        update earliest start times of all bottom tool horizons associated with i to
                         end of a;
                    end
                end
                e ← next time period;
            end
            move any activities from A with fulfilled precedence to I;
        end
    end
    add scheduling solution to solution list;
end
pick best scheduling solution from solution list;
```

6.1.3 Genetic Algorithm with Parallel Scheduling

The order in which autoclave batches are schedulled has a large impact on the resulting schedule tardiness as resource horizons get pushed back with every autoclave batch that is scheduled. Therefore, once an autoclave batch is fully scheduled, any resource used by that batch can now only be used after the end of the batch. A genetic algorithm can therefore be used to try and find better autoclave batch orderings through merging the best orderings together over several generations.

The approach used by PS-GA-sched is inspired by a genetic algorithm introduced by Hartmann [55] (see Section 3.2.3) that uses a permutation based representation of activities, referred to as activity lists. Within the context of the CMP, an activity list is an ordered list of autoclave batches. Each individual within the population of PS-GA-sched is one activity list, thus, each gene refers to one autoclave batch. Individuals in PS-GA-sched are scheduled using Algorithm 7 without randomization in selecting autoclave batches; batches are scheduled in the exact order that they appear in the activity list.

First, we randomly generate a set of individuals (activity lists) to create the initial population. When generating an individual, we start with an empty activity list. Then, we randomly select an autoclave batch from all autoclave batches that have not yet been added to the activity list and append the selected batch to the end of the activity list. We repeat this selection process until all autoclave batches have been added to the activity list. Then, we apply PS-sched to obtain the fitness of the newly created activity list (sum of job tardiness) and add the new individual into the population. After n individuals have been created in this manner, we have the starting population and can begin the reproduction process.

For each generation, we sort the individuals in the population by decreasing fitness, where a fitter individual has a lower sum of job tardiness than a less fit individual. We keep the first n individuals in the population and discard the rest. From the remaining n individuals, we randomly create pairs of individuals for reproduction. Each pair of individuals, x and y, produce two new individuals, a and b, using the two-point crossover method. In two-point crossover, we randomly draw two unique integers α and β, where $0 \leq \alpha < \beta < n$. The activity list of a consists of combining x's activity list subsequence from indices $i = 0, ..., \alpha$, with y's activity list subsequence from indices $i = \alpha + 1, ..., \beta$, and x's activity list subsequence from indices $i = \beta + 1, ..., n$. The activity list of b consists of combining y's activity list subsequence from indices $i = 0, ..., \alpha$, with x's activity list subsequence from indices $i = \alpha + 1, ..., \beta$, and y's activity list subsequence from indices $i = \beta + 1, ..., n$. To account for possible repetitions and missing batches in the children's activity lists, we add a repair step after both children have been produced. First, any repeated autoclave batches are removed, and second, any missing batches are appended to the end of the children's activity lists.

There is also a certain probability of mutation for newly reproduced individuals, where two genes (autoclave batches) are randomly selected and swapped with probability m. Finally, each child is scheduled using PS-sched and added to the population. After g generations have passed, the fittest individual is kept as the best solution. Algorithm 8 formally describes this genetic algorithm.

Algorithm 8: PS-GA Scheduling

```
Result: Feasible schedule using PS-GA-sched
n ← number of individuals in population;
I ← randomly initialized population (each individual is a unique autoclave batch ordering);
schedule each individual in population using PS-sched;
g ← number of generations;
for 1 to g do
    sort I by fitness (tardiness);
    keep best n individuals from I and discard the rest;
    for each randomly chosen pair of individuals from I do
        use two-point crossover to produce two children;
        remove repeated autoclave batches and append missing autoclave batches;
        for each child do
            if a Unif(0,1) ≤ m then
                swap two randomly chosen genes in child;
            end
        end
        add children to I schedule each child using PS-sched;
    end
end
keep fittest individual from I;
```

6.2 Numerical Results

The six solution techniques presented in this chapter were tested on 4 sets of randomly generated 30 problem instances with 1000, 2000, 3000, and 4000 jobs per instance, respectively. Refer to Section 2.2.1 for the instance generation process. All models and algorithms were implemented in Java and CP-sched was implemented using CPLEX Optimization Studio 12.9. EDD-pack and PS-sched were repeatedly executed 100 times for each instance, keeping the best solution. Runtime for EDD-pack and PS-sched includes all 100 runs. SCC-pack was given a maximum iteration threshold of 400. PS-GA-sched was parametrized with 20 individuals per population, a 25 generation limit, and a mutation probability of 5%. CP-sched was given a thirty minute time limit.

Overall Comparison of Solution Approaches. All six solution techniques were able to find feasible solutions for all instances. Figure 6.3 compares the average sum of tardiness found by the different techniques. Models 3 and 6 both use CP-sched for the scheduling component and they also produce, on average, the best quality schedules with respect to average tardiness. Models 3, 4, and 5 (using EDD-pack) produce better quality schedules than their counterparts in Models 6, 7, and 8 (using SCC-pack), respectively. Thus, we can conclude that EDD-pack and CP-sched are still the best choices for packing and scheduling.

Models 5 and 8, using PS-GA-sched, performed better than Models 4 and 7, using PS-sched. Thus, the genetic algorithm presented in Algorithm 8 did improve the performance of PS-sched over 25 generations. However, PS-GA-sched uses much more time than CP-sched to achieve the same solution qualities and CP-sched finds better schedules within thirty minutes. Models 4 and 7 have the worst performance, implying that pushing back the earliest start time after every demould activity is removing a non-negligible amount of schedule time that could have accommodated unscheduled batches. On the

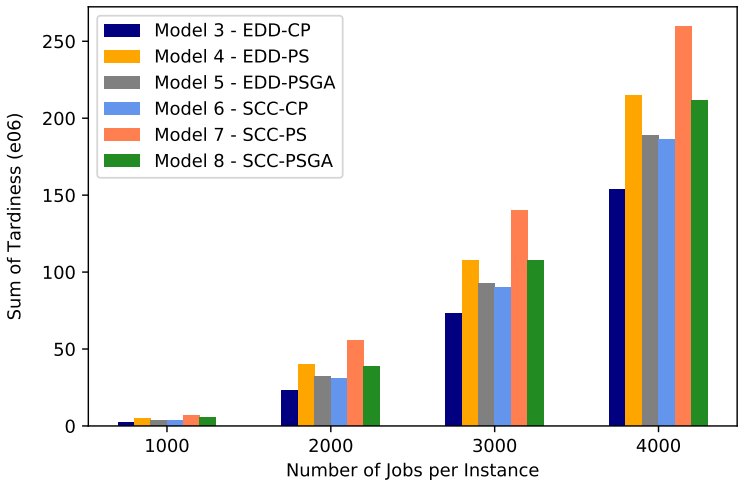


Figure 6.3: Comparison of average sum of tardiness between solution techniques.

other hand, CP-sched is constantly shuffling interval variables around and trying to determine if intervals could be feasibly moved to an earlier time.

As in Chapter 5, we plot the connection between the number of autoclave batches opened and schedule tardiness in Figure 6.4. There is a strong correlation between the number of batches and schedule tardiness, however, there is still a non-negligible discrepancy in tardiness between models, implying that the number of batches is not the only factor that impacts tardiness. For example, looking at Models 3 and 6 in Figure 6.4, we can see that both models find solutions with around the same number of autoclave batches, but CP-sched (Model 3) finds better solutions. This pattern repeats for the other two pairs of models where each pair uses the same scheduling model, Model 4 vs. Model 7 and Model 5 vs. Model 8. These patterns imply that packing quality is continuing to contribute to tardiness in larger problem instances, extending the observations we made for Figure 5.7.

Packing Quality. Figure 6.5 shows the connection between RSP distributions within autoclave batches and tardiness.

We can see that there is a fairly consistent trend in the standard deviation of RSP distribution amongst autoclave batches within a certain instance against tardiness. Thus, we can conclude that the standard deviation of RSP distributions is another positive correlating factor to tardiness. However, the relationship between the mean RSP distribution and tardiness shows two distinct trends, each correlating to one type of packing model. SCC-pack appears to be packing batches with a lower mean RSP distribution but achieving higher tardiness. We can further explore this discrepancy by looking at how balanced the autoclave batches are with respect to unfilled capacity.

Figure 6.6 shows how unfilled autoclave capacity contributes to tardiness. Similar to Figure 6.5's relationship between mean RSP distribution and tardiness, we can see two distinct trends in Figure 6.6. If uf_i is the unfilled capacity of autoclave batch i, calculated by $uf_i = v_i - \sum_{k \in \{\mathcal{B}^1 | k \text{ assigned to } i\}} \sigma_k$ (See Table 5.4 for notation), then we define the balance of autoclave batches in an instance as equal to the

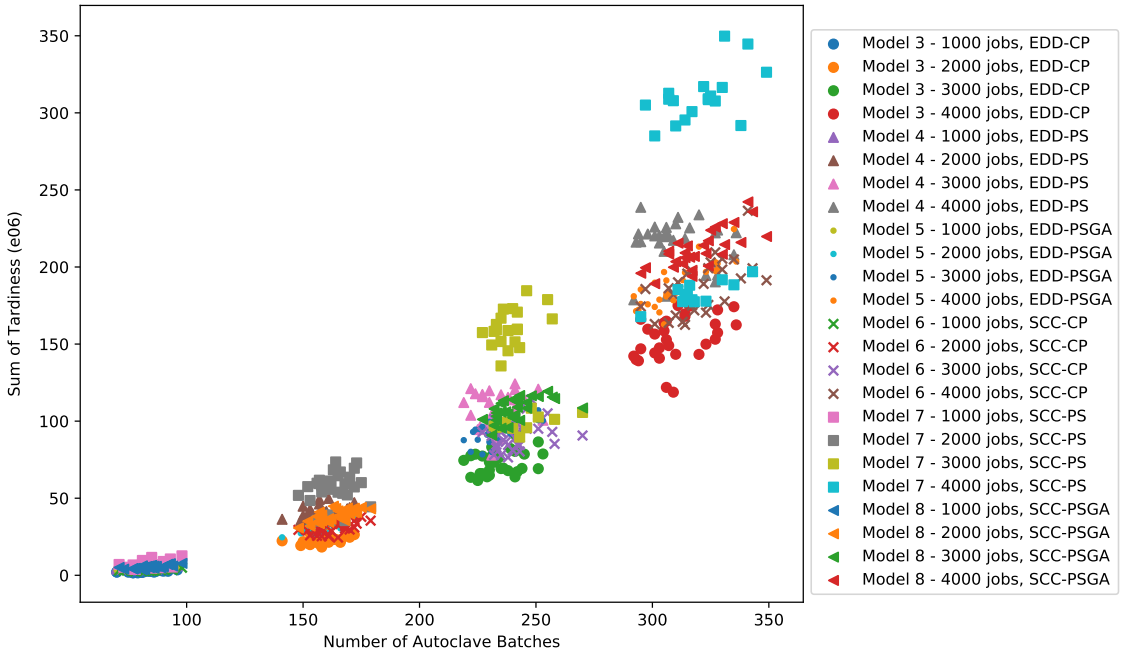


Figure 6.4: Connection between number of autoclave batches and sum of tardiness.

standard deviation of the set $\{uf_i \mid i \in \mathcal{B}^2\}$. The EDD-pack models have a higher standard deviation of unfilled capacity compared to the SCC-pack models, meaning that SCC-pack finds much more balanced batches than EDD-pack. However, it appears that balanced batches are actually undesirable compared to imbalanced batches. This result corroborates the trend in Figure 6.5, Models 3, 4, and 5 find more imbalanced batches with a higher mean RSP distribution as more jobs in a batch means a higher difference between the minimum and maximum RSP ranks. Thus, another measure of packing quality is the balance of jobs across batches. This measure can be considered a weak negative correlating factor to tardiness, where a more balanced packing is likely to increase tardiness. One possible explanation for this trend is that imbalanced batches might utilize resources more efficiently. The larger variation in autoclave batch sizes means we have some autoclave batches that take up longer periods in the schedule and some that take up shorter periods. Thus, we may be able to fill gaps in the schedule better because we have a selection of short and long batches to choose from.

In conclusion, we have extracted two measures of packing quality: the standard deviation of RSP distributions within autoclave batches and the balance of the autoclave batches. The measures have a stronger positive correlation and a weak negative correlation with tardiness, respectively. Including the very strong positive correlation of the number of autoclave batches to tardiness, we now have three predictors of tardiness given a packing solution.

We can perform a three-way analysis of variance (ANOVA) test on the effects of these three predictors on tardiness. To control for the effects of different scheduling models, we only use experimental results from Model 3 and Model 6, both of which use CP-sched. Let us refer to the standard deviation of RSP distributions amongst autoclave batches as the RSP range of an instance, and let us refer to the standard deviation of unfilled capacity amongst autoclave batches as the balance of an instance. Table

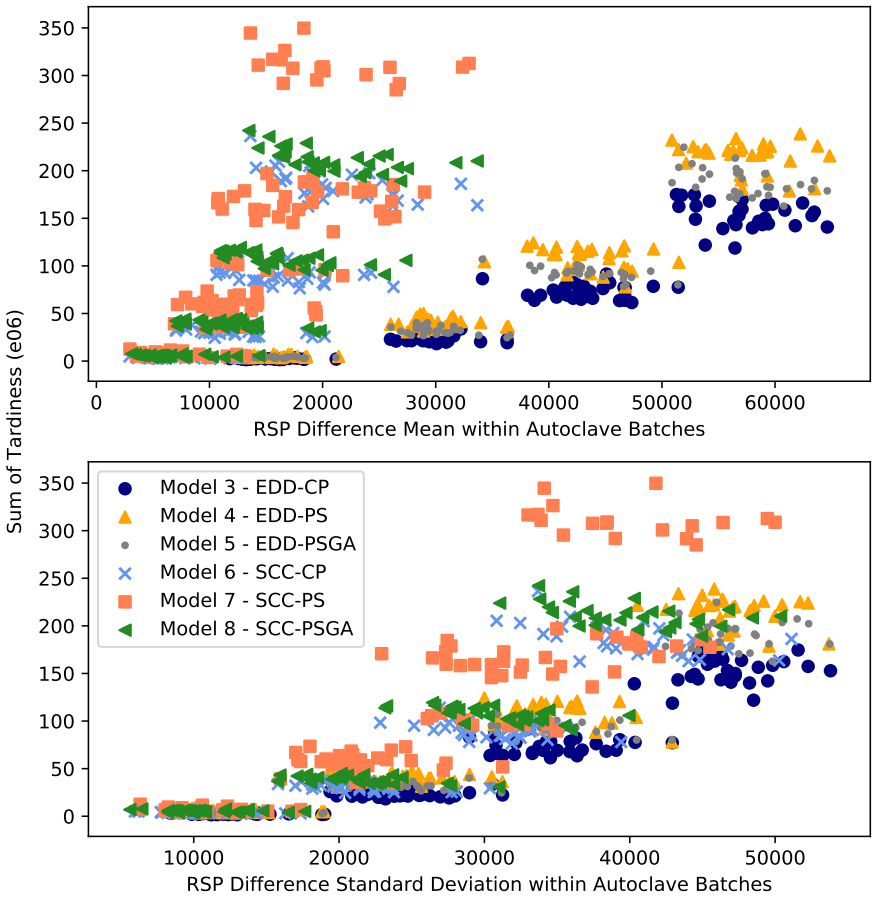


Figure 6.5: Connection between RSP distribution within autoclave batches and sum of tardiness.

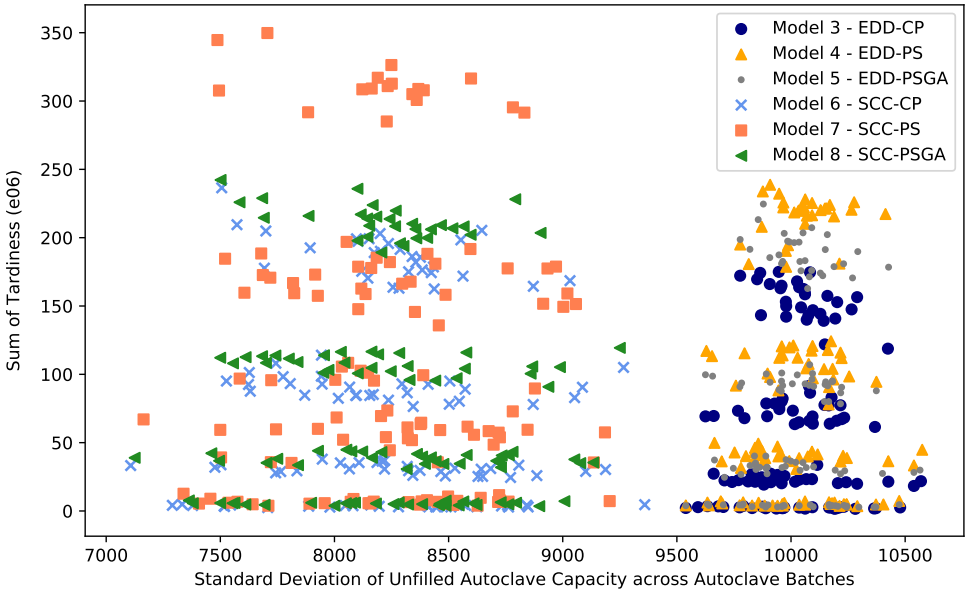


Figure 6.6: Connection between packing balance and sum of tardiness.

Table 6.2: ANOVA parameters for analysing the effects of packing on tardiness.

Factor Level	Number of Autoclave Batches	RSP Range	Balance
1	Low (70 - 163)	Low (5610 - 21678)	Low (7107 - 8263)
2	Medium (163 - 256)	Medium (21678 - 37745)	Medium (8263 - 9419)
3	High (256 - 349)	High (37745 - 53813)	High (9419 - 10575)

Table 6.3: ANOVA results for analysing the effects of packing on tardiness.

Factor	Degrees of Freedom	F-Value	PR(>F)
Number of Autoclave Batches	2	679.78	8.34e-97
RSP Range	1	73.30	1.67e-15
Balance	2	15.18	6.47e-07
RSP Range * Balance	2	4.49	1.22e-02
Number of Autoclave Batches * Balance	4	0.42	0.79

6.2 shows an overview of the ANOVA parameters and the range of values contained in each level. We determined that there is very strong multi-collinearity between the number of autoclave batches and the RSP range. Violating the assumption of no multi-collinearity means that we should not test the interaction between these two factors as a source of variation. Thus, we tested sources of variation from each of the factors individually, as well as from the interaction between RSP range and balance and the interaction between the number of autoclave batches and balance.

The ANOVA results are summarized in Table 6.3, tests for which the null hypothesis is accepted, proving that the factor level does *not* significantly affect tardiness, are in gray. There is a clear hierarchy in the level of correlation, the number of autoclave batches has the highest impact on tardiness, followed by RSP range, then balance. The interaction between RSP range and balance has a slight effect on tardiness, while we reject the null hypothesis for the interaction between the number of autoclave batches and balance. In conclusion, the three predictors individually contribute almost all of the variance, thus proving their correlations with tardiness.

Quality over Time. Figure 6.7 shows schedule quality over time. We can see that increasing the number of jobs per instance does not change the trend we saw in Figure 5.9; almost all of the major increases in schedule quality happen within the first few minutes of solve time. There are small improvements made throughout the entire thirty minutes, but are almost negligible in comparison to the initial improvement.

6.3 Bottleneck Analysis

We have characterized the impact of packing on tardiness via three correlating factors. In this section, we examine resource utilization.

Figure 6.8 shows the 25-percentile to 75-percentile of job idle times. The idle time of a job between stages is calculated by taking the difference between the job's start time in the succeeding stage and the

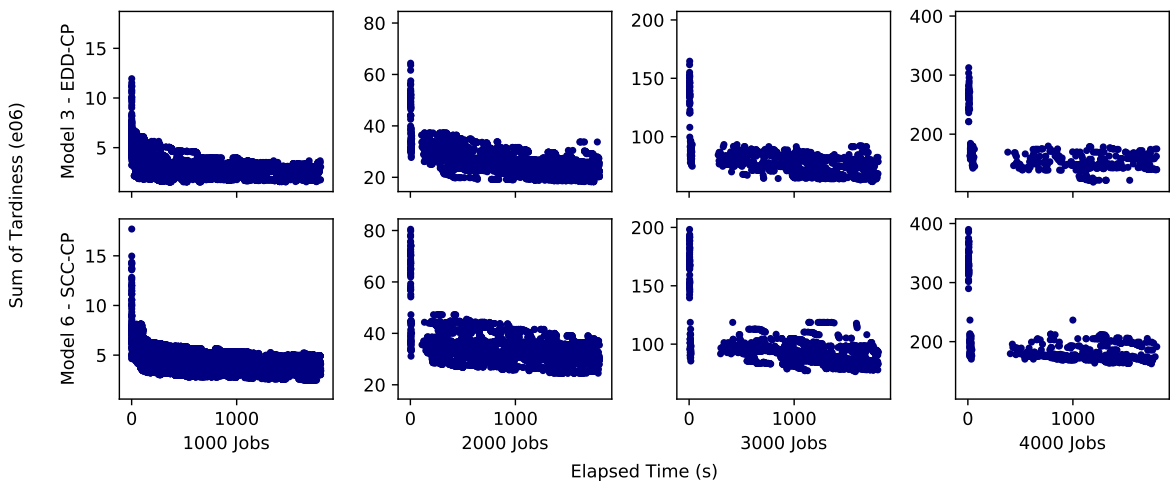


Figure 6.7: Schedule quality over time with CP-sched.

job's end time in the preceding stage. The range of idle times from tool preparation to layup and from curing to demould are quite small, especially in comparison to the range of idle times from layup to curing. A likely explanation for this discrepancy is that layup processing times are usually several times longer than curing times, and up to tens of times longer than tool preparation or demould times. We can see that the Model 3 graphs in Figure 6.8 show the least amount of idle time of all models, which corresponds with Model 3's better performance compared to other approaches. Ultimately, Figure 6.8 implies that layup or curing resources may be a bottleneck in the schedule.

However, Figure 6.8 does not distinguish between the lag due to jobs having to wait for other jobs in the same batch to finish in layup vs. the lag due to jobs having to wait for their autoclave to become available. To make sure that the lag we are seeing from Figure 6.8 is in fact being caused by the layup stage, we need to separate these two contributors to idle time into the curing lag and the layup lag. The curing lag is calculated by taking the difference between the latest end time of a job in layup within an autoclave batch and the start of the autoclave batch in curing. The layup lag is calculated by subtracting the curing lag from the difference between the end time of a job and the start of its autoclave batch in curing. Layup lag points show the maximum, average, and minimum lag across jobs within a single autoclave batch between layup and curing. Autoclave lag points show the maximum, average, and minimum lag across autoclave batches within a single instance. These lag points are plotted in Figure 6.9. We can see that the average layup lag is higher than even the maximum autoclave lag, implying that layup resources form a bottleneck in the schedule.

To determine how much of a bottleneck the layup stage is, we can calculate the propagated tardiness at the end of tool preparation, the first stage. We define propagated tardiness as the difference between the end time of a job in tool preparation and a back-propagated due date for the job in the tool preparation stage. Given a job's real due date at the end of demould, we subtract its processing times in demould, curing, and layup from the real due date to get an estimated due date at the end of tool preparation. If a job's estimated tardiness at the end of tool preparation is much smaller than the its real tardiness, then the bottleneck in layup is most likely the main bottleneck in the system. On

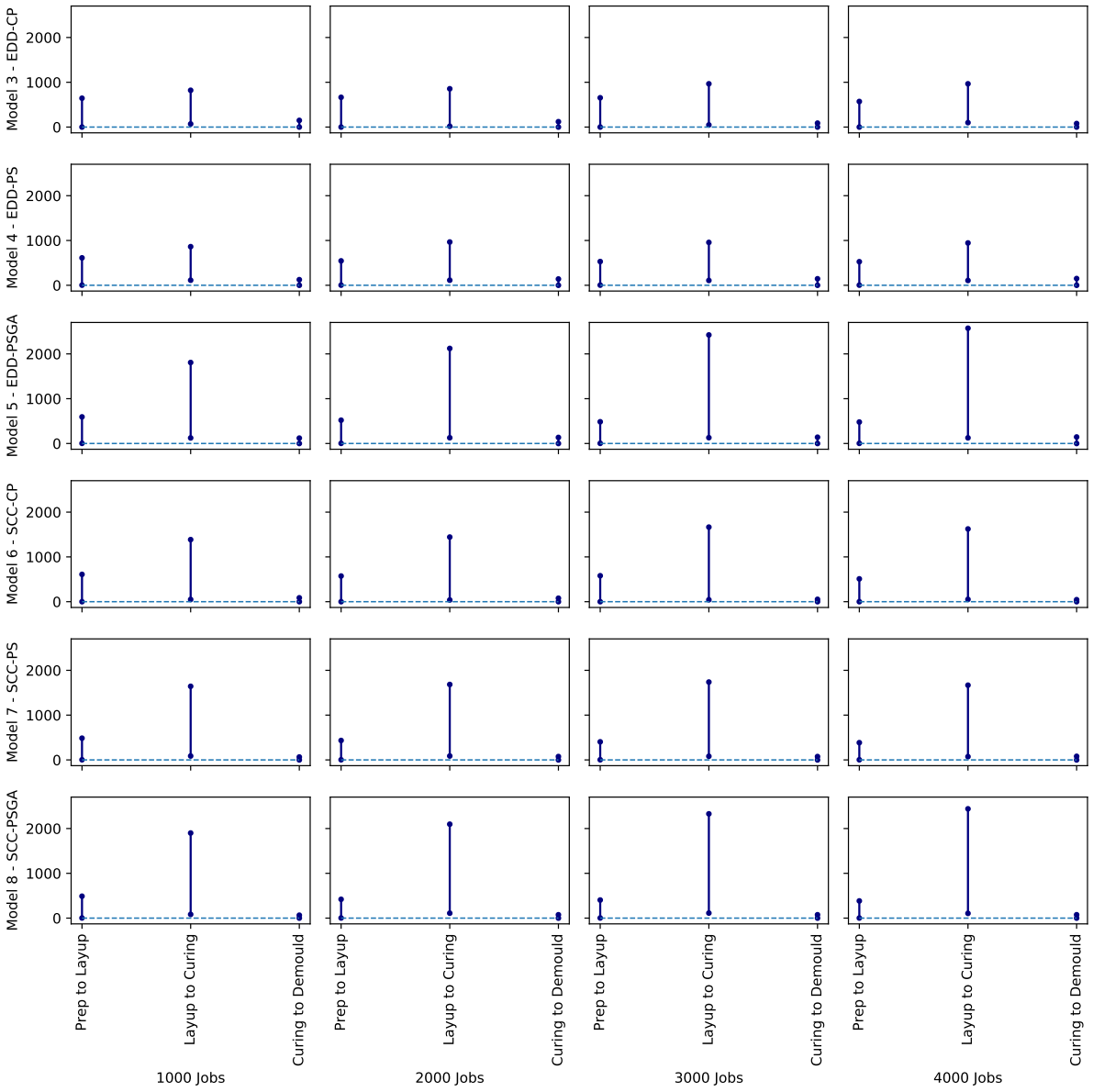


Figure 6.8: Average idle time between stages, error bars show 25-percentile to 75-percentile of idle times.

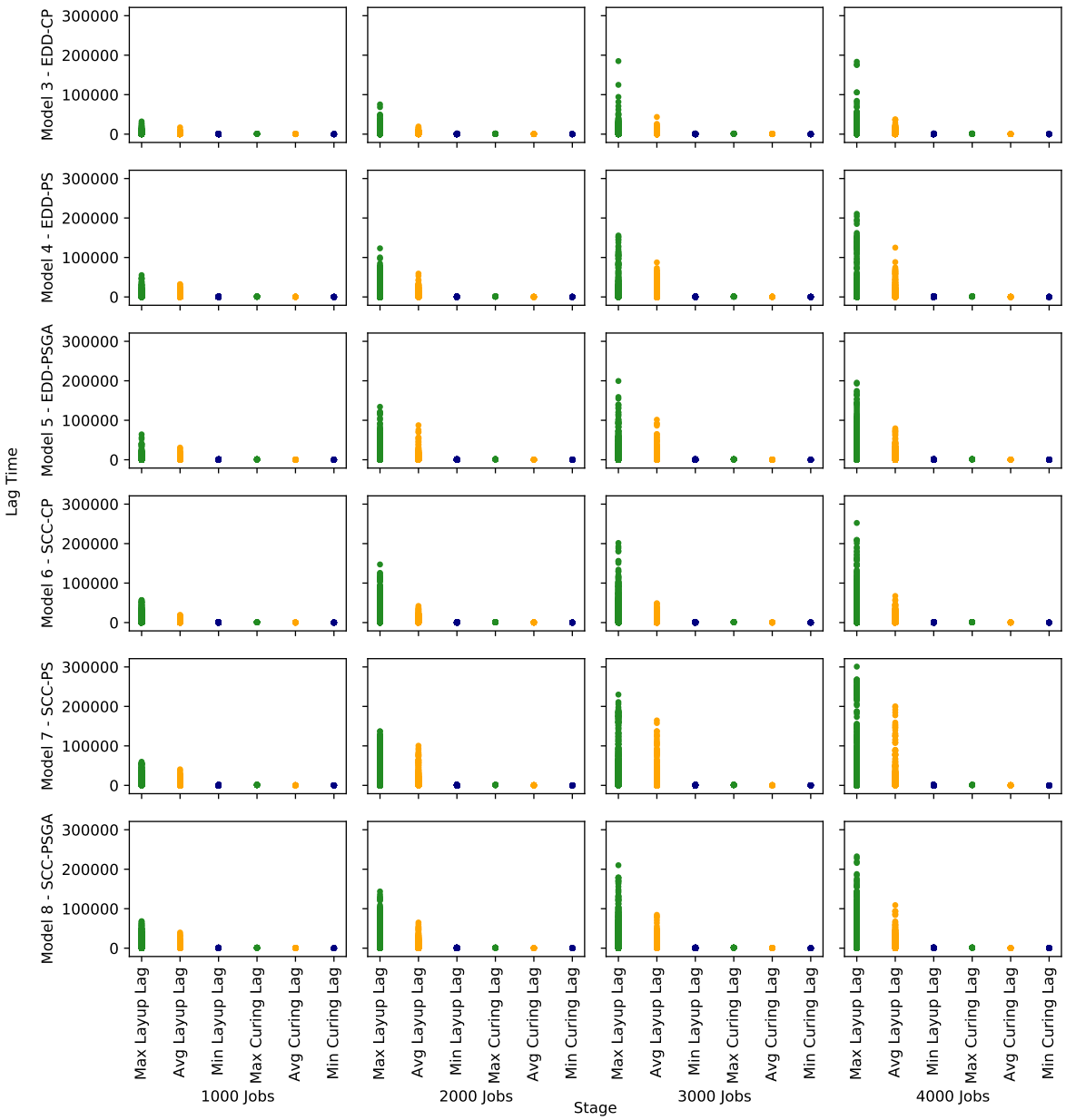


Figure 6.9: Detailed breakdown of the sources of idle time between layup and curing.

the other hand, if a job's propagated tardiness is close to its real tardiness, there is most likely another bottleneck with a larger impact than the layup stage. We can see from Figure 6.10 that the latter is true, implying the existence of a tighter bottleneck. This conclusion is further supported by Figure 6.11 which shows that tool preparation machines and labour teams have very low usage ratios over the entire schedule. These low ratios imply that the small difference between a job's propagated and real tardiness cannot be explained by a lack of resources in tool preparation.

The next step in our bottleneck analysis is to analyze tool usage. Tools are stage-independent resources, so we can calculate the usage of tools over the entire schedule. If any tools have close to 100% utilization, those tools would create a much bigger bottleneck than any stage-dependent resources. First, Figure 6.12 shows the distribution of tool usage across batches, we see that a few tools are much more popular than the rest. Let us take the five most popular tools and calculate their utilization, shown in Figure 6.13. The two most popular tools have extremely high usage ratios, meaning that these tools are almost constantly in use. When there is little idle time in the schedule where these tools are unused, it is very hard to improve the schedule as there is not enough room to push activities earlier in the schedule. We are left with a large bottleneck that cannot be removed unless the distribution from which instances are generated changes and we no longer have a few tools that are so popular. This result, combined with the Model 3 graphs from Figure 6.8, imply that the schedules found with Model 3 are reasonably close to optimal.[3] If this conclusion is true, then we can greatly improve solutions to the CMP with better system design or an increase in the capacity of bottleneck resources.

In conclusion, we found two bottlenecks inherent to the availability of resources: a small bottleneck due to layup resources (machines and labour teams) and a much larger bottleneck due to the popularity of a few tools causing almost 100% utilization of those tools.

Job Tardiness. Figure 6.14 shows the percentage of tardy jobs within a schedule. Looking at Model 3, the best performing model, we can see that at 4000 jobs, the percentage of tardy jobs ranges between 50% and 60%. These percentages are fairly high, but the previous results in Figures 6.7, 6.8, and 6.13 imply that unless the distribution of jobs from which instances are generated changes or the capacities of bottleneck resources change, 50% is most likely the lower bound on the percentage of tardy jobs in a 4000 job schedule.

6.4 Conclusions

This chapter tested six solution techniques made up of two packing approaches and three scheduling approaches on real-world sized instances, thus completing the scaling up of both complexity and size. One packing model, an Earliest Due Date packing heuristic (EDD-pack), and one scheduling model, a Constraint Programming scheduling model (CP-sched), were presented in Chapter 5. One more packing model, a size-constrained clustering algorithm (SCC-pack), and two more scheduling models, a parallel scheduling algorithm (PS-sched) and a genetic algorithm (PS-GA-sched) both inspired by project scheduling literature, were presented in this chapter. Each approach was tested on 4 sets of 30 instances

[3]Note that this claim of optimality is conditional on the exact process parameters our instances were based on. If the proportion of tools or the availability of machines were to change, the system bottlenecks may change and our approaches may be able to find better schedules.

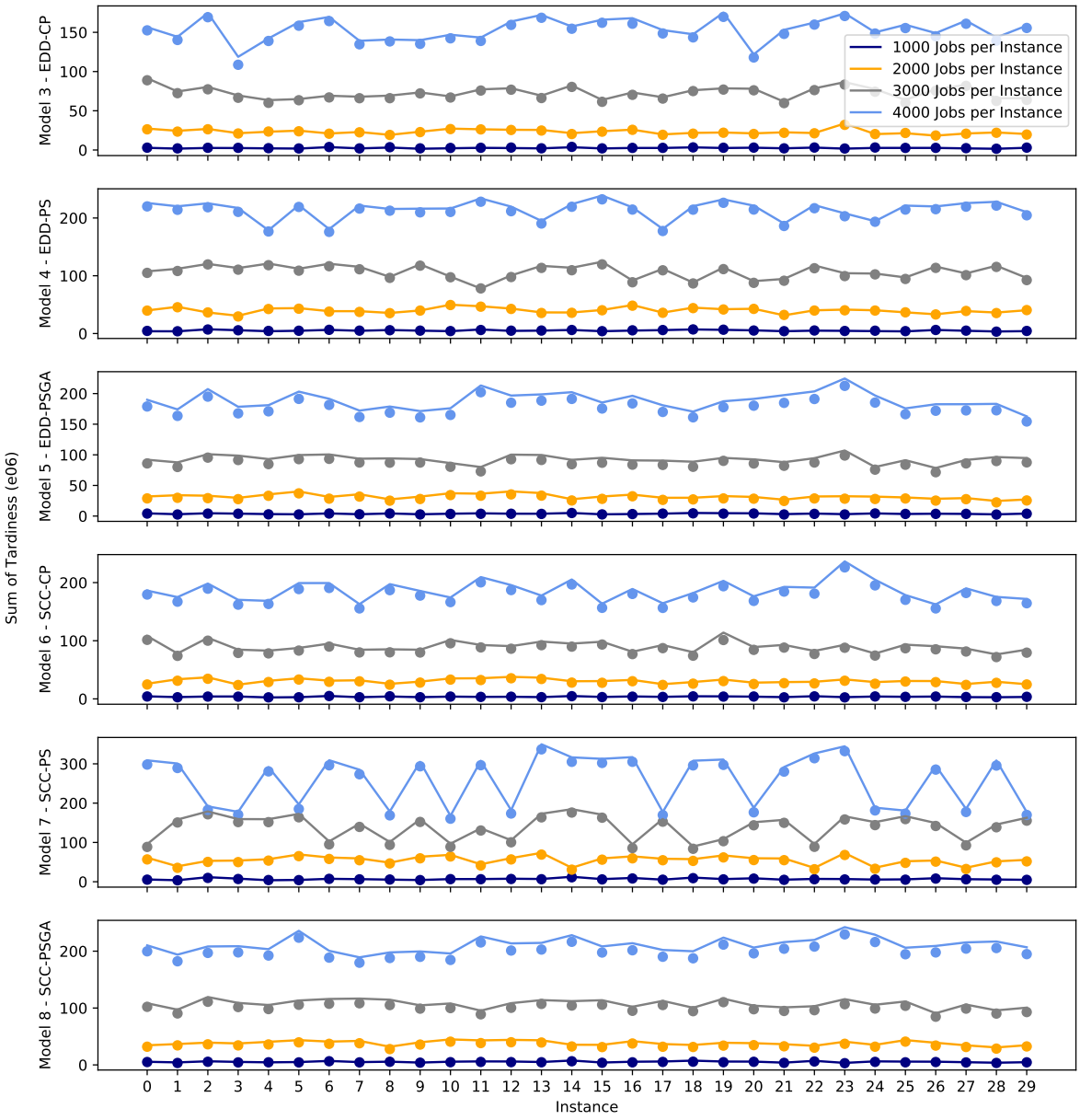


Figure 6.10: Propagated tardiness at the end of the tool preparation stage. Due dates are back-propagated through the stages and propagated tardiness is calculated by taking the difference between the end of the job in tool preparation and its propagated tool preparation due date. Points show propagated tardiness and lines show the actual tardiness calculated at the end of demould.

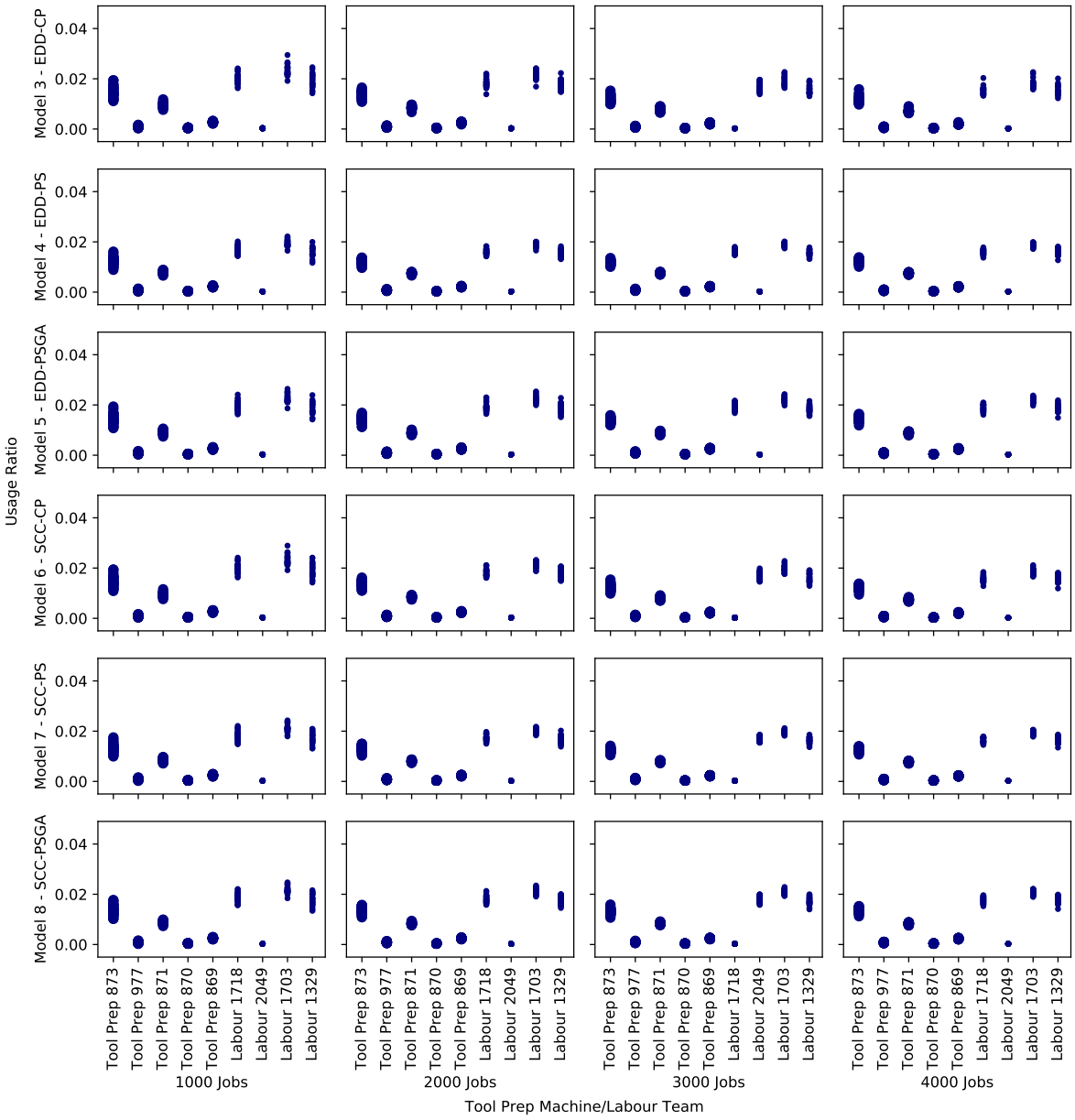


Figure 6.11: Usage ratios of tool preparation machines and labour teams.

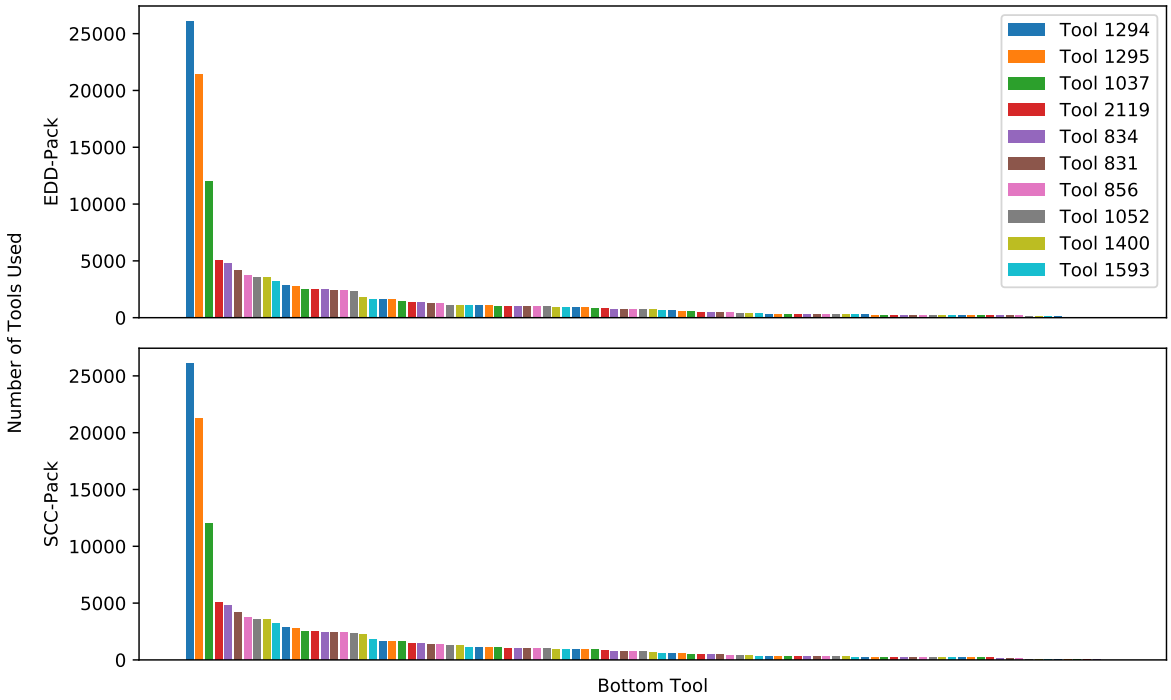


Figure 6.12: Tool popularity across batches for EDD-pack and SCC-pack.

containing 1000, 2000, 3000, and 4000 jobs per instances, respectively. EDD-pack with CP-sched performed the best again while SCC-pack with PS-sched performed the worst. The genetic algorithm in PS-GA-sched managed to improve the performance of PS-sched over 25 generations, however, still does not reach the same level of performance as CP-sched.

We extracted three predictors for schedule tardiness based on packing decisions and statistical tests confirm that these predictors significantly affect tardiness. The number of opened autoclave batches has a positive correlation with tardiness. There is also a positive correlation between the standard deviation of RSP distributions within autoclave batches and tardiness. Lastly, there is a negative correlation between the balance of jobs across autoclave batches and tardiness. These three predictors would be very useful in any continuation of model development for this problem. Ideally, we would develop future models that take advantage of these predictors and their correlations to, for example, pack batches with all three correlating factors included in the objective function.

Next, we discovered two bottlenecks inherent in the scheduling resources. Out of the stage-specific resources, i.e. machines and labour teams, the layup resources form a small bottleneck. Jobs oftentimes have to wait after layup for other jobs in the same autoclave batch to finish processing in layup before they can all proceed to be cured in autoclaves. Thus, if we can increase the available quantity of layup resources, we may be able to mitigate this bottleneck. A large bottleneck comes from the most popular tools that are shown to have almost 100% utilization. Thus, there is little idle space within the schedule for any tool batches using these tools, which makes improving the schedule very difficult and implies that our best solutions to the CMP are reasonably close to optimal. We end up with a likely lower

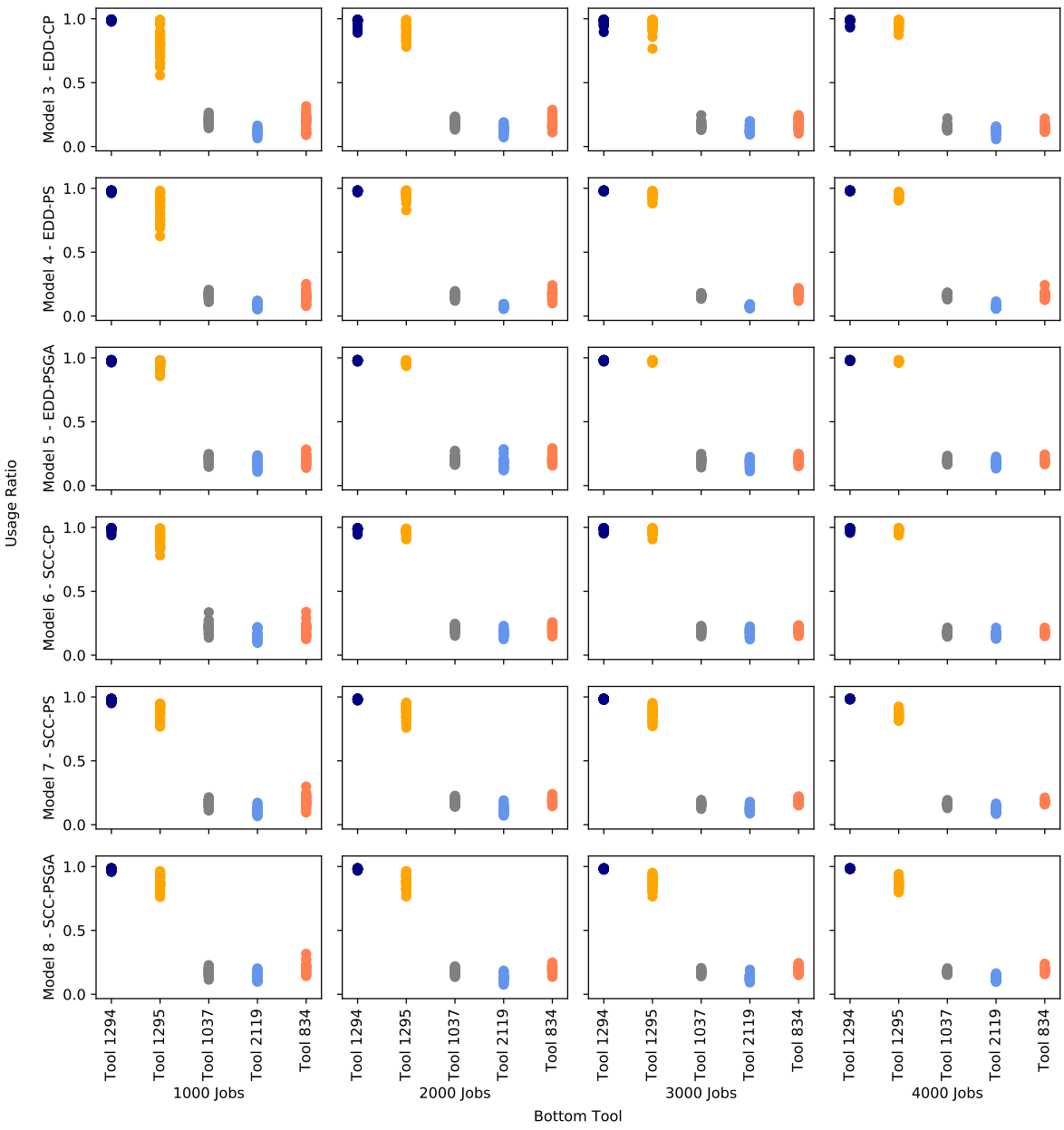


Figure 6.13: Tool usage ratios for the top five most popular tools.

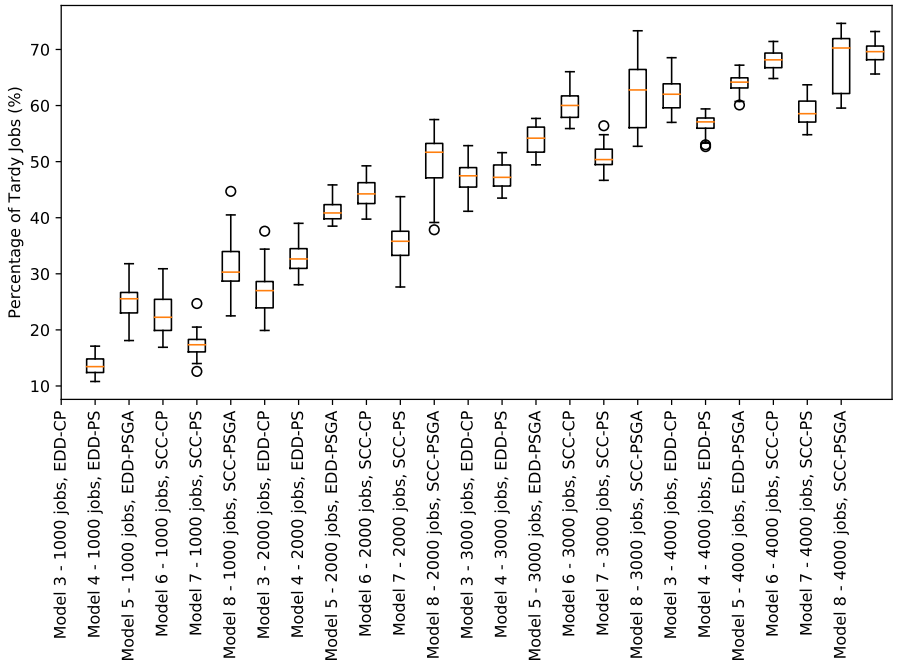


Figure 6.14: Percentage of tardy jobs within a schedule.

bound of 50% on the percentage of jobs that will end up being tardy in the 4000 job instances.

There is much room for future work with respect to the problem of scheduling in composites manufacturing. We have characterized the connections between packing and scheduling in this complicated problem and identified bottlenecks in the system. EDD-pack with CP-sched proved to be the most robust model developed to date. Future work could focus on improving the EDD-pack algorithm with additional heuristic guidance based on the tardiness predictors presented in this chapter, or focus on developing a lower bound for CP-sched such that we can obtain some measure of optimality. Another possible direction is developing an exact scheduling model that focuses on making decisions around the bottlenecks and a heuristic algorithm that then schedules remaining jobs and makes repairs.

Chapter 7

Concluding Remarks

This chapter concludes this thesis by presenting a summary of the previous chapters along with major conclusions, followed by some directions for potential future work.

7.1 Summary and Contributions

In this thesis, we formally defined a novel optimization problem, the Composites Manufacturing Problem (CMP), as well as its abstraction, the Two-Stage Bin Packing and Hybrid Flowshop Scheduling Problem (2BPHFSP). The CMP is a real-life batching and scheduling problem that almost all aerospace manufacturers face across the world. We were given an sample order of jobs by our industrial partner that we used to generate bootstrap instances for the CMP. Problem instances for the 2BPHFSP were randomly generated from uniform distributions. One contribution of this work is bringing attention to these two problems, as well as describing the full complexity of the CMP.

We presented five approaches to solve the 2BPHFSP: Mixed Integer Programming (MIP), Constraint Programming (CP), two Logic-based Benders Decompositions (LBBD) with different optimality cuts, and an Earliest-Due Date (EDD) heuristic. We tested these approaches on instances containing 5 to 100 jobs.

The two LBBD models had the best performance, but showed a negligible difference between themselves. After conducting statistical tests, we concluded that neither set of optimality cuts are strong enough to do much more than cut off the incumbent solution. The EDD heuristic performed surprisingly well, and was also able to solve all instances as opposed to the LBBD models, which were not able to solve some instances with more than 40 jobs. We also tried to improve an EDD heuristic solution using CP in two ways: warm-starting the monolithic CP model and using a CP scheduling model to improve the schedule given a fixed packing; the two heuristic-CP approaches demonstrated the highest potential for scalability.

We presented nine overall approaches to solve the CMP, where each overall approach is either made up of one packing model/algorithm and one scheduling model/algorithm or the approach is an LBBD model. We developed four packing models/algorithm: MIP, CP, an EDD heuristic algorithm, and

a constrained clustering algorithm; we developed three scheduling models/algorithms: CP, a parallel scheduling algorithm, and a genetic algorithm. We also developed a preprocessing step where we solve the instance using a Relaxed Scheduling Problem (RSP) to obtain the order of a job in the RSP solution. This order, denoted as the RSP rank, is then used to guide packing decisions to batch jobs with similar RSP ranks together. We tested these approaches on instances with up to 4000 jobs; a 4000 job instance represents the size of an average real-life instance.

The approach using the EDD packing algorithm with the CP scheduling model had the best performance across all instance sizes. The CP scheduling model is most likely outperforming the other scheduling models due to its ability to reason about where jobs are scheduled and backtrack on bad decisions. The parallel scheduling algorithm and genetic algorithm, on the other hand, rely on a moving horizon that updates its earliest start time every time an autoclave batch is fully scheduled. Therefore, we are unable to take advantage of empty spaces in the schedule after the horizon has been updated past those spaces. The results also showed that packing decisions influence the schedule tardiness through three correlating factors: the number of autoclave batches in the packing solution, the distribution of RSP ranks within an autoclave batch, and the distribution, or balance, of jobs across autoclave batches. We discovered two sets of bottleneck resources: the labour teams and machines in layup as well as tools. These bottlenecks imply that the best solutions to the CMP are reasonably close to optimal.

7.2 Future Work

The results of Chapter 4 show that the EDD heuristic performed better than the monolithic MIP and CP models, and it performed comparably to the LBBD models. Thus, we believe that there are most likely more sophisticated decompositions or exact algorithms/models that can be developed.

For both the CMP and the 2BPHFSP, we did not develop any theorems about the connections between the packing and scheduling decisions. In Chapter 6, we presented some analysis on how packing decisions may impact the eventual schedule tardiness. An important direction for future work is to pose and prove some theoretically sound statements about the nature of the connection between packing and scheduling decisions. Such a connection could then be used to construct separate packing and scheduling approaches where we know beforehand how packing decisions will affect the resulting scheduling problem. This lack of connection was one of the biggest weaknesses of the models in Chapters 5 and 6.

We also showed in Chapter 6 that there are severe inherent bottlenecks in the system, specifically the availability of tools. Thus, a new optimization problem could be posed that attempts to determine the best collection of tools a plant should have on hand based on how often the tools are used and their cost. This problem would be applicable to plants that are looking to update their resources or new plants that are looking to purchase their first set of tools.

The genetic algorithm of Hartmann [55] tested in Chapter 6 was taken from resource-constrained project scheduling literature. In recent years, several new metaheuristics have been developed that out-perform the genetic algorithm. The results in Chapter 6 showed that the genetic algorithm performed comparably to the CP scheduling model. Therefore, it would be interesting to test some newer

metaheuristics and see if they can outperform CP.

7.3 Conclusion

The goal of this thesis is to present the CMP, a novel problem, and the 2BPHFSP, its abstraction, and develop and investigate several solution approaches to each problem. The CMP is a complex problem with many layers of decisions while the 2BPHFSP is emblematic of the types of abstracted problems researchers tend to solve in place of the real-life problem. However, we believe that tackling the real problem is crucial to being able to use mathematical techniques to improve processes; it is not enough to stop at solving the abstracted problems. We presented five approaches to solve the 2BPHFSP, nine approaches to solve the 2BPHFSP, and empirical analysis on randomly generated instances for both problems. We demonstrated that ad-hoc heuristic algorithms and CP are good approaches to solve both problems.

We believe the work completed in this thesis is a comprehensive attempt at solving a complex and intricate real-life problem by developing sophisticated models and utilizing techniques from a broad range of sources. We also completed an extensive analysis of results that exposed many hidden connections and inherent structures of the problem.

Bibliography

[1] Abderrahmane Aggoun and Nicolas Beldiceanu. Extending CHIP in order to solve complex scheduling and placement problems. *Mathematical Computer Modelling*, 17:57–73, 1993.

[2] Ali Allahverdi, Jatinder N.D. Gupta, and Tariq Aldowaisan. A review of scheduling research involving setup considerations. *International Journal of Management Science*, 27:219–239, 1999.

[3] Ali Allahverdi, C.T. Ng, T.C.E. Cheng, and Mikhail Y. Kovalyov. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187:985–1032, 2006.

[4] Christian Artigues, Philippe Michelon, and Stephane Reusser. Insertion techniques for static and dynamic resource-constrained project scheduling. *European Journal of Operational Research*, 149:249–267, 2003.

[5] Susan F. Assmann, David S. Johnson, Daniel J. Kleitman, and Joseph Y.T. Leung. On a Dual Version of the One-Dimensional Bin Packing Problem. *Journal of Algorithms*, 5:502–525, 1984.

[6] Aria Azami, Kudret Demirli, and Nadia Bhuiyan. Scheduling in aerospace composite manufacturing systems: a two-stage hybrid flow shop problem. *The International Journal of Advanced Manufacturing Technology*, 95:3259–3274, 2018.

[7] Nagraj Balakrishnan, John J. Kanet, and Sri V. Sridharan. Early/tardy scheduling with sequence dependent setups on uniform parallel machines. *Computers and Operations Research*, 26:127–141, 1999.

[8] Arindam Banerjee and Joydeep Ghosh. Frequency-sensitive competitive learning for scalable balanced clustering on high-dimensional hyperspheres. *IEEE Transactions on Neural Networks*, 15:702–719, 2004.

[9] Philippe Baptiste and Claude Le Pape. Constraint Propagation and Decomposition Techniques for Highly Disjunctive and Highly Cumulative Project Scheduling Problems. *Constraints*, 5:119–139, 2000.

[10] Philippe Baptiste, Claude Le Pape, and Wim Nuijten. *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*. Springer, 2012.

[11] S. Basu, I. Davidson, and K. Wagstaff. *Constrained Clustering: Advances in Algorithms, Theory, and Applications*. CRC Press, 2008.

[12] Sugato Basu, Arindam Banerjee, and Raymond J. Mooney. Active Semi-Supervision for Pairwise Constrained Clustering. In *Proceedings of the Fourth SIAM International Conference on Data Mining*, pages 333–344. SIAM, April 2004.

[13] J. Christopher Beck and Mark S. Fox. Constraint-directed techniques for scheduling alternative activities. *Artificial Intelligence*, 121:211–250, 2000.

[14] D.D. Bedworth and J.E. Bailey. *Integrated Production Control Systems - Management, Analysis, Design*. Wiley, 1982.

[15] A. Bellanger and Ammar Oulamara. Scheduling hybrid flowshop with parallel batching machines and compatibilities. *Computers and Operations Research*, 36:1982–1992, 2009.

[16] Gleb Belov and Guntram Scheithauer. A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting. *European Journal of Operational Research*, 171:85–106, 2006.

[17] H. Ben Amor and J.M. Valerio de Carvalho. Cutting stock problems. In Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon, editors, *Column Generation*, pages 131–161. Springer, 2005.

[18] Jacques F. Benders. Partitioning Procedures for Solving Mixed-Variables Programming Problems. *Numerische Mathematik*, 4:238–252, 1962.

[19] Timo Bethold, Stefan Heinz, Marco E. Lubbecke, Rolf H. Mohring, and Jens Schulz. A Constraint Integer Programming Approach for Resource-Constrained Project Scheduling. In *Proceedings of the Seventh International Conference on the Integration of AI and OR Techniques in Constraint Programming.*, pages 313–317. Springer, June 2010.

[20] F Blomer and H.-O. Gunther. Scheduling of a multi-product batch process in the chemical industry. *Computers in Industry*, 36:245–259, 1998.

[21] F.F. Boctor. Some efficient multi-heuristic procedures for resource-constrained project scheduling. *European Journal of Operational Research*, 49:3–13, 1990.

[22] Tim Bowler. Carbon fibre planes: Lighter and stronger by design. *BBC News*, 2014.

[23] Edward H. Bowman. The Schedule-Sequencing Problem. *Operations Research*, 7:621–624, 1959.

[24] Filipe Brandao and Joao Pedro Pedroso. Solving Bin Packing Related Problems Using an Arc Flow Formulation. Report, University of Porto, 2017.

[25] Peter Brucker, Andrea Drexl, Rolf Mohring, Klaus Neumann, and Erwin Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112:3–41, 1999.

[26] Peter Brucker, Andrei Gladky, Han Hoogeveen, Mikhail Y. Kovalyov, Chris N. Potts, Thomoas Tautenhahn, and Steef L. Van de Velde. Scheduling a Batching Machine. *Journal of Scheduling*, 1:31–54, 1998.

[27] Edmund K. Burke, Sanja Petroniv, and Rong Qu. Case-based heuristic selection for timetabling problems. *Journal of Scheduling*, 9:115–132, 2006.

[28] C.H. Cheng, B.R. Feiring, and T.C.E. Cheng. The cutting stock problem - a survey. *International Journal of Production Economics*, 36:291–305, 1994.

[29] Henrik I. Christensen, Arindam Khan, Sebastian Pokutta, and Prasad Tetali. Approximateion and online algorithms for multidimensional bin packing: A survey. *Computer Science Review*, 24:63–79, 2017.

[30] Yingyi Chu and Quanshi Xia. A Hybrid Algorithm for a Class of Resource Constrained Scheduling Problems. In *Proceedings of the Second International Conference on the Integration of AI and OR Techniques in Constraint Programming.*, pages 110–124. Springer, May 2005.

[31] Andre A. Cire, Elvin Coban, and John N. Hooker. Logic-based Benders decomposition for planning and scheduling: a computational analysis. *The Knowledge Engineering Review*, 31:440–451, 2016.

[32] Edward G. Coffman, J. Csirik, Gabor Galambos, Silvano Martello, and Daniele Vigo. Bin packing approximation algorithms: Survey and classification. *Handbook of Combinatorial Optimization*, 1:455–531, 2013.

[33] Andres Collart. An Application of Mathematical Optimization to Autoclave Packing and Scheduling in a Composites Manufacturing Facility. Master's thesis, Dalhousie University, 2015.

[34] Dale F. Cooper. Heuristics for Scheduling Resource-Constrained Projects: An Experimental Investigation. *Management Science*, 22:1186–1194, 1976.

[35] H.A.J. Crauwels, A.M.A. Hariri, Chris N. Potts, and Luk N. Van Wassenhove. Branch and bound algorithms for single-machine scheduling with batch set-up times to minimize total-weighted completion time. *Annals of Operations Research*, 83:59–76, 1998.

[36] H.A.J. Crauwels, Chris N. Potts, and Luk N. Van Wassenhove. Local search heuristics for single machine scheduling with batch set-up times to minimize total weighted completion time. *Annals of Operations Research*, 70:261–279, 1997.

[37] George B. Dantzig and Philip Wolfe. Decomposition Principle for Linear Programs. *Operations Research*, 8:101–111, 1960.

[38] Thi-Bich-Hanh Dao, Khanh-Chuong Duong, and Christel Vrain. Constrained clustering by constraint programming. *Articial Intelligence*, 244:70–94, 2017.

[39] E.W. David and J.H. Patterson. A comparison of heuristic and optimum solutions in resource-constrained project scheduling. *Management Science*, 21:944–955, 1975.

[40] Maxence Delorme, Manuel Iori, and Silvano Martello. Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research*, 255:1–20, 2016.

[41] Guy Desaulniers, Jacques Desrosiers, and S. Spoorendonk. Cutting planes for branch-and-price algorithms. *Networks*, 58, 2011.

[42] Dieter Devels, Bert De Reyck, Roel Leus, and Mario Vanhoucke. A hybrid scatter search/electromagnetism meta-heuristic for project scheduling. *European Journal of Operational Research*, 169:638–653, 2006.

[43] Simon Emde, Lukas Polten, and Michel Gendreau. Logic-Based Benders Decomposition for Scheduling a Batching Machine. Discussion paper, CIRRELT, Montreal, Canada, 2018.

[44] Oluf Faroe, David Pisinger, and Martin Zachariasen. Guided Local Search for the Three-Dimensional Bin-Packing Problem. *INFORMS Journal on Computing*, 15:267–283, 2003.

[45] Krzysztof Fleszar and Khalil S. Hindi. Solving the resource-constrained project scheduling problem by a variable neighbourhood search. *European Journal of Operational Research*, 155:402–413, 2004.

[46] Ginger Gardiner. Optimizing composite aerostructures production. *Composites World*, 2015.

[47] Jay B. Ghosh. Batch scheduling to minimize total completion time. *Operations Research Letters*, 16:271–275, 1994.

[48] Paul C. Gilmore and Ralph E. Gomory. Multistage Cutting Stock Problems of Two and More Dimensions. *Operations Research*, 13:94–120, 1965.

[49] Anupriya Gogna and Akash Tayal. Metaheuristics: review and applications. *Journal of Experimental and Theoretical Artificial Intelligence*, 25:503–526, 2012.

[50] Hua Gong, Lixin Tang, and C.W. Duin. A two-stage flow shop scheduling problem on a batching machine and a discrete machine with blocking and shared setup times. *Computers and Operations Research*, 37:960–969, 2010.

[51] Monique Guignard and Siwhan Kim. Lagrangean decomposition: A model yielding stronger Lagrangean bounds. *Mathematical Programming*, 39:215–228, 1987.

[52] Jatinder N.D. Gupta and Johnny C. Ho. A new heuristic algorithm for the one-dimensional bin-packing problem. *Production Planning and Control*, 10:598–603, 1999.

[53] LLC Gurobi Optimization. Gurobi Optimizer Reference Manual, 2020.

[54] Andy Ham, John W. Fowler, and Eray Cakici. Constraint Programming Approach for Scheduling Jobs With Release Times, Non-Identical Sizes, and Incompatible Families on Parallel Batching Machines. *IEEE Transactions on Semiconductor Manufacturing*, 30:500–507, 2017.

[55] Sonke Hartmann. A Competitive Genetic Algorithm for Resource-Constrained Project Scheduling. *Naval Research Logistics*, 45:733–750, 1998.

[56] Sonke Hartmann and Dirk Briskorn. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207:1–14, 2010.

[57] Viktoria A. Hauder, Andreas Beham, Sebastian Raggl, Sophie N. Parragh, and Michael Affenzeller. On constraint programming for a new flexible project scheduling problem with resource constraints, 2019.

[58] S. Heipcke. Comparing constraint programming and mathematical programming approaches to discrete optimisation - the change problem. *The Journal of the Operational Research Society*, 50:581–595, 1999.

[59] Kim Hindle and Matt Duffin. Simul8-Planner for Composites Manfacturing. In *Proceedings of the 2006 Winter Simulation Conference*, pages 1779–1748. IEEE, December 2006.

[60] J.H. Holland. *Adaptation in Natural and Artificial Systems*. Michigan Press, 1975.

[61] John N Hooker. *Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction*. Wiley, 2001.

[62] John N. Hooker. Logic, Optimization, and Constraint Programming. *INFORMS Journal on Computing*, 14:293–420, 2002.

[63] John N. Hooker. Planning and Scheduling by Logic-Based Benders Decomposition. *Operations Research*, 55:588–602, 2007.

[64] Christian Hueber, Gudrun Fischer, Nikolaus Schwingshandl, and Ralf Schledjewski. Production planning optimisation for composite aerospace manufacturing. *International Journal of Production Research*, 57:5857–5873, 2018.

[65] IBM. Class IloDistribute, see Note. `https://www.ibm.com/support/knowledgecenter/SSSA5P_12.9.0/ilog.odms.cpo.help/refcppcpoptimizer/html/classes/IloDistribute.html`. Accessed: 2019-09-11.

[66] IBM ILOG. CPLEX, 2020.

[67] IBM ILOG. Interval variables in CP Optimizer, 2020.

[68] David S. Johnson. Fast algorithms for bin packing. *Journal of Computer and System Sciences*, 8:272–314, 1974.

[69] Hans Kellerer and Ulrich Pferschy. Cardinality constrained bin-packing problems. *Annals of Operations Research*, 92:335–348, 1999.

[70] J.E. Kelley Jr. The Critical Path Method: Resources Planning and Scheduling. *Industrial Scheduling*, 13:347–365, 1963.

[71] Robert Klein. Project scheduling with time-varying resource constraints. *International Journal of Production Research*, 38:3937–3952, 2000.

[72] Rainer Kolisch and Sonke Hartmann. Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research*, 174:23–37, 2006.

[73] E. Kondili, Constantinos Pantelides, and Roger W.H. Sargent. A General Algorithm for Scheduling Batch Operations. In *Proceedings of the Third International Symposium on Process Systems Engineering*, pages 62–75. Barton, ACT: Institution of Engineers, August 1988.

[74] Sebastian Kosch and J. Christopher Beck. A New MIP Model for Parallel-Batch Scheduling with Non-identical Job Sizes. In *Proceedings of the 11th International Conference on the Integration of AI and OR Techniques in Constraint Programming.*, pages 55–70. Springer, May 2014.

[75] K.L. Krause, V.Y. Shen, and H.D. Schwetman. Analysis of Several Task-Scheduling Algorithms for a Model of Multiprogramming Computer Systems. *Journal of the Association of Computer Machinery*, 22:522–550, 1975.

[76] Mary E. Kurz and Ronald G. Askin. Scheduling flexible flow lines with sequence-dependent setup times. *European Journal of Operational Research*, 159:66–82, 2004.

[77] Ailsa Land and Alison Doig. An Automatic Method of Solving Discrete Programming Problems. *Econometrica*, 28:497–520, 1960.

[78] Xueping Li and Kaike Zhang. Single batch processing machine scheduling with two-dimensional bin packing constraints. *International Journal of Production Economics*, 196:113–121, 2018.

[79] Zhenguo Li, Jianzhuang Liu, and Xiaoou Tang. Constrained clustering via spectral regularization. In *Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 421–428. IEEE, June 2009.

[80] Ching-Jong Liao and Li-Man Liao. Improved MILP models for two-machine flowshop with batch processing machines. *Mathematical and Computer Modelling*, 48:1254–1264, 2008.

[81] Ching-Jong Liao and Chii-Tsuen You. An Improved Formulation for the Job-Shop Scheduling Problem. *The Journal of the Operational Research Society*, 43:1047–1054, 1992.

[82] Philippe Liess, Olivier aand Michelon. A constraint programming approach for the resource-constrained project scheduling problem. *Annals of Operational Research*, 157:25–36, 2008.

[83] Richard Linn and Wei Zhang. Hybrid flow shop scheduling: A survey. *Computers and Industrial Engineering*, 37:57–61, 1999.

[84] Andrea Lodi, Silvano Martello, and Michele Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141:241–252, 2002.

[85] Andrea Lodi, Silvano Martello, and Daniele Vigo. Models and Bounds for the Two-Dimensional Level Packing Problem. *Journal of Combinatorial Optimization*, 8:363–379, 2004.

[86] J.B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Symbopsium on Math, Statistics, and Probability*, pages 281–297. University of California Press, 1967.

[87] Arnaud Malapert, Christelle Gueret, and Louis-Martin Rousseau. A constraint programming approach for a batch processing problem with non-identical job sizes. *European Journal of Operational Research*, 221:533–545, 2012.

[88] Alan S. Manne. On the Job-Shop Scheduling Problem. *Operations Research*, 8:219–223, 1960.

[89] Feng Mao, Edgar Blanco, Mingang Fu, Rohit Jain, Anurag Gupta, Sebastien Mancel, Rong Yuan, Stephen Guo, Sai Kumar, and Yayang Tian. Small Boxes Big Data: A Deep Learning Approach to Optimize Variable Sized Bin Packing. In *Proceedings of the Third IEEE International Conference on Big Data Computing Services and Applications*, pages 80–89. IEEE, April 2017.

[90] George Marsh. Arbus takes on Boeing with reinforced plastic A350 XWB. *Reinforced Plastics*, 52, 2007.

[91] Silvano Martello, David Pisinger, and Daniele Vigo. The Three-Dimensional Bin Packing Problem. *Operations Research*, 48:256–267, 2000.

[92] Silvano Martello and Paolo Toth. Lower bounds and reduction procedures for the bin packing problem. *Discrete Applied Mathematics*, 28:59–70, 1990.

[93] Andrew J. Mason. Genetic algorithms and scheduling problems. Master's thesis, University of Cambridge, 1992.

[94] Lars Monch, John N. Fowler, Stephane Dauzere-Peres, and Scott J. Mason. A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations. *Journal of Scheduling*, 14:583–599, 2011.

[95] Clyde L. Monma and Chris N. Potts. On the Complexity of Scheduling with Batch Setup Times. *Operations Research*, 37:798–804, 1989.

[96] Anulark Naber and Rainer Kolisch. MIP models for resource-constrained project scheduling with flexible resource profiles. *European Journal of Operational Research*, 239:335–348, 2014.

[97] Q.Q. Nong, C.T. Ng, and T.C.E. Cheng. The bounded single-machine parallel-batching scheduling problem with family jobs and release dates to minimize makespan. *Operations Research Letters*, 36:61–66, 2008.

[98] Irfan M. Ovacik and Reha Uzsoy. *Decomposition Methods for Complex Factory Scheduling Problems*. Springer, 1997.

[99] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, 1998.

[100] Laurent Perron and Vincent Furnon. OR-Tools, 2020.

[101] Chris N. Potts and Mikhail Y. Kovalyov. Scheduling with batching: A review. *European Journal of Operational Research*, 120:228–249, 2000.

[102] Chris N. Potts and Luk N. Van Wassenhove. A Decomposition Algorithm for the Single Machine Total Tardiness Problem. *Operations Research Letters*, 1:177–181, 1982.

[103] Geoff Poulton. Aviation's material evolution. *Airbus*, 2017.

[104] A. Alan B. Pritsker, Lawrence J. Waiters, and Philip M. Wolfe. Multiproject Scheduling with Limited Resources: A Zero-One Programming Approach. *Management Science*, 16:93–107, 1969.

[105] Jakob Puchinger and Gunther R. Raidl. Models and algorithms for three-stage two-dimensional bin packing. *European Journal of Operational Research*, 183:1304–1327, 2007.

[106] Marcela Quiroz-Castellanos, Laura Cruz-Reyes, Jose Torres-Jimenez, Claudia Gomez S., Hector J. Fraire Haucuja, and Adriana C.F. Alvim. A grouping genetic algorithm with controlled gene transmission for the bin packing problem. *Computers and Operations Research*, 55:52–64, 2015.

[107] J.C. Regin. *Global Constraints: A Survey*, pages 63–134. Springer, 2010.

[108] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of Constraint Programming*. Elsevier, 2006.

[109] Napat Rujeerapaiboon, Kilian Schindler, Daniel Kuhn, and Wolfram Wiesemann. Size Matters: Cardinality-Constrained Clustering and Outlier Detection via Conic Optimization, 2017.

[110] John Schmidt. A Major Aerospace Concern: Manufacturing Delays and Ways to Solve Them. *Manufacturing Net*, 2014.

[111] Andreas Schutt, Thibaut Feydy, Peter J. Stuckey, and Mark G. Wallace. Explaining the `cumulative` propagator. *Constraints*, 16:250–282, 2011.

[112] Paul Shaw. A Constraint for Bin Packing. In *Proceedings of the 10th International Conferences on the Principles and Practice of Constraint Programming*, pages 648–662. Springer, September 2004.

[113] J. Skorin-Kapov and Vakharia. A.J. Scheduling a flow-line manufacturing cell: A tabu search approach. *The International Journal of Production Research*, 31:1721–1734, 1993.

[114] Jeff Sloan. Composites 101: Fibers and resins. *Composites World*, 2016.

[115] Alexander Strehl and Joydeep Ghosh. Relationship-Based Clustering and Visualization for High-Dimensional Data Mining. *INFORMS Journal on Computing*, 15:208–230, 2003.

[116] Chang-Sup Sung, Young-Hwan Kim, and Sang-Hum Yoon. A problem reduction and decomposition approach for scheduling for a flowshop of batch processing machines. *European Journal of Operational Research*, 121:179–192, 2000.

[117] Lixin Tang and Peng Liu. Two-machine flowshop scheduling problems involving a batching machine with transportation or deterioration consideration. *Applied Mathematical Modelling*, 33:1187–1199, 2009.

[118] Arne Thesen. Heuristic Scheduling of Activities under Resource and Precedence Restrictions. *Management Science*, 23:412–422, 1976.

[119] Vincent T'kindt, Jatinder N.D. Gupta, and Jean-Charles Billaut. Two-machine flowshop scheduling wth a secondary criterion. *Computers and Operations Research*, 30:505–526, 2003.

[120] Tony T. Tran, Arthur Araujo, and J. Christopher Beck. Decomposition Methods for the Parallel Machine Scheduling Problem with Setups. *INFORMS Journal on Computing*, 28:83–95, 2016.

[121] Tuan Tony Tran. *Decomposition Models for Complex Scheduling Applications*. PhD thesis, University of Toronto, 2017.

[122] A.J. Vahkaria and Y.L. Chang. A simulated annealing approach to scheduling a manufacturing cell. *Naval Rsearch Logistics*, 37:559–577, 1990.

[123] Willem-Jan Van Hoeve, Gilles Pesant, Louis-Martin Rousseau, and Ashish Sabharwal. Revisiting the Sequence Constraint. In *Proceedings of the 12th International Conference on the Principles and Practice of Constraint Programming*, pages 620–634. Springer, September 2006.

[124] Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schroedl. Constrained K-means Clustering with Background Knowledge. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 577–584. Morgan Kaufmann, June 2001.

[125] Frank Werner. *A Survey of Genetic Algorithms for Shop Scheduling Problems*, pages 161–222. Nova Science Publishers, 2013.

[126] Dongkuan Xu and Yingjie Tian. A Comprehensive Survey of Clustering Algorithms. *Annal of Data Science*, 2:165–193, 2015.

[127] R. Xu and D. Wunsch. *Clustering*. Wiley, 2008.

[128] Naoto Yamashita and Kohei Adachi. A Modified k-Means Clustering Procedure for Obtaining a Cardinality-Constrained Centroid Matrix. *Journal of Classification*, 97, 2019.

[129] Shaohong Zhang and Hau-San Wong. Partial closure-based constrained clustering with order ranking. In *Proceedings of the Nineteenth International Conference on Pattern Recognition*, pages 1–4. IEEE, December 2008.

[130] Shaohong Zhang, Hau-San Wong, and Dongqing Xie. Semi-supervised clustering with pairwise and size constraints. In *Proceedings of the 2014 International Joint Conference on Neural Networks.*, pages 2450–2457. IEEE, July 2014.

[131] Shunzhi Zhu, Dingding Wang, and Tao Li. Data clustering with size constraints. *Knowledge-Based Systems*, 23:883–889, 2010.