CONTRIBUTIONS TOWARDS SOLVING THE PARALLEL MACHINE SCHEDULING PROBLEM WITH NON-AVAILABILITY PERIODS

by

Mark Manser

Submitted in partial fulfillment of the requirements
for the degree of Master of Applied Science

at

Dalhousie University
Halifax, Nova Scotia
August 2021

Dedicated to the late Dr. Eldon Gunn

Table of Contents

List of Tables

List of Figures

Abstract

An abundance of research exists studying parallel machine scheduling, but only a portion of this research has focused on the case with non-availability periods and job resumability factors. Research considering this case has mostly focused on limited cases such as a small, fixed number of machines or fixed number of non-availability periods. In this research, we improve the runtime of the model by Beaton et al. (2016) who proposed a mixed-integer linear programming (MILP) model allowing an arbitrary number of machines, non-availability periods, and resumability factor for small instances of the makespan minimization case. We propose a new MILP model for the Total Weighted Completion Time (TWC) case and two constraint programming (CP) models for both the makespan and TWC cases and find that the CP models outperform the MILP for the TWC case. We confirm that the well-known Weighted Shortest Processing Time heuristic works well for large TWC instances.

Acknowledgements

Thank you to my friends and family for your love and support throughout my journey in the fields of industrial engineering and operations research. A special thank you to my supervisor Dr. Claver Diallo, a friend and mentor without whom I surely would not have completed this thesis. I also express my sincerest gratitude to Dr. John Blake, a member of my supervisory committee, and to Dr. Majid Taghavi, the external examiner. They have provided valuable suggestions and insightful comments.

Chapter 1

Introduction

Scheduling plays an important role in the modern economy and in everyday life. Appointments, meetings, conferences, and recreational activities all need to be organized to occur within the set time intervals. Proper scheduling ensures that all of these activities can occur without unnecessary waiting periods and avoids resource conflicts. In short, scheduling provides us with confidence that activities can be executed efficiently.

Depending upon the type of activity being scheduled, a different performance metric may be used to describe what makes a schedule efficient. For example, a repair shop may be tasked with completing all tasks of a work order in as short a time as possible (called makespan) in order to return a vehicle to service promptly. A contract manufacturer may wish to prioritize the completion of its jobs according to their economic value (total weighted completion time). A satellite specialist may need to schedule events to start or end as close to a certain alignment time as possible (weighted earliness and tardiness). In general, scheduling involves assigning tasks to a finite number of resources in a way that does not exceed capacity and satisfies some target or optimizes some performance metric.

A common approach for modelling such problems is the machine scheduling problem which represents resources as machines and activities as jobs which must be assigned to machines. This type of problem has been the subject of a vast and diverse amount of research over past decades and continues today due to its impact in all parts of our lives. A common assumption in machine scheduling problems is that machines are continuously available throughout the planning horizon. This is not always the case, and there is opportunity for improved performance by determining how to best schedule jobs that must stop before and resume immediately after

non-availability periods, especially in the case of interruption penalties.

1.1 Preliminaries

1.1.1 Parallel Machine Scheduling Notation

Parallel machine scheduling problems can be described using a notation originally proposed by Graham et al. [27]. This notation is usually presented as the ordered triple $\alpha \mid \beta \mid \gamma$ which represent the environment, the job characteristics, and the objective or performance metric respectively.

The environment parameter α classifies the environment in which the machine operates as either single, parallel, or dedicated [57]. A single machine environment is one in which all jobs must be processed on one machine. A parallel (or multiple) machine environment allows jobs to be processed on more than one machine. Each parallel machine may be either identical, meaning that jobs require the same amount of time to process regardless of the machine to which they are assigned; uniform, meaning that all jobs assigned to that machine require an amount of time to process that is a constant (called *speed factor*) multiple of their base processing time; or unrelated meaning that there is, in general, no relation between the machines. A dedicated machine environment is one in which jobs have multiple steps that must be processed on different machines. Examples include open shops where machines are uniform (different speed factors) and each job must be processed on each machine; job shops where the order of machines on which a job must be processed differs by job; and flow shops where there is a single machine sequence and all jobs must be processed on machines according to that sequence[31].

The job characteristics parameter β specifies the choices (and consequences) that can be made in scheduling the processing of each job. Examples include whether jobs can be preempted (interrupted prior to completion in order to start a different job), resumed (processing interrupted and then continued at a later time, possibly with some rework or other considerations), and also whether there exists non-availability periods in the schedule that must be accommodated. Preemption has many types, two

of the more common of which are *operational* and *arbitrary* [57]. Operational preemption requires that a job must be resumed on the same machine upon which it started whereas arbitrary preemption relaxes this requirement allowing the job to resume on any machine to which it could otherwise be assigned. Problems where jobs cannot be preempted are intuitively called non-preemptive. Resumability refers to the rework penalty that is incurred when a job resumes immediately after an interruption. Jobs can be fully-resumable (no rework penalty), non-resumable (job must restart from its beginning) [47], or semi-resumable (incur a rework penalty proportional to some fraction of the work completed prior to interruption) [48]. The parameter β, as well as the other parameters, can be a compound parameter, in which case it is usually hyphenated to separate sub-parameters. This allows the combination of multiple job characteristics in a single model such as resumability and non-availability periods [31].

The objective metric parameter γ specifies the metric to be optimized. Examples include minimizing makespan (end time of last job completed on any machine), weighted total completion time (scalar product of each job's end time and a numeric weight given for each job), and total lateness (sum of completion time of each job subtract its due date) [31].

The following scheduling problems are considered in this research:

$$P_m \mid ar - a_{i,q} \mid C_{max}$$
$$P_m \mid r - a_{i,q} \mid \sum w_j C_j$$

In the first scheduling problem the first parameter (α) has value P_m which represents the parallel machine problem. The second, compound parameter (β) has first value ar which represents an arbitrary resumability factor falling anywhere between 0 and 1, where 0 represents zero rework required (fully-resumable case) and 1 represents 100% rework required (non-resumable case). The second value of the β parameter has value $a_{i,q}$ which represents the number of non-availability periods q on each machine i. This research considers an arbitrary number of machines and an arbitrary number of non-availability periods per machine. The third parameter (γ) has value C_{max} where C represents the completion time of jobs and therefore C_{max} represents

the maximum completion time of any job (a.k.a makespan) which is to be minimized.

In the second scheduling problem the β parameter has first value r which represents the fully-resumable case (nr would represent the non-resumable case). The γ parameter has value $\sum w_j C_j$ which means Total Weighted Completion Time (TWC), or equivalently the scalar product of machine completion times and their priority scores.

1.1.2 Accounting For Non-Availability Periods & Resumability

We now discuss the importance of considering non-availability periods when scheduling. Consider the following situation generalized from Hashemian [31]: Two jobs $j1$ and $j2$ need to be scheduled on parallel machines $M1$ and $M2$ to minimize makespan. Job $j1$ has a longer duration than job $j2$. We refer to Figure 1.1 for this example. If there are no non-availability periods, then the obvious solution is to assign each job to a different machine arbitrarily. However, if there are non-availability periods the decision making process requires more thought. If the jobs must not be interrupted, then only jobs that can be completed during the time preceding a non-availability period may be scheduled. As such, longer jobs could only start after the end of the non-availability period. If the jobs can be interrupted, then we wish to avoid unnecessary interruptions such as in Figure 1.1 where the interruption of job $j1$ by the non-availability period causes it to have two sections, the later of which completes at a later time than if the jobs had been assigned to opposite machines. This is because job $j2$ would have completed before the start of the non-availability period, also shown in Figure 1.1 if it had been assigned to machine $M1$ and the longer job $j1$ would not have been interrupted on machine $M2$.

Yet another factor needs to be considered when scheduling if a rework penalty is introduced. Figure 1.2 shows how the completion time of a job changes depending upon the magnitude of the rework penalty. In such cases the scheduling decision must also account for the amount of processing time completed prior to interruption in addition to the duration of the non-availability period.

The interaction between a job's completion time and the amount of processing

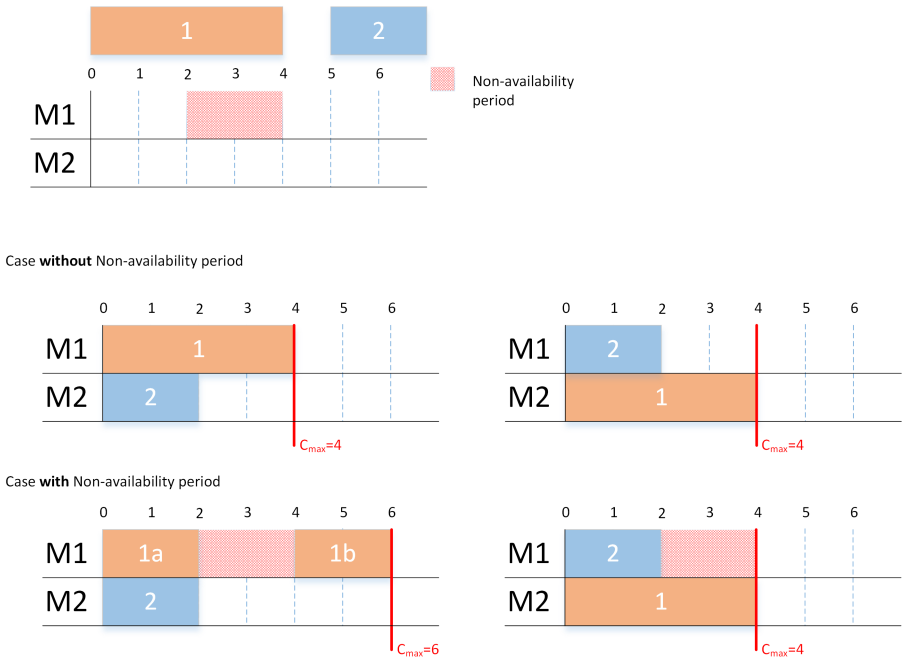

Figure 1.1: Optimal Solution With and Without Non-Availability Period

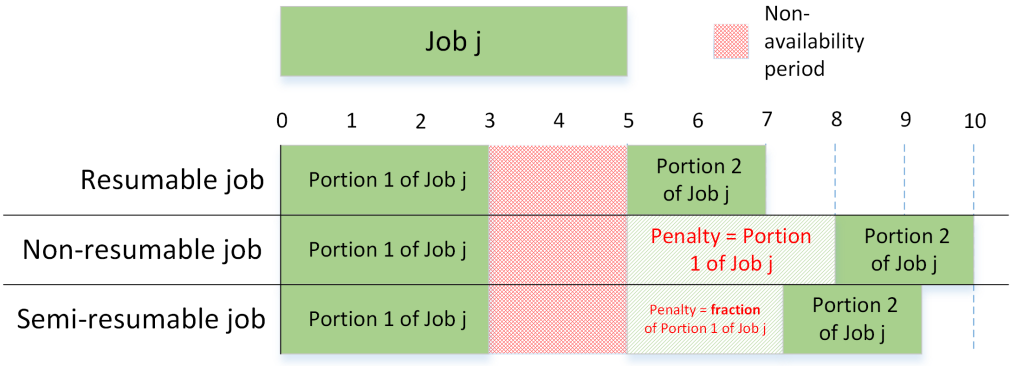

Figure 1.2: Types of Resumability

time completed before successive jobs are interrupted leads to an extremely challenging scheduling problem. Indeed, even makespan minimization problems with two machines are NP-Hard as shown by Lenstra et al. [53]. Given the importance of these types of scheduling problems, it is thus no surprise that special methodologies have been developed in attempts to find good solutions to the problems.

1.1.3 Solution Methods

Given the importance and complexity of the parallel machine scheduling problem, many heuristic methods have been designed to find good solutions quickly. Many of these heuristics involve the well-known List Scheduling (LS) procedure. In LS, jobs are sorted according to a sorting rule and then, one at a time, in sorted order, a machine is chosen to which they are assigned using an assignment rule. Examples of sorting rules include Longest Processing Time (LPT) such as in the LPT2 heuristic of Lee [47], and Weighted Shortest Processing Time (WSPT) [69] where jobs are sorted in decreasing order of their priority to processing time ratio. Examples of assignment rules include choosing the machine with the earliest available starting time, or in the case of non-availability periods, choosing the machine which will result in the job completing earliest.

A common exact method used for scheduling is Mixed Integer Linear Programming (MILP). Formulating a parallel machine scheduling problem as a MILP typically involves defining binary variables representing the assignment of jobs to specific machines, defining a linear objective function that is some (possibly indirect) function of the assignment variables, and expressing the restrictions on job assignments and job start times as linear constraints. The model is then solved using a generic MILP solver which produces progressively better solutions using a combination of branching strategies, cutting planes, and heuristics. The strength of a MILP model generally comes from the strength of its linear programming relaxation. Unfortunately, it is well known that many scheduling problems suffer from weak linear programming relaxations which is a major contributor to the difficulty of proving optimality using MILP models.

An alternative exact method that is well known to have achieved significant success on scheduling problems (at least for the minimum makespan objective [42]) is Constraint Programming (CP). Unlike MILP, the strength of CP does not (at least traditionally) rely upon the strength of a linear programming relaxation. In fact, many CP models need not be linear at all. Historically, CP has instead been a method focused on finding feasible solutions to problems. However, optimality can be achieved by successively solving feasibility problems and then adding a constraint that any solution must improve upon the objective function value of the incumbent solution.

One limitation of CP is that all variables and coefficients must be integers. This is because (finite domain) CP solvers represent each variable as its integer domain. However, this has not posed a problem for many scheduling problems since researchers have traditionally been most interested in solving problems where time variables are guaranteed to be integer valued, or the time resolution of the model can be scaled such that this is achieved. A strength of this domain-based representation is that it allows the use of specialized filtering algorithms to eliminate values that cannot lead to a feasible solution, thereby reducing the search space. The domains of variables are watched by the solver and when a value in the domain of one variable is filtered, the effect is propagated to other constraints which then determine the values that can be filtered from the domains of other variables as a consequence. This process cascades, reducing the number of domain values.

CP solvers also use different types of search strategies. Fixed search strategies are typically problem-specific and user-provided. These often consist of a variable selection strategy that decides upon which variable to branch in the search tree, and a value selection strategy that selects the next integer value in the domain to try. This is often sufficient control to implement some of the scheduling heuristics that have proven so effective which is a major contributor to the success of CP for scheduling problems, along with the filtering power of specialized global constraints [70].

More recently, several solvers have adopted Lazy Clause Generation [63][20], a

technique that leverages improvements from Boolean satisfiability (SAT) solving. This technique has improved solvers' abilities to learn from mistakes during search via explanations and often leads to significant reductions in solve times along with allowing some solvers [65] to develop effective automatic search. Historically, when modelling with CP it has often been advantageous to represent a problem as combinations of global constraints [9]. This is because global constraints provide the strong filtering and propagation that lead to efficient solving. Consequently, equivalent CP and MILP models can look quite different.

1.2 Literature Review

1.2.1 Results for the Makespan Problem with Machine Non-Availability

Much of the machine scheduling literature consists of special restricted cases of the problems considered in this research.

The case of minimizing makespan with identical parallel machines where each machine has at most a single non-availability period and one of the machines is always available ($P_m \mid r - a_{m-1,1} \mid C_{max}$) was studied by Lee [46]. He proposed an algorithm based on Longest Processing Time (LPT) for m machines with optimality bound $(\frac{3}{2} - \frac{1}{2m}) \times C_{opt}$ that struggles [47] when non-availabilities are scheduled later than time 0 or if there is a non-availability period on each machine. The optimality bound shrinks to $(\frac{3}{2} - \frac{1}{2m_{avail}}) \times C_{opt}$ when we can exclude some machines because they are not available for a duration greater than the current makespan [49]. Lee [46] proposed a second Modified Longest Processing Time (MLPT) algorithm with optimality bound $\frac{4}{3}C_{opt}$ which was later improved to $\frac{5}{4}C_{opt}$ by Guohui et al [29]. Lee [47] introduced the LPT2 algorithm to better handle the case where non-availabilities are scheduled later than time 0. LPT2 differs from LPT in that a job is scheduled on the machine that would cause it to finish earliest rather than on the machine with the earliest start time. Still using the assumption that at least one of the machines is always available, Lee proved that the optimality bound for LPT2 for this problem is $(1 + \frac{(1-\frac{1}{m})}{2})C_{opt}$.

Two identical machines with one machine having a single non-availability period and one machine always being available ($P_2 \mid r - a_{1,1} \mid C_{max}$) was solved optimally for up to 100 jobs by Liao et al. [55] by decomposing the problem into four sub-problems.

The case where each job can only be assigned to a subset of machines and each machine has a non-availability period ($P_m \mid r - M_j - a_{m,1} \mid C_{max}$) was considered by Liao and Sheen [56]. They used a network flow approach to formulate the problem as a series of maximum flow problems that were then solved by a polynomial time search algorithm that either verified the infeasibility of the problem or otherwise solved it optimally.

Lee [47] showed that the case of non-resumable jobs was NP-Hard, even for a single machine with a single non-availability period ($1 \mid nr - a \mid C_{max}$), and strongly NP-Hard with more than one non-availability period. However, an optimality bound of $\frac{4}{3}C_{opt}$ can still be achieved by sorting the jobs in decreasing order of processing time and then assigning as many jobs as possible to the machine prior to the non-availability period and the remaining jobs after the non-availability period.

Non-resumable jobs with multiple machines and a single non-availability period ($P_m \mid nr - a_{m,1} \mid C_{max}$) was considered by Lee [47] who showed that the optimality bound for list scheduling with arbitrarily-ordered jobs was mC_{opt} and the optimality bound for the LPT heuristic was $C_{opt}\frac{m+1}{2}$ where m is the number of machines.

A single machine with non-resumable jobs and periodic maintenance periods that must be scheduled to start and end within a pre-defined interval ($1 \mid nr - fpa \mid C_{max}$) was considered by Chen [14] who proposed binary integer programming (BIP) models capable of solving problems up to 100 jobs to optimality as well as heuristic using LPT sorting rule to solve larger instances within 1% of optimality.

Two parallel machines with non-preemptable jobs where one machine is always active and the other has a periodic non-availability period ($P_2, M_1PU \mid C_{max}$) was considered by Xu and Yang [73] who proposed a BIP to optimally solve the problem

as well as approximation methods focused on average case analysis.

Multiple machines where machines are not initially available and jobs have a (release, delivery) time pair $(P, NC_{inc} \mid r_j - q_j \mid C_{max})$ was considered by Gharbi and Haouari [25] who proposed a branch-and-bound algorithm and semi-preemptive lower bounding procedure based on max-flow for optimally solving medium sized problems. The value NC_{inc} in the first parameter denotes the type of availability, in this case increasing since the machines are not initially available.

Two machines with a non-availability period on one machine and batch transportation to a delivery centre after processing $(P_2 \mid r - a, D \mid C_{max})$ is considered by Pan et al. [64] who proposed the H2 algorithm to optimally solve the problem with a worst-case ratio of $\frac{3}{2}$.

A single machine with non-resumable jobs and flexible preventive maintenance (FPM) $(P_2 \mid nr - FPM \mid C_{max})$ is considered by Chen et al. [16] who propose two MILP models, a branch-and-bound procedure, and four heuristics for solving the problem. FPM means that the machine cannot run longer than a time interval T without a non-availability. Similarly, two machines with non-resumable jobs and FPM $(P_2 \mid nr - FPM \mid C_{max})$ is considered by Chen et al. [15]. Two MILP models are proposed along with heuristics and solution improvement procedures for solving the problem.

Three machines with periodic non-availability periods, schedulable non-availability periods, and tool-change non-availability periods was considered by Xu et al. [74] who proposed a mixed BIP to solve the problem.

Multiple identical machines with resumable jobs and a single non-availability period $(P_m|r - a_{m,1}|C_{max})$ as well as an arbitrary number of non-availability periods $(P_m|r - a_{m,q}|C_{max})$ were considered by Hashemian et al. [31] [32] who proposed ILP models for both problems as well as an exact implicit enumeration algorithm to optimally solve large instances.

Multiple machines with arbitrary resumability factor and an arbitrary number of non-availability periods ($P_m|ar - a_{m,q}|C_{max}$) was considered by Beaton et al. [7] who proposed a MILP model for small instances and four heuristics for finding good solutions to large problems.

Several authors have reviewed the literature on availability constraints: Lee et al. (1997)[50], Schmidt (2000)[68], Sanlaville and Schmidt (2008)[67], Ma et al. (2010)[57], Kaabi and Harrath (2014)[34]. Most of the reviewed studies proposed methods for either one or two machines or considered a maximum number of non-availability periods on multiple machines. To the best of our knowledge, only Beaton et al. [7] propose a general MILP formulation.

1.2.2 Results for the Weighted Completion Time Problem with Machine Non-Availability

While the single machine minimum Total Completion Time (TC) problem ($1 \mid\mid \sum C_j$) can be solved in polynomial time by the well-known Shortest Processing Time (SPT) heuristic, and the single machine minimum Total Weighted Completion Time (TWC) problem ($1 \mid\mid \sum w_j C_j$) can be solved in polynomial time by the Weighted Shortest Processing Time (WSPT) heuristic [69], the multiple parallel machine minimum TWC problem ($P_m \mid\mid \sum w_j C_j$) was proven to be NP-Hard [11]. Readers are referred to the literature review of Kramer et al. [40] that covers the research on this topic ranging from the early work of Eastman et al. [19] who proposed a lower bound and a simple upper bound heuristic similar to WSPT for parallel machine TWC which was proven to have optimality bound $\frac{\sqrt{2}+1}{2} \cdot TWC_{opt}$ by Kawaguchi and Kyan [38], all the way to their own work proposing an arc-flow formulation that can optimally solve problems containing up to 400 jobs and finds good solutions up to 1000 jobs.

For the case of minimum TWC on a single machine with non-resumable jobs and a single non-availability period ($1 \mid nr - a \mid \sum w_j C_j$), Lee [47] showed that the objective bound of the WSPT heuristic is infinite even if all weights are equal.

A single machine with non-resumable jobs, one non-availability period, and unit weights ($1 \mid nr - a \mid \sum C_j$) was considered by Adiri et al. [3] who showed that an optimality bound of the SPT heuristic was $\frac{1}{4}$ and this was later improved to $\frac{2}{7}$ by Lee and Liman [51].

A single machine with preemptive jobs and multiple non-availability periods ($P_m, h_{1,q} \mid pmtn \mid \sum w_j C_j$) was considered by Wang et al. [72]. Note that $h_{i,q}$ in the first parameter of the notation specifies that there are i machines with q non-availability periods and is sometimes omitted if it can be inferred. Wang et al. addressed the two important special cases of proportional weights and a single non-availability period.

A single machine with non-resumable jobs and a single non-availability period ($1 \mid nr - a \mid \sum w_j C_j$) was considered by Kacem and Mahjoub [37] who proposed a fully polynomial time approximation scheme with approximation factor $1+\epsilon$ and give a chronology of approximation improvement for the problem. Kacem et al. [36] also proposed 3 exact methods: branch & bound (B&B), dynamic programming (DP), and MILP. B&B and DP were found to be complimentary and could optimally solve problems with up to 3000 jobs. Kacem and Chu [35] later proposed an improved B&B coupled with lower bound and heuristics that could solve problems with up to 6000 jobs.

A single machine with preemptive jobs with a release time (and optionally a single non-availability period) ($1 \mid pmtn, r_j \mid \sum w_j C_j$) was considered by Batsyn et al. [6] who proposed an online Weighted Shortest Remaining Processing Time (WSRPT) heuristic for problems with thousands of jobs with average relative error less than 0.1%.

A single machine with multiple non-availability periods and jobs that cannot be preempted by non-availability periods ($1, h_q \mid\mid \sum w_j C_j$) was considered by Sadfi et al. [66] who proposed four lower bounding procedures, the best of which had an average relative error of 0.4% converging to 0 as $n \to 500$ jobs.

Minimizing TWC and Maximum Lateness (ML) on a single machine with semi-resumable jobs and one or two maintenance periods that must also be scheduled within a fixed time window T was considered by Graves and Lee [28]. They show that when the planning horizon duration is long by comparison to the width of the maintenance time window the problem is NP-Complete and they propose a pseudo-polynomial time dynamic programming method for both objective functions. They also show that when the planning horizon duration is short by comparison to the width of the maintenance time window and one maintenance period must occur, then minimizing TWC is still NP-Complete but the SPT heuristic is optimal for the Total Completion Time (TC) case and the Earliest Due Date (EDD) heuristic is optimal for the ML case.

The case of minimizing TWC on multiple machines with non-preemtive jobs and release dates and multiple non-availability periods ($P_m, h_q \mid r_j \mid \sum w_j C_j$) was considered by Nessah and Chu [61] who proposed a polynomial time infinite split scheduling lower bound that improved some 2, 7, and 15 machine problems by up to 95% when coupled with existing B&B procedures for the problem.

A single machine with periodic maintenance ($1 \mid pm \mid \sum w_j C_j$) was considered by Krim et al. [41] who proposed a MILP model and heuristics for up to 1000 jobs with 10% average relative error.

Two machines where one machine becomes unavailable after a fixed time was considered by Lee and Liman [52] who proved it was NP-Complete and proposed a pseudo-polynomial time dynamic programming model for its solution, as well as a heuristic with 50% error bound.

Two machines with one machine having a single non-availability period and resumable jobs ($P_2, h_{2,1} \mid r-a \mid \sum w_j C_j$) and the case of non-resumable jobs ($P_2, h_{2,1} \mid nr-a \mid \sum w_j C_j$) were studied by Lee [47] who proposed dynamic programming models for each case. The similar case with non-resumable jobs ($P_2, h_{2,1} \mid nr-a \mid \sum w_j C_j$)

and multiple machines ($P_m, h_{1,1} \mid nr - a \mid \sum w_j C_j$) was considered by Zhao et al. [76] who proposed a fully polynomial time approximation scheme (FPTAS).

For multiple machines with a single non-availability period and non-preemptive jobs ($P_m, h_{i,1} \mid\mid \sum w_j C_j$), Mellouli et al. [58] proposed lower bounds based upon Lagrangian relaxation, a heuristic, and column-generation methods. For the unit weight version Mellouli et al. [59] proposed four MILP models, a dynamic programming model, two B&B procedures, dominance properties, and a constructive lower bound, some of which can solve problems with up to 4 machines and 50 jobs. Yoo and Lee [75] studied a similar problem, distinguishing between the independent and dependent maintenance cases where simultaneously multiple machines and only one machine may experience a non-availability period respectively.

Several cases of minimizing TWC on multiple machines with either resumable or non-resumable jobs and preventive maintenance non-availability periods or fixed-job non-availability periods with job weights equal to processing times and other limiting restrictions was considered by Fu et al. [21][22] who proposed inapproximability proofs for some problems and FPTAS for others.

Li & Yang [54] reviewed the literature on Total Weighted Completion Time (TWC) focusing on unrelated parallel machines. It is clear from the above review that there is a need for a formulation for the general case of the minimum Total Weighted Completion Time (TWC) problem for parallel machines with multiple arbitrary non-availability periods.

1.3 Motivation and Research Goals

Most scheduling problems in the literature make the assumption that each machine is available continuously over a finite or infinite horizon. While this assumption greatly simplifies the problem and may be valid in situations where interruptions are negligible, there are many cases where interruptions should not be ignored. For example, scheduled preventive maintenance, operator breaks, and shift changes are examples of foreseeable interruptions that, when accounted for, need not have as severe an

impact on a schedule as when ignored. Additionally, when a task resumes after an interruption, some amount of rework may be required causing further deviations. For this reason, there is value in developing methods that determine how to best mitigate the impact of schedule interruptions.

Most papers dealing with the parallel machine scheduling problem with non-availability periods deal with makespan minimization (Beaton et al. [7]; Hashemian et al. [32]). Our first research goal is to improve upon the results of the existing Mixed Integer Linear Programming (MILP) models for parallel machine scheduling with makespan minimization, multiple non-availability periods, and arbitrary resumability. We propose extensions to the model of Beaton et al. [7] that allows us to prove optimality of an additional instance of the Graham's experiment benchmark. We also propose two constraint programming models as an alternative, open-source, exact solution method that performs comparably to the commercial solvers used for solving the MILP.

The Total Weighted Completion Time (TWC) performance metric is less common in the parallel machine scheduling literature than the makespan performance metric. Even less common are papers that combine parallel machine scheduling, TWC as performance metric, and non-availability periods. Mellouli et al. [59] and Yoo and Lee (2016) [75] deal with multiple parallel machines with TWC and only one non-availability period per machine. Zhao and Tang [76] consider two parallel machines scheduling problem where one machine is not available in a specified time period (i.e., 2 machines and 1 non-availability period in all).

Our second research goal is to propose, to the best of our knowledge, the first MILP formulation to deal with identical parallel machine scheduling for total weighted completion times with multiple non-availability periods on all machines without additional restrictions. This is achieved by combining the position-based modelling approach by Mönch and Shen [60] and the non-availability modelling approach by Beaton et al. [7]. It is assumed that all jobs are resumable after an interruption by a non-availability

interval. The semi-resumable and non-resumable cases will be covered as future extensions. We also propose a Graham's experiment benchmark for this problem as well as two constraint programming models that outperform the MILP model. For larger problems, we propose using a Weighted Shortest Processing Time (WSPT) [69] heuristic and show that it quickly provides near-optimal solutions on the benchmark.

The rest of this thesis is organized as follow. Chapter 2 deals with the makespan minimization problem. It is defined and is modelled mathematically in its MILP and CP versions. Various numerical experiments are then conducted to test and compare several solution methods and improvements. Chapter 3 deals with the total weighted completion times problem. The proposed new MILP formulation is presented and numerical experiments conducted to compare the results obtained using the MILP and two CP versions. Lastly, Chapter 4 presents the conclusions along with some suggestions and areas for future study.

Chapter 2

Makespan Minimization Problem

In this chapter, we focus on the parallel machine scheduling problem with the objective of minimizing the makespan. The makespan problem is defined in section 2.1 and an existing MILP formulation is presented in 2.2. Our suggested contributions for improving the MILP formulation of the problem are introduced and evaluated in section 2.3. Two constraint programmings models are introduced in section 2.4 and compared to the MILP formulations in section 2.6.

2.1 Definition of the Makespan Problem

Given m parallel machines, N jobs with processing times p_j, r_i non-availability periods per machine i with corresponding start times s_{iq} and end times e_{iq} and rework factor α, the makespan minimization problem is to assign each job j to a machine i such that jobs assigned to the same machine do not overlap and the completion time of the job ending last (makespan) is minimized. Jobs can only be interrupted by non-availability periods (not preempted by other jobs). If a job is interrupted by a non-availability period, it must resume immediately after the end of the non-availability period having its remaining duration extended by a rework penalty equal to the portion of its processing time completed before interruption w_{iq} multiplied by the rework factor α (See Figure 2.1).

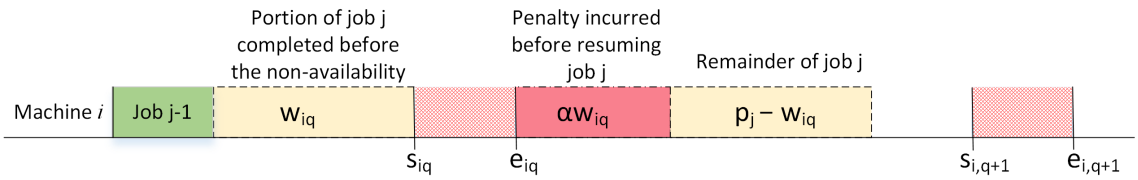

Figure 2.1: Illustration of the penalty incurred when a job is interrupted. Adapted from [7]

2.2 MILP Modelling of the Makespan Problem

We begin by reviewing the MILP model of Beaton et al. [7] for makespan minimization which we then extend with symmetry-breaking constraints and an improved lower bound. The model of Beaton et al. [7] makes the following assumptions that we also adopt throughout this thesis.

1. All machines are identical and can perform all operations.
2. Each machine can only process one job at a time.
3. Each job consists of only one operation.
4. The amount of time between consecutive non-availability periods must be at least as long as the longest processing time.
5. All jobs are available at time zero, however some machines may not be available at that time.
6. The non-availability periods are pre-determined and their durations are also known and constant.

The following notation is used in the formulation of the MILP [7].

i	Machine index
j	Job index
q	Non-availability period index
N	The total number of jobs
m	The total number of machines
r_i	The total number of non-availability periods on machine i
s_{iq}	The starting time of non-availability period q on machine i
e_{iq}	The ending time of non-availability period q on machine i
d_{iq}	The duration of non-availability period q on machine i ($d_{iq} = e_{iq} - s_{iq}$)
p_j	The processing time of job j
α	The resumability factor (between 0 and 1; 0 gives the fully resumable

case and 1 gives the non-resumable case)

C_{max} The makespan

M A sufficiently large positive number

The MILP models the case of minimizing the makespan of N jobs on m parallel machines where machine i has r_i non-availability periods. It uses a resumability factor α to specify the amount of rework time incurred for each job as a percentage of the processing time completed before each job is interrupted; thus allowing the model to cover fully-resumable and non-resumable scheduling as special cases of semi-resumable scheduling [48]. The amount of processing time completed prior to a job being interrupted by a non-availability period is given by w_{iq} and therefore, the amount of rework needing to be completed immediately after the end of the non-availability period is $\alpha \cdot w_{iq}$. Once the rework is completed, the job continues processing for the remaining portion of the job's original processing time. This results in a total processing time equal to $p_j + \alpha w_{iq}$ for each job. Model components for the MILP of Beaton et al. [7], including the objective function, decision variables, and constraints are as follows.

Decision Variables:

$$x_{ij} = \begin{cases} 1 & \text{if job } j \text{ is assigned to machine } i \\ 0 & \text{otherwise} \end{cases}$$

$$y_{iq} = \begin{cases} 1 & \text{if all jobs on machine } i \text{ are completed before } s_{iq} \\ 0 & \text{otherwise} \end{cases}$$

$$b_{ijq} = \begin{cases} 1 & \text{if job } j \text{ is completed before } s_{iq} \\ 0 & \text{otherwise} \end{cases}$$

w_{iq} = The amount of time between the completion of the last job before non-availability period q on machine i and the start of non-availability period q

The MILP is then as follows:

$$\min C_{max} \tag{2.1}$$

s.t.:

$$\sum_{j=1}^{N} p_j x_{ij} + \sum_{k=1}^{q-1} (d_{ik} + \alpha w_{ik}) \leq s_{iq} + M(1 - y_{iq}) \qquad \forall i, q \qquad (2.2)$$

$$\sum_{j=1}^{N} p_j x_{ij} + \sum_{q=1}^{r_i} (d_{iq}(1 - y_{iq}) + \alpha w_{iq}) \leq C_{max} \qquad \forall i \qquad (2.3)$$

$$\sum_{i=1}^{m} x_{ij} = 1 \qquad \forall i \qquad (2.4)$$

$$w_{iq} \geq s_{iq}(1 - y_{iq}) - \sum_{j=1}^{N} p_j b_{ijq} - \sum_{k=1}^{q-1} (d_{ik} + \alpha w_{ik}) \qquad \forall i, q \qquad (2.5)$$

$$w_{iq} \leq s_{iq} - \sum_{j=1}^{N} p_j b_{ijq} - \sum_{k=1}^{q-1} (d_{ik} + \alpha w_{ik}) \qquad \forall i, q \qquad (2.6)$$

$$b_{i,j,q+1} \geq b_{ijq} \qquad \forall i, j, q \qquad (2.7)$$

$$b_{ijq} \leq x_{ij} \qquad \forall i, j, q \qquad (2.8)$$

$$\sum_{j=1}^{N} (p_j b_{i,j,q+1} - p_j b_{ijq}) \leq s_{i,q+1} - e_{iq} - \alpha w_{iq} + w_{iq} \qquad \forall i, q \qquad (2.9)$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \qquad (2.10)$$

$$y_{iq} \in \{0, 1\} \qquad \forall i, q \qquad (2.11)$$

$$b_{ijq} \in \{0, 1\} \qquad \forall i, j, q \qquad (2.12)$$

$$C_{max} \geq 0 \qquad (2.13)$$

$$w_{iq} \geq 0 \qquad \forall i, q \qquad (2.14)$$

Constraint (2.2) sets y_{iq} equal to one if all jobs on machine i are completed before the start of non-availability period q. Constraint (2.3) excludes a non-availability period from the calculation of the makespan if all jobs scheduled on the machine are completed before the start of the non-availability period. Constraint (2.4) ensures that all jobs are assigned to exactly one machine. Constraints (2.5) and (2.6) define the amount of uncompleted work for each job that is interrupted by a non-availability period. The amount of uncompleted work will equal 0 in the case where all jobs are completed before the non-availability period, and consequently no rework will be added to the makespan in this case. Constraint (2.7) ensures that if a job is completed on a machine before a non-availability period, then it must also be completed

before the next non-availability period on that machine. Constraint (2.8) ensures that a job can only be completed before a non-availability period on a machine on which it is scheduled. Finally, constraint (2.9) ensures that only jobs that can be completed within the time preceding a non-availability period will be chosen to do so.

After running several numerical experiments, Beaton et al. [7] conclude that the MILP is difficult to solve optimally and suggest further developments. In what follows, several ideas such as adding new constraints, symmetry breaking and using CP will be investigated.

2.3 New contributions to improve the MILP

We now give a detailed description of our proposed additions to the MILP model of Beaton et al. [7].

2.3.1 Adding new constraints to the MILP

First, it should be noted that we experimented with adding the following three constraints to the model, but ultimately they were omitted since they did not significantly reduce runtime. However, constraint 2.15 may be particularly relevant to anyone attempting to apply decomposition methods to this problem because it helps avoid an invalid value of $y_{i,1}$ when no jobs are assigned to a machine with a non-availability period starting at time 0. This is a possible erroneous outcome when decomposing by machine which considers each machine's schedule in isolation.

$$\sum_{j=1}^{N} x_{ij} \geq 1 - y_{i,1} \qquad \forall i | s_{i,1} = 0 \tag{2.15}$$

$$y_{iq} \leq y_{i,q+1} \qquad \forall i, q \in 1..r_i - 1 \tag{2.16}$$

$$b_{ijq} \geq y_{iq} - (1 - x_{ij}) \qquad \forall i, j, q \tag{2.17}$$

Constraint (2.15) ensures that if the first non-availability period on a machine begins at time 0, and no jobs are scheduled on that machine, then all jobs are complete on that machine before the first non-availability period. Constraint 2.16 ensures that if all jobs are completed before a non-availability period, then all jobs are also

completed before subsequent non-availability periods. This extends the same rationale of constraint 2.7 to the y_{iq} variables. Constraint 2.17 ensures that if all jobs are completed before non-availability period q on machine i then either job j is not assigned to machine i, or job j is assigned to machine i such that it is completed before non-availability period q.

2.3.2 Adding symmetry breaking

Symmetry breaking is the process of eliminating solutions that are equivalent to one another conceptually yet are different in the implementation of a model [24]. Eliminating symmetries can help speed up the solving process by preventing a solver from wasting time visiting solutions that are equivalent to one another. Symmetry breaking in parallel machine scheduling is a commonly applied technique [33][39]. In the model of Beaton et al. [7], one type of symmetry occurs from the fact that all jobs can be swapped between any two identical machines and this will give an equivalent solution despite the fact that the x_{ij} variables will have their values swapped between the machines.

Two mutually-exclusive approaches to symmetry-breaking are proposed. The first is chaining lexicographic ordering constraints, a technique that is well-established in the constraint programming community [13], and the second is ordering identical machines in decreasing order of their individual machine makespans. For the first we use the linearized implementation of the “lex_chain_lesseq" global constraint [2] from the Minizinc constraint programming library [62]. The formulation requires defining the following new indices, parameters, and auxiliary variables.

Indices & Parameters:

G	The number of identical machine groups
g	Identical machine group index
V^g	The number of identical machine indices in machine group g
H^g	An ordered list of the subset of machine indices i belonging to machine group g (length(H^g) = V^g)

h The index of H^g. Used for indexing successive pairs of machine indices in H^g.

Decision Variables:

$$fleq_{g,i\in H_h^g|h<V^g,j} = \begin{cases} 1 & \text{if } x_{H_h^g,j} \text{ is equal to } x_{H_{h+1}^g,j} \\ & \text{or if both } x_{H_h^g,j} \text{ is less than } x_{H_{h+1}^g,j} \text{ and } flt_{g,H_h^g,j} \text{ equals 0} \\ 0 & \text{otherwise} \end{cases}$$

$$flt_{g,i\in H_h^g|h<V^g,j} = \begin{cases} 1 & \text{if } x_{H_h^g,j} \text{ is stricly less than } x_{H_{h+1}^g,j} \text{ and } fleq_{g,H_h^g,j} \text{ equals 0} \\ 0 & \text{otherwise} \end{cases}$$

Constraints:

$$\sum_{j=1}^{N} flt_{g,H_h^g,j} \leq 1 \qquad \forall g,h|h<V^g \qquad (2.18)$$

$$fleq_{g,H_h^g,1} + flt_{g,H_h^g,1} = 1 \qquad \forall g,h|h<V^g \qquad (2.19)$$

$$fleq_{g,H_h^g,j-1} = fleq_{g,H_h^g,j} + flt_{g,H_h^g,j} \qquad \forall g,h|h<V^g,j>1 \qquad (2.20)$$

$$x_{H_h^g,j} - x_{H_{h+1}^g,j} \leq 1 - fleq_{g,H_h^g,j} \qquad \forall g,h|h<V^g,j \qquad (2.21)$$

$$x_{H_{h+1}^g,j} - x_{H_h^g,j} \leq 1 - fleq_{g,H_h^g,j} \qquad \forall g,h|h<V^g,j \qquad (2.22)$$

$$x_{H_h^g,j} - x_{H_{h+1}^g,j} + 1 \leq 2 \cdot (1 - flt_{g,H_h^g,j}) \qquad \forall g,h|h<V^g,j \qquad (2.23)$$

The effect of the lexicographic constraints can be visualized as follows. For a given identical machine group g, construct an array A^g with N rows and V^g columns. Locate variable $x_{i,j}$ at row j and column h of A^g. Column h represents the x_{ij} variables of machine $i = H_h^g$. The lexicographic constraints require the columns of A^g to be ordered lexicographically from left to right. This breaks the symmetry of swapping the assignment of all x_{ij} solution values between columns of A^g. This can be visualized as follows for the case of $G = 2$, $V_1 = 3$, $V_2 = 2$, $H^1 = [1, 3, 5]$, $H^2 = [2, 4]$.

$$A^1 = \begin{bmatrix} x_{1,1} & x_{3,1} & x_{5,1} \\ x_{1,2} & x_{3,2} & x_{5,2} \\ \vdots & \vdots & \vdots \\ x_{1,N} & x_{3,N} & x_{5,N} \end{bmatrix}, A^2 = \begin{bmatrix} x_{2,1} & x_{4,1} \\ x_{2,2} & x_{4,2} \\ \vdots & \vdots \\ x_{2,N} & x_{4,N} \end{bmatrix}$$

The second, mutually-exclusive symmetry-breaking approach involves ordering identical machines in decreasing order of their individual machine makespans. To do this, we first note that the left hand side of constraint 2.3 represents the machine makespan of machine i. Therefore, we just need to add an inequality constraint for successive values of machine i:

$$\begin{aligned} &\sum_{j=1}^{N} p_j x_{H_h^g,j} + \sum_{q=1}^{r_{H_h^g}} (d_{H_h^g,q}(1 - y_{H_h^g,q}) + \alpha w_{H_h^g,q}) \\ &\leq \sum_{j=1}^{N} p_j x_{H_{h+1}^g,j} + \sum_{q=1}^{r_{H_{h+1}^g}} (d_{H_{h+1}^g,q}(1 - y_{H_{h+1}^g,q}) + \alpha w_{H_{h+1}^g,q}) \quad \forall g, h | h < V^g \end{aligned} \tag{2.24}$$

2.3.3 Adding lower bounds

Another addition to the model of Beaton et al. [7], is to calculate and add a lower bound on the makespan. The lower bound consists of dividing the sum of the job processing times across machines such that each machine has an equal machine makespan while accounting for non-availability periods. This is the equivalent of the well known lower bound for machines without non-availabilities that consists of averaging total processing time across all machines. This is not a new idea but providing a lower bound is a common practice when introducing a new formulation, or in this case when no lower bound had been described for the formulation.

The lower bound is computed by:

1. Partitioning the planning horizon by creating a section break at the start and end of each non-availability period;

2. Indexing the time interval between successive section breaks by index *sec*;

3. Calculating the duration of each interval *section_duration* and the number of available machines in each interval *m_avail_machines_in_section*.

4. Applying the pseudo-code given in Algorithm 1 below.

Algorithm 1: Calculate Makespan Lower Bound

```
Result: Makespan_LB
remaining_p = ∑_{j=1}^{N} p_j;
sec = 1;
lb = 0;
while remaining_p > 0 do
    section_p = m_avail_machines_in_section[sec] *
     section_duration[sec];
    if section_p <= remaining_p then
        lb += section_duration[sec];
        remaining_p -= section_p;
    else
        lb += remaining_p / m_avail_machines_in_section[sec];
        remaining_p = 0;
    end
    sec += 1;
end
```

This algorithm can be visualized with the help of Figure 2.2 for an example with five availability sections. The number of available machines in each section is (2, 2, 3, 2, 3) respectively, and the job processing times are (5, 5, 3, 3, 2). At each iteration the algorithm assigns an equal portion of the remaining total processing time to each available machine in the section up to an amount equal to the area of the section (available machines multiplied by section duration). If the remaining total processing time is less than the total area of the section then the remaining total processing time is evenly distributed among the available machines in the section. The lower bound is the sum of the section widths filled in this way.

One way to think of the end result is using the analogy of how a liquid (processing time) would fill in the space around obstacles (non-availability periods) in a container (machine schedules). Figure 2.3 provides a visual aid showing how the processing time could fill around the non-availability periods if the processing time of jobs was infinitesimally divisible like a liquid. In this figure the algorithm is broken into iterations for each job processing time to aid in understanding the analogy, but the algorithm does not require this and instead just uses the sum of all job processing times.

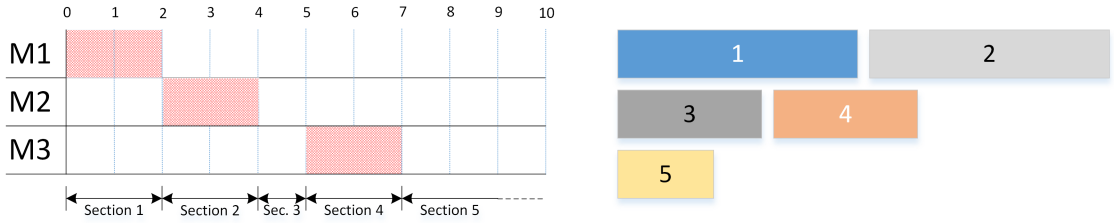

Figure 2.2: Availability Sections of Planning Horizon

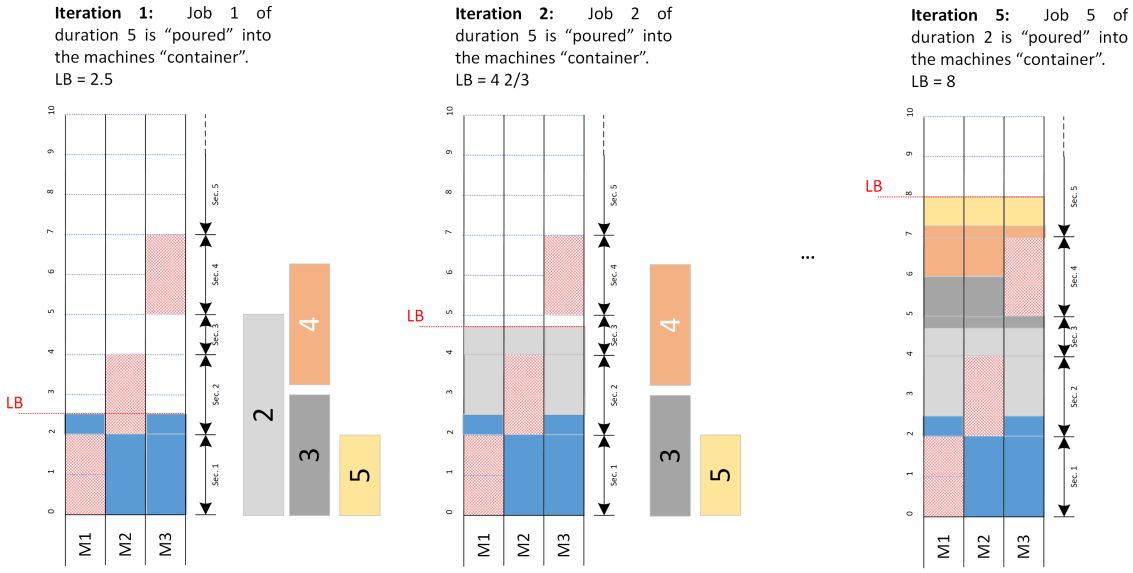

Figure 2.3: Liquid Processing Time Analogy

2.3.4 Using warm-start

Lastly, we also propose warm-starting the solving of the MILP using a good initial feasible solution from a constructive heuristic. Beaton et al. [7] proposed 4 heuristics (LPT2, LPT2-Swap, LPT2-GS1, and LPT2-GS2) based upon a modified version of the LPT2 longest processing time heuristic of Lee [47] for the general semi-resumable case with non-availability periods. Given that Beaton et al. [7] showed that the heuristics generally produce a good initial feasible solution quickly compared to the runtime of the MILP. We implement and sequentially run all four of LPT2, LPT2-Swap, LPT2-GS1, and LPT2-GS2. We use the best result from the four heuristics as the warm-start for the MILP.

2.3.5 MILP Experiments

In this section, we discuss the experiments conducted to evaluate the performance of our proposed additions to the MILP model. We chose to use instances of the modified Graham's Experiment benchmark of Beaton et al. [7] to facilitate comparison. Furthermore, to help minimize the effect of hardware improvements, all experiments were conducted on a 10 year old laptop with an Intel Core i7-2720QM CPU @ 2.20GHz and 8 GB of 1333MHz DDR3 RAM running Ubuntu 20.04.2 LTS. The MILP solver used was Gurobi 9.1.2, and all experiments used the best non-default parameter settings found during tuning: MIPFocus = 2 and Cuts = 3 which tell the solver to focus more attention on proving optimality and to apply very aggressive cut generation respectively [30]. A time limit of 1200 seconds was enforced for all instances.

The original Graham's Experiment [26] consists of m machines onto which $2m+1$ jobs must be scheduled. The jobs are given in decreasing order of processing time: $2m-1$, $2m-1$, $2m-2$, $2m-2$, ..., $m+1$, $m+1$, m, m, m. As the original experiment did not contain non-availability periods, Beaton et al. [7] proposed a modified version with $\lfloor \frac{m}{10} \rfloor$ non-availability periods separated by $2m$ time units. There are 2 groups of identical machines in the experiment: group 1 contains the machines with odd indices, and group 2 contains the machines with even indices. The first non-availability period of machines in group 1 starts at time 0 while the first non-availability period of machines in group 2 starts at time $2m$. To maintain consistency

with Beaton et al. [7], we test three levels of the resumability factor α: (0, 0.5, and 1).

First, the MILP model of Beaton et al. [7] was run without modification to establish a baseline denoted *Base*. Next, we added the lexicographic ordering constraints (2.18 - 2.23) to get model *LEX*. Model *LB* consists of *Base* plus the lower bound calculated by Algorithm 1 which gives the lower bounds in table 2.2. Model *LEX + LB* combines *Base* with both the lexicographic ordering constraints and the lower bound. Model *LB + UB* combines *Base* with the lower bound and warm-starts the MILP with the best solution from the four heuristics of Beaton et al. [7]. Finally, model *MM* combines *Base* with the machine makespan symmetry-breaking from constraint (2.24).

Table 2.1: Graham's Experiment (MILP): Time to Prove Optimal Solution

			Time (s)					
m	N	α	Base	Lex	LB	Lex+LB	LB+UB	MM
10	21	0	0.14	1.01	**0.13**	0.24	0.19	0.43
15	31		1.79	3.57	1.12	**0.66**	0.94	2.20
20	41		**7.26**	20.68	11.07	12.33	9.06	20.56
25	51		**11.34**	28.31	160.95	307.68	233.38	92.81
10	21	0.5	1.03	2.45	1.29	1.49	**0.78**	10.76
15	31		8.42	10.76	2.52	8.93	**1.4**	415
20	41		–	761	**55**	775	863	–
25	51		–	–	–	–	–	–
10	21	1	2.25	6.81	2.56	6.13	**1.95**	35.22
15	31		50.86	180.29	**11.39**	225.21	23.78	–
20	41		–	–	–	–	–	–
25	51		–	–	–	–	–	–

*Values given as "−" did not prove optimality within 1200 seconds.

Table 2.1 shows the time to prove optimality whereas table 2.3 shows the time to reach the optimal solution (when the optimal solution has already been determined).

The results in Table 2.1 suggest that the fully-resumable case ($\alpha = 0$) is the easiest variation of the problem; all models are able to prove optimality within the 1200

Table 2.2: Graham's Experiment (MILP): Improved Lower Bounds

m	N	Lower Bound
10	21	32
15	31	48
20	41	67.5
25	51	84.12

second time limit, even for the 25 machine problem. This is consistent with the findings of Beaton et al. [7]. The results suggest that the problem becomes more difficult for the MILP formulations with increasing α, which is information that was not clear from the limited previous results. This is supported by the increased runtimes and the fact that none of the models were able to prove optimality beyond 15 machines when $\alpha = 1$.

Our proposed additions do appear to improve the model if only modestly, with four out of the five combinations of additions proving optimality of the 20 machine problem with $\alpha = 0.5$, an instance that remained open. There appears to be additive effects when combining various model additions: while model LB was able to prove optimality of the 20 machine, $\alpha = 0.5$ instance in only 55 seconds, its progress slowed for larger instances when paired with LEX and the heuristic upper bound. Overall, no one model dominated the others, but lower bounding looks promising, and lexicographic ordering did succeed in reducing the number of explored branches in the search tree; although this did not lead to a faster proof of optimality in all cases. The machine makespan symmetry breaking performed worst in general, increasing the solution time in most cases. One possibility why the machine makespan symmetry breaking might have slowed the solve time is because it changed the cuts that the Gurobi solver was able to generate and cutting planes play a major role in the efficient solution of this problem. Further investigation may be needed before concluding that machine makespan symmetry breaking cannot be helpful.

The results in Table 2.3 further show that even when the MILP models were not able to prove optimality, they were frequently able to determine the optimal solution within a relatively short amount of time.

Table 2.3: Graham's Experiment (MILP): Time to Reach Optimal Solution

			CPU Time (s)					
m	N	α	Base	Lex	LB	Lex+LB	LB+UB	MM
10	21	0	0.14	1.01	**0.13**	0.24	0.19	0.43
15	31		1.79	3.57	1.12	**0.66**	0.94	2.2
20	41		7.26	20.68	9	**3**	9.06	20.56
25	51		**4**	20	11	71	9	92.81
10	21	0.5	1.03	< 1	1.29	< 1	< 1	7
15	31		8.42	10.76	**1**	8.93	1.4	415
20	41		52	96	39	42	**15**	–
25	51		–	–	–	–	–	–
10	21	1	< 1	< 1	< 1	< 1	< 1	4
15	31		1	4	1	2	1	96
20	41		–	–	–	–	–	–
25	51		–	–	–	–	–	–

*Values given as "−" did not achieve known optimal objective value within 1200 seconds, or optimal objective value is unknown.

2.4 CP Modelling of the Makespan Problem

Two different approaches to modelling the makespan minimization problem using constraint programming are developed. The first is a traditional CP scheduling approach using specialized global constraints for scheduling and explicit interval variables. The second is a verbatim translation of the *Base* MILP model into constraint programming.

In order to model the problem using CP, we must restrict ourselves to instances where decision variables are guaranteed to take on integer values. This initially appears to present a problem for the semi-resumable case since, for example, $\alpha = 0.5$ will result in a fractional amount of rework any time that the amount of processing time completed before a non-availability period assumes an odd value. The number of decimals required to represent a solution exactly could also increase with each additional non-availability period overlapped on a machine. For example, given a single-machine problem with $\alpha = 0.5$, non-availability period start times $s_{1,1} = 3$, $s_{1,2} = 8$, end times $e_{1,1} = 4$, $e_{1,2} = 9$, and job processing times $p_1 = 5$, $p_2 = 5$ an

optimal solution involves scheduling job 1 at time 0 resulting in $w_{1,1} = 3$ units of processing time prior to interruption, and therefore $\alpha \cdot w_{1,1} = 0.5 \cdot 3 = 1.5$ units of rework. This causes job 2 to begin at fractional time unit $3 + 1 + 1.5 + 2 = 7.5$, $w_{1,2} = 0.5$, $\alpha \cdot w_{1,2} = 0.5 \cdot 0.5 = 0.25$ units of rework, and therefore job 2 ends at fractional time unit $7.5 + 0.5 + 1 + 0.25 + 4.5 = 13.75$. As a solution to this problem, we note that α is most commonly an approximation in practice, and typically can be represented with sufficient accuracy by a rational number. By noting that the number of decimals is a function of the maximum number of non-availability periods on any machine, we then propose to apply the following scaling factor to the time dimension:

$$\Phi = \max_{1 \leq i \leq m} (\alpha_{den}^{r_i}) \cdot (\alpha_{num}^{r_i}) \tag{2.25}$$

where α_{num} and α_{den} are the numerator and denominator of the rational number α respectively when given as a fraction.

In the MILP model, many types of unnecessary symmetry were avoided by omitting any explicit notion of ordering between the jobs on a machine. In this traditional CP formulation, we include these symmetries by representing each job with an explicit start, end, duration, and interval variable [45]. In addition to making the model easier to adapt to different objective functions, this allows the use of powerful scheduling constraints over the interval variables. Interval variables are both a useful modelling abstraction that encapsulates the relation between start, end, and duration variables and helps define precedence constraints when they exist; as well as a means to allow the solver to perform additional reasoning. Interval variables can be made optional by pairing them with the $x_{i,j}$ variables to cover the case where a job may not be present on a particular machine [43][44]. Many powerful global constraints exist for scheduling problems that are a major contributor to the success of CP models for scheduling[70]. CP solvers can use specialized data structures to help global scheduling constraints efficiently propagate filtered domain values[71]. The disjunctive scheduling global constraint (also known as the NoOverlap constraint) takes a list of (optional) interval variables representing jobs and ensures that none of the interval variables assigned to

a machine overlap[12].

A numerical example [1] from the global constraint catalogue [9][8] showing the effect of the disjunctive constraint is as follows: Let ($s_1 \in [1,4]$, $s_2 \in [1,3]$, $s_3 \in [2,5]$, $s_4 \in [1,6]$) and ($d_1 \in [2,4]$, $d_2 \in [1,6]$, $d_3 \in [4,4]$, $d_4 \in [1,3]$) be job start time and duration variables respectively. The disjunctive global constraint would result in the four (s_j, d_j) feasible solutions of non-overlapping jobs on a single machine as given in Table 2.4 and depicted in Figure 2.4.

Table 2.4: Example Feasible Solutions: Disjunctive Constraint

Feasible Solution	s_1	d_1	s_2	d_2	s_3	d_3	s_4	d_4
1	1	2	3	1	5	4	4	1
2	2	2	1	1	5	4	4	1
3	3	2	1	1	5	4	2	1
4	3	2	2	1	5	4	1	1

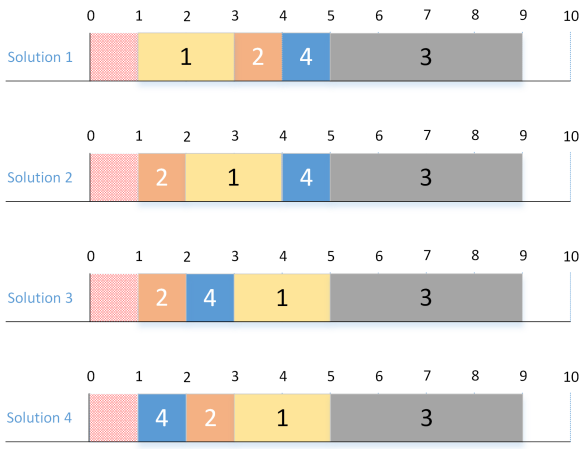

Figure 2.4: Disjunctive Global Constraint Example Solutions
*Adapted From: https://sofdem.github.io/gccat/ctrs/disjunctive-2-tikz.png

The following additional parameters, decision variables are introduced and used to develop a traditional CP scheduling model using specialized global constraints for scheduling and explicit interval variables.

Indices & Parameters:

α_{num}	The numerator of α when α is expressed as a rational fraction
α_{den}	The denominator of α when α is expressed as a rational fraction
Φ	Scaling factor to ensure integer decision variable values
ψ_{ijq}	Earliest start time of job j that results in interruption by non-availability period q
	*Other indices & parameters same as MILP model

Decision Variables:

C_{max}	Makespan
$x_{ij} = \begin{cases} 1 & \text{if job } j \text{ is assigned to machine } i \\ 0 & \text{otherwise} \end{cases}$	
dur_rw_{ij}	Rework duration
$dur\_rw_{ij}^{num}$	Rework duration numerator (intermediate variable only defined if $\alpha_{num} \neq 1$)
st_{ij}	Start time of job j if assigned to mahine i
et_{ij}	End time without rework
dur_{ij}	Duration without rework
$Intrv_{ij}$	Interval variable for portion of job without rework
w_{ij}	Processing time of job j completed before a non-availability period
et_rw_{ij}	End time with rework
$Intrv\_rw_{ij}$	Interval variable for rework portion of job
$gte_{ijq}^{\psi} = \begin{cases} 1 & \text{if job } j \text{ is assigned to machine } i \text{ and } st_{ij} \geq \psi_{ijq} \\ 0 & \text{otherwise} \end{cases}$	

$$lteq_{ijq} = \begin{cases} 1 & \text{if job } j \text{ is assigned to machine } i \text{ and starts before} \\ & \text{non-availability period } q \\ 0 & \text{otherwise} \end{cases}$$

$$qhas\_rw_{ijq} = \begin{cases} 1 & \text{if job } j \text{ is assigned to machine } i \text{ and is interrupted} \\ & \text{by non-availability period } q \\ 0 & \text{otherwise} \end{cases}$$

$$has\_rw_{ij} = \begin{cases} 1 & \text{if job } j \text{ is assigned to machine } i \text{ and is interrupted} \\ 0 & \text{otherwise} \end{cases}$$

$$no\_rw_{ij} = \begin{cases} 1 & \text{if job } j \text{ is assigned to machine } i \text{ and is not interrupted} \\ 0 & \text{otherwise} \end{cases}$$

$$\min C_{max}$$

s.t.:

$$\sum_{i=1}^{m} x_{ij} = 1 \qquad \forall i,j \quad (2.26)$$

$$\neg x_{ij} \implies \neg has\_rw_{ij} \qquad \forall i,j \quad (2.27)$$

$$x_{ij} \implies st_{ij} + dur_{ij} = et_{ij} \qquad \forall i,j \quad (2.28)$$

$$x_{ij} \implies et_{ij} + dur\_rw_{ij} = et\_rw_{ij} \qquad \forall i,j \quad (2.29)$$

$$no\_rw_{ij} \Leftrightarrow (x_{ij} \wedge \neg has\_rw_{ij}) \qquad \forall i,j \quad (2.30)$$

$$\neg gte^{\psi}_{i,j,1} \wedge \neg lteq_{i,j,1} \wedge \neg qhas\_rw_{i,j,1} \qquad \forall i | s_{i,1} = 0, j \quad (2.31)$$

$$gte^{\psi}_{ijq} \Leftrightarrow (st_{ij} \geq \Phi \cdot \psi_{ijq}) \qquad \forall i,j,q | s_{i,q} > 0 \quad (2.32)$$

$$lteq_{ijq} \Leftrightarrow (st_{ij} \leq \Phi \cdot s_{iq}) \qquad \forall i,j,q | s_{i,q} > 0 \quad (2.33)$$

$$qhas\_rw_{ijq} \Leftrightarrow (gte^{\psi}_{ijq} \wedge lteq_{ijq}) \qquad \forall i,q | s_{i,q} > 0 \quad (2.34)$$

$$\text{if } \alpha_{num} \neq 1 : dur\_rw^{num}_{ij} = \alpha_{num} \cdot w_{ij} \qquad \forall i,j \quad (2.35)$$

$$\text{if } \alpha_{num} \neq 1 : dur\_rw_{ij} = \frac{dur\_rw^{num}_{ij}}{\alpha_{den}} \qquad \forall i,j \quad (2.36)$$

$$\text{if } \alpha_{num} = 1 : dur\_rw_{ij} = \frac{w_{ij}}{\alpha_{den}} \qquad \forall i, j \quad (2.37)$$

$$no\_rw_{ij} \implies (dur_{ij} = \Phi \cdot p_j) \qquad \forall i, j \quad (2.38)$$

$$qhas\_rw_{ijq} \implies (dur_{ij} = \Phi \cdot (p_j + d_{iq})) \qquad \forall i, j, q \quad (2.39)$$

$$\neg has\_rw_{ij} \implies (w_{ij} = 0) \qquad \forall i, j \quad (2.40)$$

$$qhas\_rw_{ijq} \implies (w_{ij} = \Phi \cdot s_{iq} - st_{ij}) \qquad \forall i, j, q \quad (2.41)$$

$$disjunctive([Intrv_{ij}, Intrv\_rw_{ij}]) \qquad \forall i \quad (2.42)$$

$$has\_rw_{ij} \implies (\sum_{q=1}^{r_i} qhas\_rw_{ijq} = 1) \qquad \forall i, j \quad (2.43)$$

$$\neg has\_rw_{ij} \implies (\sum_{q=1}^{r_i} qhas\_rw_{ijq} = 0) \qquad \forall i, j \quad (2.44)$$

$$C_{max} \geq 0 \qquad (2.45)$$

$$dur\_rw_{ij}, dur\_rw_{ij}^{num}, st_{ij}, et_{ij}, dur_{ij}, w_{ij}, et\_rw_{ij} \geq 0 \qquad \forall i, j \quad (2.46)$$

$$has\_rw_{ij}, no\_rw_{ij} \in \{0, 1\} \qquad \forall i, j \quad (2.47)$$

$$gte_{ijq}^{\psi}, lteq_{ijq}, qhas\_rw_{ijq} \in \{0, 1\} \qquad \forall i, j, q \quad (2.48)$$

where $\wedge$, $\vee$, $\neg$, $\implies$, $\Leftrightarrow$ are logical AND, OR, NOT, IMPLICATION, and EQUALITY respectively.

Constraints (2.26) ensure that each one of the jobs is assigned to exactly one machine. Constraints (2.27) ensure that jobs do not have rework on machines to which they are not assigned. Constraints (2.28) and (2.29) are defined implicitly by the creation of the interval variables and define the relation between start time, duration, and end time of non-rework and rework interval variables. Constraints (2.30) implement the definition of the no_rw_{ij} Boolean variable. Constraints (2.31) ensure no jobs can be labelled as being interrupted by a non-availability period starting at time 0. This is primarily a modelling convenience as these Boolean variables are removed by the CP Presolve. Constraints (2.32), (2.33) and (2.34) implement the definition of gte_{ijq}^{ψ}, $lteq_{ijq}$ and $qhas\_rw_{ijq}$ respectively. Constraints (2.35), (2.36) and (2.37) calculate the amount of rework for a job. They are separate constraints because most CP solvers will not allow multiplication by a fraction since this could result in

fractional decision variable values. The division by α_{dem} actually performs integer division, but this is not a problem because we are a priori guaranteed an integer solution because of the scaling factor Φ. Constraints (2.38) and (2.39) restrict the non-rework portion of the job to equal one of only three values: 0 if the job is not assigned to the machine, the processing time if the job is not interrupted, or the processing time plus the non-availability period duration if interrupted. Constraints (2.40) link the Boolean no_rw_{ij} variable to the integer rework variable. Constraints (2.41) link job rework to job start time. Constraints (2.42) ensure that all jobs on a machine do not overlap. Constraints (2.43) and (2.44) ensure a job is defined as overlapping at most one non-availability period and links to the corresponding indicator Boolean variables. Constraints (2.45 – 2.48) are binary and non-negativity constraints.

The second CP model is a direct translation of the *Base* MILP model into CP. To do this, we use the same scale-factor Φ to ensure integer decision variable values, but otherwise the model is the same.

2.5 CP Experiments

To test the effectiveness of the CP formulation of the makespan minimization problem, we construct and solve equivalent instances of the same Graham's Experiment benchmark as the MILP experiments. The CP models were solved using the CP-SAT solver of Google ORTools version 9.0.9163 [65] on the same laptop as the MILP experiments. The solver parameter *num_search_workers* was set to 8 which causes a portfolio of different solving approaches to be executed across 8 different cores in parallel.

The results in Table 2.5 show that the traditional CP model is unable to prove optimality in even the easiest instances whereas the verbatim CP translation of the *Base* MILP model proves optimality in at least one instance for each value of α. However, Table 2.6 shows that despite not proving optimality, both CP models had found the optimal solution for many of the instances within a relatively short amount of time.

Table 2.5: Graham's Experiment (CP): Time To Prove Optimal Solution

			CPU Time (s)	
m	N	α	Trad	Verb
10	21	0	–	0.91
15	31		–	4.56
20	41		–	–
25	51		–	–
10	21	0.5	–	17.59
15	31		–	749.48
20	41		–	–
25	51		–	–
10	21	1	–	31.51
15	31		–	–
20	41		–	–
25	51		–	–

*Values given as "−" did not prove optimality within 1200 seconds.

Table 2.6: Graham's Experiment (CP): Time to Reach Optimal Solution

			CPU Time (s)	
m	N	α	Trad	Verb
10	21	0	0.99	0.91
15	31		12.67	4.55
20	41		94.82	–
25	51		25.13	30.27
10	21	0.5	0.71	0.25
15	31		8.66	6.69
20	41		28.02	5.64
25	51		–	–
10	21	1	1.83	0.08
15	31		5.23	6.64
20	41		–	–
25	51		–	–

*Values given as "−" did not achieve known optimal objective value within 1200 seconds, or optimal objective value is unknown.

2.6 Comparison of MILP & CP

Having run identical experiments on both formulation types, we are in a position to compare the strengths and weaknesses of both approaches for this particular scheduling problem. From Table 2.7, we can see that the combination of the commercial Gurobi solver and the MILP formulations are the clear winner when it comes to proving optimality. On the other hand, Table 2.8 shows that both MILP and CP have similar performance for the time required to find (but not prove) an optimal solution, especially for the $\alpha > 0$ cases. One downside of the CP model is the requirement to estimate an upper bound on the required length of the planning horizon (upper bound on makespan variable and job start and end times). This can lead to large finite domains which can cause the model to use significantly more memory than the MILP models. Fortunately, the good performance of constructive heuristics could go a long way towards keeping this a priori upper bound small and thus reducing memory consumption. Both the MILP and CP solvers labelled many solutions as having been found by heuristics.

Table 2.7: Graham's Experiment - Time to Prove Optimality

			CPU Time (s)							
			MILP						CP	
m	N	α	Base	Lex	LB	Lex+LB	LB+UB	MM	Trad	Verb
10	21	0	0.14	1.01	**0.13**	0.24	0.19	0.43	–	0.9
15	31		1.79	3.57	1.12	**0.66**	0.94	2.2	–	4.56
20	41		**7.26**	20.68	11.07	12.33	9.06	20.56	–	–
25	51		**11.34**	28.31	160.95	307.68	233.38	92.81	–	–
10	21	0.5	1.03	2.45	1.29	1.49	**0.78**	10.76	–	17.59
15	31		8.42	10.76	2.52	8.93	**1.4**	415	–	749.48
20	41		–	761	**55**	775	863	–	–	–
25	51		–	–	–	–	–	–	–	–
10	21	1	2.25	6.81	2.56	6.13	**1.95**	35.22	–	31.51
15	31		50.86	180.29	**11.39**	225.21	23.78	–	–	–
20	41		–	–	–	–	–	–	–	–
25	51		–	–	–	–	–	–	–	–

*Values given as "−" did not prove optimality within 1200 seconds.

Table 2.8: Graham's Experiment - Time to Reach Optimality

			CPU Time (s)							
			MILP						CP	
m	N	α	Base	Lex	LB	Lex+LB	LB+UB	MM	Trad	Verb
10	21	0	0.14	1.01	**0.13**	0.24	0.19	0.43	0.99	0.9
15	31		1.79	3.57	1.12	**0.66**	0.94	2.2	12.67	4.55
20	41		7.26	20.68	9	**3**	9.06	20.56	94.82	–
25	51		**4**	20	11	71	9	92.81	25.13	30.27
10	21	0.5	1.03	< 1	1.29	< 1	< 1	7	0.7	**0.25**
15	31		8.42	10.76	**1**	8.93	1.4	415	8.66	6.69
20	41		52	96	39	42	15	–	28.02	**5.64**
25	51		–	–	–	–	–	–	–	–
10	21	1	< 1	< 1	< 1	< 1	< 1	4	1.83	0.08
15	31		1	4	1	2	1	96	5.23	6.64
20	41		–	–	–	–	–	–	–	–
25	51		–	–	–	–	–	–	–	–

*Values given as "−" did not achieve known optimal objective value within 1200 seconds, or optimal objective value is unknown.

Chapter 3

Total Weighted Completion Time Minimization

In this chapter, we discuss the problem of solving parallel machine scheduling problems with non-availability periods for the case of minimizing total weighted completion time (TWC). We only deal with the resumable case and leave the non-resumable and semi-resumable cases as future extensions. TWC models situations where jobs have been assigned priorities.

3.1 Definition of the Total Weighted Completion Time Problem

Conceptually, the minimum TWC version of the problem is identical to the minimum makespan version except that instead of minimizing the end time of the job finishing last, we are instead in the situation that each job has an associated weight w_j and the objective is to minimize the sum:

$$TWC = \sum_{j=1}^{N} w_j \cdot C_j \tag{3.1}$$

Although this may seem like a small change, it requires a completely different formulation approach to model the problem.

3.2 Constructive Heuristic

It is well known that the weighted shortest processing time (WSPT) heuristic is optimal for the *single* machine scheduling problem with TWC [69]. Therefore, this heuristic could be used as a method for generating an initial feasible solution for the parallel machine scheduling problem with non-availability periods to minimize TWC. The idea of using a variation of the WSPT heuristic for parallel machine scheduling is not new but is commonly coupled with the introduction of new formulations. Using the same notation as Beaton et al. [7] for describing the LPT2 heuristic of Lee [47],

we let L_i be the load (current latest end time of assigned jobs) of machine i. The next job j will be scheduled on machine i^* such that:

$$i^* = \begin{cases} argmin\{L_i + p_j : i = 1...m\} & \text{if } L_i + p_j \leq s_{iq} \\ argmin\{L_i + p_j + d_{iq} + \alpha(s_{iq} - L_i) : i = 1...m\} & \text{if } L_i + p_j > s_{iq} \end{cases} \tag{3.2}$$

The TWC can be calculated using Eq. (3.1) once all jobs have been assigned to a machine. The version of the WSPT heuristic for this problem is then given by the following algorithm:

WSPT:

- *Step 1:* Sort N jobs in decreasing order of $\frac{w_j}{p_j}$
- *Step 2:* Set $j = 1$, resulting in the WSPT job selection rule
- *Step 3:* Assign job j to machine i^* using the LPT2 machine selection rule in Eq. (3.2)
- *Step 4:* if $j = N$ then go to the next step, otherwise set $j = j + 1$ and go to Step 3
- *Step 5:* Calculate TWC by using Eq. (3.1)

3.3 MILP Modelling of the Total Weighted Completion Time Problem

In this section we propose a model for TWC minimization combining the position-based modelling approach by Mönch and Shen [60] and the non-availability modelling approach by Beaton et al. [7]. Assumptions are as in the makespan minimization model.

We use the same notation for the formulation of the TWC MILP model as in the makespan MILP model with the addition of the following parameter:

w_j: weight of job j.

Decision variables:

$$x_{j,i} = \begin{cases} 1 & \text{if job } j \text{ is assigned to machine } i \\ 0 & \text{otherwise} \end{cases}$$

$$z_{j,k} = \begin{cases} 1 & \text{if job } j \text{ is a predecessor of job } k \\ 0 & \text{otherwise} \end{cases}$$

$$b_{i,j,q} = \begin{cases} 1 & \text{if job } j \text{ is completed } \underline{\textbf{\textit{after}}} \text{ non-availability } q \text{ on machine } i \\ 0 & \text{otherwise} \end{cases}$$

$S_j =$ Start time of job j

$C_j =$ End time of job j

The proposed MILP model is given by:

$$\min \sum_{j=1}^{N} w_j \cdot C_j \tag{3.3}$$

s.t.:

$$\sum_{i=1}^{m} x_{ji} = 1 \qquad \forall j \tag{3.4}$$

$$S_k + p_k - M \cdot (2 - x_{j,i} - x_{k,i} + z_{j,k}) \leq S_j \qquad \forall i, j, k (k \neq j) \tag{3.5}$$

$$S_j + p_j - M \cdot (3 - x_{j,i} - x_{k,i} - z_{j,k}) \leq S_k \qquad \forall i, j, k (k \neq j) \tag{3.6}$$

$$S_j + p_j \cdot x_{j,i} + \sum_{q=1}^{r_i} d_{i,q} \cdot b_{i,j,q} \leq C_j \qquad \forall j, i \tag{3.7}$$

$$C_j \leq s_{i,q} + M \cdot b_{i,j,q} \qquad \forall i, j, q \tag{3.8}$$

$$b_{i,j,q+1} \leq b_{i,j,q} \qquad \forall i, j, q \leq r_i \tag{3.9}$$

$$x_{j,i},\ z_{j,k},\ b_{i,j,q} \in \{0, 1\} \qquad \forall i, j, k, q \tag{3.10}$$

$$S_j,\ C_j \geq 0 \qquad \forall j \tag{3.11}$$

Constraints (3.4) ensure each job is assigned to exactly one machine. Constraints (3.5) and (3.6) define the precedence relations between jobs. Constraints (3.7) define

the completion time of jobs. The completion time of job j is at least equal to the sum of its starting time on a given machine, its processing time if assigned to that machine, and the duration of all nonavailability periods that occur between the start of the job and its completion. Constraints (3.8) determine whether a job completes before or after a non-availability period. Constraints (3.9) state that if a job is completed after a non-availability period, then it is also completed after preceding non-availability periods. Constraints (3.9) are not required but are used to reduce computation times by the same mechanism as constraints (2.7) in the makespan model. Constraints (3.10) and (3.11) are binary and non-negativity constraints respectively.

3.4 MILP Experiments

In this section, we discuss the experiments conducted to evaluate the performance of our proposed MILP model. We start by showing results for examples from Mellouli et al. [59] used to verify the model correctness. Next, we propose a new modified Graham's Experiment benchmark for the TWC problem. As with the makespan problem, all experiments were conducted on a 10 year old laptop with an Intel Core i7-2720QM CPU @ 2.20GHz and 8 GB of 1333MHz DDR3 RAM running Ubuntu 20.04.2 LTS. The MILP solver used was Gurobi 9.1.2, and all experiments used the best non-default parameter settings found during tuning: MIPFocus = 2 and Cuts = 3 which tell the solver to focus more attention on proving optimality and to apply very aggressive cut generation respectively [30]. A time limit of 1200 seconds was enforced for all instances.

3.4.1 Experiment #1: Validation example

The goal of the first numerical example is to verify the correctness of our proposed MILP model. It is tested on a problem published in the literature: an instance from Mellouli et al. [59]. This instance involves 3 machines, 10 jobs, unit weights, and the processing times in Table 3.1 below. Each machine has only one non-availability period as depicted in Figure 3.1. This is one shortcoming of the Mellouli et al. [59] paper.

Table 3.1: Instance from Mellouli et al. [59]

Jobs j	1	2	3	4	5	6	7	8	9	10
Processing Times p_j	1	1	1	2	3	3	4	4	5	5

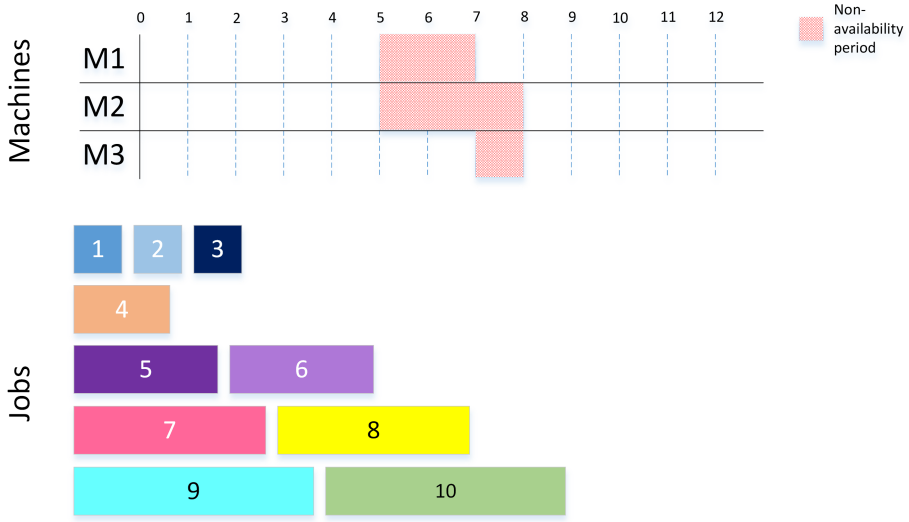

Figure 3.1: Mellouli Example - Setup

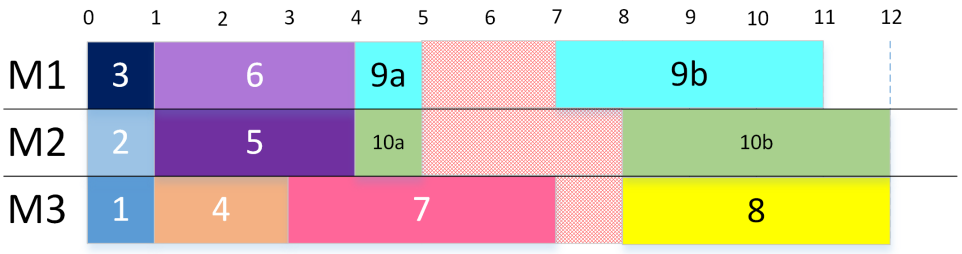

Figure 3.2: Instance from Mellouli - Solution

Our model obtains the optimal solution $TWC^* = 56$, which is represented in Figure 3.2

3.4.2 Experiment #2: Impact of altering job weights

The next example shows how changing the weights affects the solution. It involves 2 machines, 5 jobs, and multiple non-availability periods per machine. The processing times are given in Table 3.2. Initially unit weights are used followed by the weights in Table 3.2.

Table 3.2: Altered Job Weights

Jobs j	1	2	3	4	5
Processing Times p_j	5	5	3	3	2
Weights w_j	3	1	1	3	1

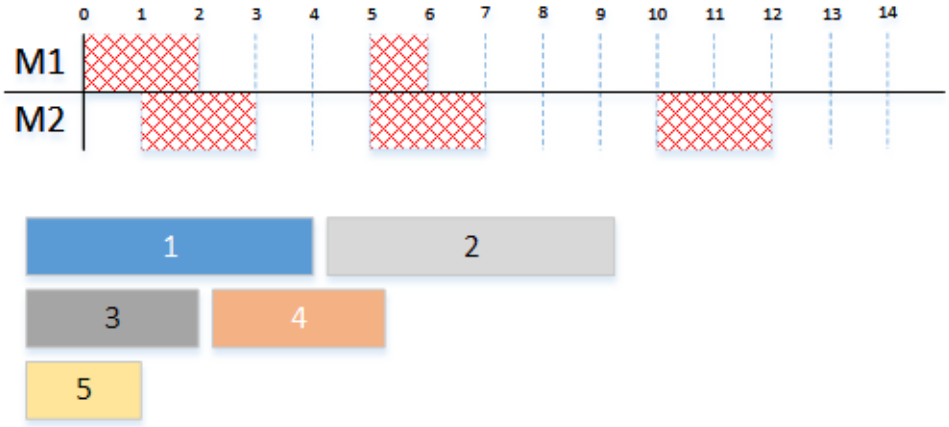

Figure 3.3: Setup for second experiment

From the results in Figures 3.3 and 3.4, we can see that when we increase the weights for jobs 1 and 4, the model moves them earlier in the schedule as indicated by the red arrows attached at the completion times. This example also shows that our formulation is capable of dealing with multiple non-availability periods per machines. The solutions depict how jobs are split by non-availability periods.

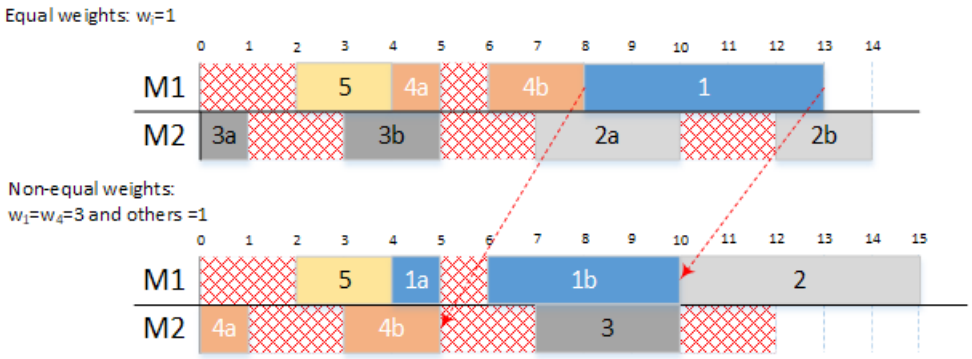

Figure 3.4: Altered Job Weights - Solutions

3.4.3 Experiment #3: Graham's Experiment for weighted completion times

A modified version of Graham's Experiment is implemented for the total weighted completion time problem. The original Graham's Experiment [26] consists of m machines onto which $2m+1$ jobs must be scheduled. The jobs are given in decreasing order of processing time: $2m-1$, $2m-1$, $2m-2$, $2m-2$, ..., $m+1$, $m+1$, m, m, m. As the original experiment did not contain non-availability periods, we propose a modified version with $\lfloor \frac{m}{2} \rfloor$ non-availability periods separated by $2m$ time units. Each non-availability period has a duration lasting $\lfloor \frac{m}{2} \rfloor$ time units. Note that this is different from the makespan problem which used $\lfloor \frac{m}{10} \rfloor$ non-availability periods, each with a duration lasting $\lfloor \frac{m}{4} \rfloor$ time units. There are 2 groups of identical machines in the experiment: group 1 contains the machines with odd indices, and group 2 contains the machines with even indices. The first non-availability period of machines in group 1 starts at time 0 while the first non-availability period of machines in group 2 starts at time $2m$. The weights for machines are calculated as:

$$w_j = \begin{cases} \lfloor (j+1)^{\frac{3}{4}} \rfloor + 2 & \forall j \leq \lfloor \frac{N}{2} \rfloor - 1 \\ \lfloor (N-j)^{\frac{3}{4}} \rfloor + 2 & \forall \lfloor \frac{N}{2} \rfloor - 1 < j < N \\ 3 & j = N \end{cases} \tag{3.12}$$

This formula for the weights was chosen because it scales well for large numbers of jobs and machines, puts the most weight on the jobs with processing times near the mean, and gives similar weights to short and long jobs with processing times equal

distances from the mean.

Table 3.3: Graham's Experiment (MILP): Total Weighted Completion Time

		CPU Time (s)			
m	N	LB	UB	Time	$Time_{UB}$
2	5	75	75	0.09	< 0.09
3	7	167	167	3.37	< 2
4	9	284.09	332	1200	1
5	11	348.63	533	1200	6
6	13	488	817	1200	22

*Values were recorded after 1200 seconds of run time

Table 3.3 shows the lower bound and upper bound achieved within the time given in the time column (time limit of 1200 seconds, or shorter if optimality was proven). The $Time_{UB}$ column gives the earliest time at which the upper bound was achieved. The results in Table 3.3 show that we are only able to solve TWC problems to provable optimality for 3 machines and 7 jobs using our proposed MILP formulation. This is in contrast to 15 or more machines and 31 or more jobs for the makespan benchmark. This speaks to the difficulty of the problem. It is known from other experiments that 332 is the optimal objective value for the 4 machine, 9 job instance. Given that the MILP model was able to find this solution within approximately 1 second, and that the 5 and 6 machine instances stop progressing the upper bound after a similarly short amount of time, this suggests that the MILP model may be able to quickly find the optimal solution for small instances even if it is not able to prove optimality, similarly to the makespan model. Figure 3.5 shows an optimal solution to the 4 machine, 9 job instance obtained after 7342.85 seconds.

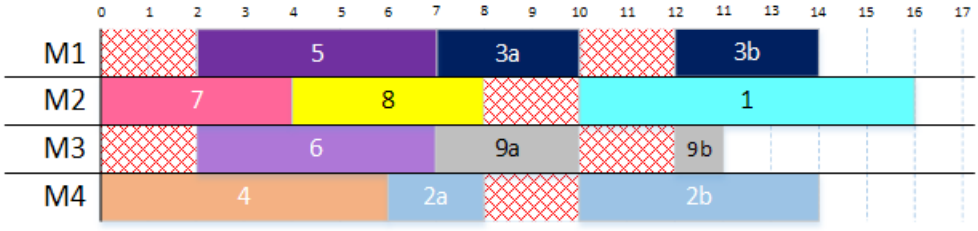

Figure 3.5: Total Weighted Completion Time - 4 Machines, 9 Jobs - Optimal Solution

3.5 CP Modelling of the Total Weighted Completion Time Problem

We propose two different formulation approaches to modelling the minimum TWC problem using constraint programming. The first is a traditional CP scheduling approach using specialized global constraints for scheduling and explicit interval variables. The second is a verbatim translation of the MILP model into constraint programming.

One of the benefits of the traditional CP scheduling model for the makespan problem is that absent the objective function it is a basis for parallel machine scheduling problems with non-availability periods and arbitrary resumability in general. This means that with minor additions we can model many different objectives. This is in general not the case for the MILP makespan model which lacks explicit task start and end times. In what follows, we leverage this fact to express the CP model for minimizing TWC as additions to the CP model for makespan minimization.

The key difference that allows modelling TWC is the addition of encapsulating interval variables. Whereas the non-rework and rework interval variables of the makespan model have one interval variable for each combination of job and machine, each job gains just a single additional encapsulating interval variable and the encapsulating interval variable is not optional. This is similar to how the start and end time variables of jobs are not associated with machines in the MILP model for TWC. The start and end of a job's encapsulating interval variable is set equal to the start time of the non-rework interval variable and the end time of the rework interval variable corresponding to whichever machine the job is assigned respectively.

The weight parameter is used again:

w_j: weight of job j

The following decision variables are added:

TWC	Total weighted completion time objective variable
st_j^{enc}	Start time of encapsulating interval variable for job j
et_j^{enc}	End time of encapsulating interval variable for job j

dur_j^{enc}	Duration of encapsulating interval variable for job j
$Intrv_j^{enc}$	Encapsulating interval variable for job j

The objective function becomes:

$$\min TWC \tag{3.13}$$

s.t.[1]:

$$\sum_{j=1}^{N} w_j \cdot et_j^{enc} = TWC \tag{3.14}$$

$$x_{ij} \implies (st_{ij} = st_j^{enc}) \qquad \forall i, j \tag{3.15}$$

$$x_{ij} \implies (et\_rw_{ij} = et_j^{enc}) \qquad \forall i, j \tag{3.16}$$

$$st_j^{enc} + dur_j^{enc} = et_j^{enc} \qquad \forall i, j \tag{3.17}$$

$$TWC \geq 0 \tag{3.18}$$

$$st_j^{enc}, et_j^{enc}, dur_j^{enc} \geq 0 \qquad \forall j \tag{3.19}$$

Constraint (3.14) defines the weighted completion time objective variable. Constraints (3.15) and (3.16) ensure that the start and end of a job's encapsulating interval variable is set equal to the start time of the non-rework interval variable and the end time of the rework interval variable corresponding to whichever machine the job is assigned respectively. Constraints (3.17) are defined implicitly by the creation of the encapsulating interval variables and define the relation between start time, duration, and end time of the encapsulating interval variables. Constraints (3.18) and (3.19) are non-negativity constraints.

The second CP model is a direct translation into CP of the proposed MILP model for minimum TWC. To do this we use the same scale-factor Φ to ensure integer decision variable values, but otherwise the model is the same.

3.6 CP Experiments

To test the effectiveness of the CP formulation of the minimum TWC problem we construct and solve equivalent instances of the same Graham's Experiment benchmark

[1] $\wedge$, $\vee$, $\neg$, $\implies$, $\Leftrightarrow$ are logical AND, OR, NOT, IMPLICATION, and EQUALITY respectively.

as the MILP experiments. The CP models were solved using the CP-SAT solver of Google ORTools version 9.0.9163 [65] on the same laptop as the MILP experiments. The solver parameter *num_search_workers* was set to 8 which causes a portfolio of different solving approaches to be executed across 8 different cores in parallel. Both

Table 3.4: Graham's Experiment Total Weighted Completion Time - CP

		CP_{Trad}				CP_{Verb}			
m	N	LB	UB	Time	$Time_{UB}$	LB	UB	Time	$Time_{UB}$
2	5	75	75	0.02	0.01	75	75	0.02	0.01
3	7	167	167	1.14	0.15	167	167	0.96	0.07
4	9	332	332	819.34	0.5	332	332	96.89	2.26
5	11	424	533	1200	49.99	424	533	1200	5.59
6	13	502	817	1200	22	625	817	1200	15.79

*Values were recorded after 1200 seconds of run time

the traditional CP scheduling model and the verbatim translation of the MILP model to CP showed their strength on the minimum TWC instances proving optimality on the two, three, and four machine instances. This is largely due to the core-based MAX-SAT solver which is one of the algorithms in the portfolio of solvers applied to the problem. The core-based MAX-SAT solver is based upon ideas of minimum unsatisfiable subset/core (MUS/MUC) extraction [5][10] and SAT-based MaxSAT solving [4][18][23][17], and is especially well suited for problems with a sum objective involving variables with a similar structure in the constraints which happens to be the case for this problem.

3.7 Comparison of MILP, CP, & Heuristic

Having run identical experiments for the MILP model, CP model, and WSPT heuristic, we are in a position to compare the strengths and weaknesses of each approach for this particular scheduling problem.

From Table 3.5, we can see that the MILP and CP models found the same upper bound for the small instances of the proposed modified Graham's Experiment within a relatively short amount of time. However, unlike the makespan problem where the MILP formulation was the clear winner for proving optimality, both the traditional

CP scheduling model and the verbatim CP scheduling model were better at proving optimality for the TWC problem. Whereas the MILP model could only prove optimality for the 2 and 3 machine problems, the CP models were able to prove optimality up to the 4 machine problem as well as finding a superior lower bound than the MILP on the 5 and 6 machine problems. The results also suggest that the verbatim translation of the MILP to CP may perform better than the traditional CP model given that it found the best lower bound for the 6 machine problem and had the shortest runtimes for both proving optimality and time until finding the optimal (or best known) solution on most of the instances. The WSPT heuristic also performed extremely well finding solutions within 1.5% of the best known solution within 1 second.

For larger instances, we compared the best exact solver (verbatim CP) to the WSPT heuristic. The results from Table 3.6 show that the CP model was able to find a better feasible solution than the heuristic within the twenty minute time limit for the 10 and 15 machine problems, but the 20 machine problem proved too difficult and CP was bested by the heuristic. Based upon these results, we would recommend using the verbatim CP model on instances of size up to 15 machines and either heuristic methods, or exact solvers with heuristic warm-starts for larger problems.

Table 3.5: Graham's Experiment Total Weighted Completion Time - Small Instances

		MILP				CP_{Trad}				CP_{Verb}				WSPT		
m	N	LB	UB	Time	$Time_{UB}$	LB	UB	Time	$Time_{UB}$	LB	UB	Time	$Time_{UB}$	UB	$\frac{UB}{BEST}$	$Time_{UB}$
2	5	75	75	0.09	< 0.09	75	75	0.02	0.01	75	75	0.02	0.01	75	1	< 1
3	7	167	167	3.37	< 2	167	167	1.14	0.15	167	167	0.96	0.07	169	1.012	< 1
4	9	284.09	332	1200	1	332	332	819.34	0.5	332	332	96.89	2.26	333	1.003	< 1
5	11	348.63	533	1200	6	424	533	1200	49.99	424	533	1200	5.59	541	1.015	< 1
6	13	488	817	1200	22	502	817	1200	22	625	817	1200	15.79	827	1.012	< 1

*Values were recorded after 1200 seconds of run time

Table 3.6: Graham's Experiment Total Weighted Completion Time - Large Instances

		CP_{Verb}				WSPT		
m	N	LB	UB	Time	$Time_{UB}$	UB	$\frac{UB}{BEST}$	$Time_{UB}$
10	21	1715	2661	1200	415.67	2693	1.012	< 1
15	31	4497	7058	1200	921.21	7076	1.003	< 1
20	41	8857	14592	1200	1076.27	14440	0.990	< 1

*Values were recorded after 1200 seconds of run time

Chapter 4

Conclusion & Directions For Further Research

An abundance of research exists studying parallel machine scheduling, but only a portion of this research has focused on the case with non-availability periods and job resumability factors. Research considering this case has mostly focused on limited cases such as a small, fixed number of machines or fixed number of non-availability periods.

In this research, we improve the runtime of the MILP model of Beaton et al. [7] that allows an arbitrary number of machines, non-availability periods, and arbitrary resumability factor for the makespan minimization case. We did this by considering different model additions including: lexicographic ordering constraints to break symmetry in the ordering of identical machines, a lower bound on the makespan consisting of averaging total job processing time across machines while accounting for non-availability periods, and warm-starting the model with the best solution from the four heuristics of Beaton et al. [7]. Four different configurations of model additions were able to prove optimality of the 20 machine, semi-resumable case of the modified Graham's Experiment for the first time. Additionally, even when the MILP models were not able to prove optimality, they were frequently able to find the optimal solution within a relatively short amount of time.

We also proposed two different constraint programming (CP) models for the makespan case; the first a traditional CP scheduling model involving explicit job start time, end time, duration, and interval variables that takes advantage of the powerful filtering and propagation algorithms of the global disjunctive scheduling constraint, and the second a verbatim translation of the MILP model into an equivalent CP model. While the CP models were less effective at proving optimality, they performed similarly to the MILP model for finding the optimal solution, especially

for the semi-resumable and non-resumable cases.

We proposed a new MILP model for the Total Weighted Completion Time (TWC) case allowing an arbitrary number of machines and non-availability periods for the fully-resumable case as well as a new modified Graham's Experiment benchmark for testing solution approaches. The correctness of the model was verified by comparison with an example from Mellouli et al [59]. Similar to the makespan case, we also introduced a traditional CP scheduling model and a verbatim translation of the MILP model into an equivalent CP model. For the TWC objective, we found that both CP models outperformed the MILP model by proving optimality of a larger instance as well as finding improved lower bounds on problems where optimality was not proven. Additionally, we confirmed that a weighted shortest processing time (WSPT) heuristic performs extremely well for the TWC objective finding solutions within 1.5% of the best solution from an exact solver in less than one second. Furthermore, the exact solvers began to struggle with larger problem instances and the WSPT heuristic outperformed the best exact solver at the 20 machine problem with a 20 minute time limit.

There are many possible directions for future research on the topic of parallel machine scheduling with non-availability periods and arbitrary resumability factor. In future research we intend to extend the exact TWC model to handle arbitrary resumability. Another interesting line of research would be to generalize the model to allow scheduling both jobs and non-availability periods. We could also consider different performance metrics such as weighted earliness and tardiness or minimum lateness, the effect of machine eligibility constraints, and the effect of stochastic values such as processing times and non-availability period start times and durations.

Bibliography

[1] Disjunctive global constraint definition. `https://sofdem.github.io/gccat/gccat/Cdisjunctive.html`. Accessed: 2021-08-05.

[2] lex_chain_lesseq_bool.mzn linearized global constraint definition. `https://github.com/MiniZinc/libminizinc/blob/master/share/minizinc/linear/fzn_lex_chain_lesseq_bool.mzn`. Accessed: 2021-07-28.

[3] Igal Adiri, John Bruno, Esther Frostig, and AHG Rinnooy Kan. Single machine flow-time scheduling with a single breakdown. *Acta Informatica*, 26(7):679–696, 1989.

[4] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. Sat-based maxsat algorithms. *Artificial Intelligence*, 196:77–105, 2013.

[5] Gilles Audemard, Jean-Marie Lagniez, and Laurent Simon. Improving glucose for incremental sat solving with assumptions: Application to mus extraction. In *International conference on theory and applications of satisfiability testing*, pages 309–317. Springer, 2013.

[6] Mikhail Batsyn, Boris Goldengorin, Panos M Pardalos, and Pavel Sukhov. Online heuristic for the preemptive single machine scheduling problem of minimizing the total weighted completion time. *Optimization Methods and Software*, 29(5):955–963, 2014.

[7] Clifford Beaton, Claver Diallo, and Eldon Gunn. Makespan minimization for parallel machine scheduling of semi-resumable and non-resumable jobs with multiple availability constraints. *INFOR: Information Systems and Operational Research*, 54(4):305–316, 2016.

[8] Nicolas Beldiceanu, Mats Carlsson, Sophie Demassey, and Thierry Petit. Global constraint catalogue: Past, present and future. *Constraints*, 12(1):21–62, 2007.

[9] Nicolas Beldiceanu, Mats Carlsson, and Jean-Xavier Rampon. Global constraint catalog. 2005.

[10] Jeremias Berg and Matti Järvisalo. Weight-aware core extraction in sat-based maxsat solving. In *International Conference on Principles and Practice of Constraint Programming*, pages 652–670. Springer, 2017.

[11] James Bruno, Edward G Coffman Jr, and Ravi Sethi. Scheduling independent tasks to reduce mean finishing time. *Communications of the ACM*, 17(7):382–387, 1974.

[12] Jacques Carlier. The one-machine sequencing problem. *European Journal of Operational Research*, 11(1):42–47, 1982.

[13] Mats Carlsson and Nicolas Beldiceanu. Arc-consistency for a chain of lexicographic ordering constraints, 2002.

[14] Jen-Shiang Chen. Scheduling of nonresumable jobs and flexible maintenance activities on a single machine to minimize makespan. *European Journal of Operational Research*, 190(1):90–102, 2008.

[15] Ya-Yong Chen, Pei-Yu Huang, Cheng-Jun Huang, Shen-Quan Huang, and Fuh-Der Chou. Makespan minimization for scheduling on two identical parallel machiens with flexible maintenance and nonresumable jobs. *Journal of Industrial and Production Engineering*, 38(4):271–284, 2021.

[16] Yarong Chen, Chenjun Huang, Fuh-Der Chou, and Shenquan Huang. Single-machine scheduling problem with flexible maintenance and non-resumable jobs to minimise makespan. *IET Collaborative Intelligent Manufacturing*, 2(4):174–181, 2020.

[17] Jessica Davies and Fahiem Bacchus. Solving maxsat by solving a sequence of simpler sat instances. In *International conference on principles and practice of constraint programming*, pages 225–239. Springer, 2011.

[18] Toby Davies, Graeme Gange, and Peter Stuckey. Automatic logic-based benders decomposition with minizinc. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[19] Willard L Eastman, Shimon Even, and I Martin Isaacs. Bounds for the optimal scheduling of n jobs on m processors. *Management science*, 11(2):268–279, 1964.

[20] Thibaut Feydy and Peter J Stuckey. Lazy clause generation reengineered. In *International Conference on Principles and Practice of Constraint Programming*, pages 352–366. Springer, 2009.

[21] Bin Fu, Yumei Huo, and Hairong Zhao. Exponential inapproximability and fptas for scheduling with availability constraints. *Theoretical Computer Science*, 410(27-29):2663–2674, 2009.

[22] Bin Fu, Yumei Huo, and Hairong Zhao. Approximation schemes for parallel machine scheduling with availability constraints. *Discrete Applied Mathematics*, 159(15):1555–1565, 2011.

[23] Zhaohui Fu and Sharad Malik. On solving the partial max-sat problem. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 252–265. Springer, 2006.

[24] Ian P Gent, Karen E Petrie, and Jean-François Puget. Symmetry in constraint programming. *Foundations of Artificial Intelligence*, 2:329–376, 2006.

[25] Anis Gharbi and Mohamed Haouari. Optimal parallel machines scheduling with availability constraints. *Discrete Applied Mathematics*, 148(1):63–87, 2005.

[26] Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics*, 17(2):416–429, 1969.

[27] Ronald Lewis Graham, Eugene Leighton Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*, volume 5, pages 287–326. Elsevier, 1979.

[28] Gregory H Graves and Chung-Yee Lee. Scheduling maintenance and semiresumable jobs on a single machine. *Naval Research Logistics (NRL)*, 46(7):845–863, 1999.

[29] Lin Guohui, He Yong, Yao Yujun, and Lu Haiyan. Exact bounds of the modified lpt algorithms applying to parallel machines scheduling with nonsimultaneous machine available times. *Applied Mathematics-A Journal of Chinese Universities*, 12(1):109–116, 1997.

[30] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2021.

[31] Navid Hashemian. Makespan minimization for parallel machines scheduling with availability constraints. 2010.

[32] Navid Hashemian, Claver Diallo, and Béla Vizvári. Makespan minimization for parallel machines scheduling with multiple availability constraints. *Annals of Operations Research*, 213(1):173–186, 2014.

[33] Raf Jans. Solving lot-sizing problems on parallel identical machines using symmetry-breaking constraints. *INFORMS Journal on Computing*, 21(1):123–136, 2009.

[34] Jihene Kaabi and Youssef Harrath. A survey of parallel machine scheduling under availability constraints. *International Journal of Computer and Information Technology*, 3(2):238–245, 2014.

[35] Imed Kacem and Chengbin Chu. Efficient branch-and-bound algorithm for minimizing the weighted sum of completion times on a single machine with one availability constraint. *International Journal of Production Economics*, 112(1):138–150, 2008.

[36] Imed Kacem, Chengbin Chu, and Ahmed Souissi. Single-machine scheduling with an availability constraint to minimize the weighted sum of the completion times. *Computers & operations research*, 35(3):827–844, 2008.

[37] Imed Kacem and A Ridha Mahjoub. Fully polynomial time approximation scheme for the weighted flow-time minimization on a single machine with a fixed non-availability interval. *Computers & Industrial Engineering*, 56(4):1708–1712, 2009.

[38] Tsuyoshi Kawaguchi and Seiki Kyan. Worst case bound of an lrf schedule for the mean weighted flow-time problem. *SIAM Journal on Computing*, 15(4):1119–1129, 1986.

[39] Daniel Kowalczyk and Roel Leus. An exact algorithm for parallel machine scheduling with conflicts. *Journal of Scheduling*, 20(4):355–372, 2017.

[40] Arthur Kramer, Mauro Dell'Amico, and Manuel Iori. Enhanced arc-flow formulations to minimize weighted completion time on identical parallel machines. *European Journal of Operational Research*, 275(1):67–79, 2019.

[41] Hanane Krim, Rachid Benmansour, David Duvivier, Daoud Aït-Kadi, and Said Hanafi. Heuristics for the single machine weighted sum of completion times scheduling problem with periodic maintenance. *Computational Optimization and Applications*, 75(1):291–320, 2020.

[42] Wen-Yang Ku and J Christopher Beck. Mixed integer programming models for job shop scheduling: A computational analysis. *Computers & Operations Research*, 73:165–173, 2016.

[43] Philippe Laborie and Jerome Rogerie. Reasoning with conditional time-intervals. In *FLAIRS conference*, pages 555–560, 2008.

[44] Philippe Laborie, Jerome Rogerie, Paul Shaw, and Petr Vilím. Reasoning with conditional time-intervals. part ii: An algebraical model for resources. In *Twenty-Second International FLAIRS Conference*, 2009.

[45] Philippe Laborie, Jérôme Rogerie, Paul Shaw, Petr Vilím, and Ferenc Katai. Interval-based language for modeling scheduling problems: An extension to constraint programming. In *Algebraic Modeling Systems*, pages 111–143. Springer, 2012.

[46] Chung-Yee Lee. Parallel machines scheduling with nonsimultaneous machine available time. *Discrete Applied Mathematics*, 30(1):53–61, 1991.

[47] Chung-Yee Lee. Machine scheduling with an availability constraint. *Journal of global optimization*, 9(3-4):395–416, 1996.

[48] Chung-Yee Lee. Two-machine flowshop scheduling with availability constraints. *European Journal of Operational Research*, 114(2):420–429, 1999.

[49] Chung-Yee Lee, Yong He, and Guochun Tang. A note on "parallel machine scheduling with non-simultaneous machine available time". *Discrete Applied Mathematics*, 100(1-2):133–135, 2000.

[50] Chung-Yee Lee, Lei Lei, and Michael Pinedo. Current trends in deterministic scheduling. *Annals of operations Research*, 70:1–41, 1997.

[51] Chung-Yee Lee and Surya Danusaputro Liman. Single machine flow-time scheduling with scheduled maintenance. *Acta Informatica*, 29(4):375–382, 1992.

[52] Chung-Yee Lee and Surya Danusaputro Liman. Capacitated two-parallel machines scheduling to minimize sum of job completion times. *Discrete Applied Mathematics*, 41(3):211–222, 1993.

[53] Jan Karel Lenstra, AHG Rinnooy Kan, and Peter Brucker. Complexity of machine scheduling problems. In *Annals of discrete mathematics*, volume 1, pages 343–362. Elsevier, 1977.

[54] Kai Li and Shan-lin Yang. Non-identical parallel-machine scheduling research with minimizing total weighted completion times: Models, relaxations and algorithms. *Applied mathematical modelling*, 33(4):2145–2158, 2009.

[55] Ching-Jong Liao, Der-Lin Shyur, and Chien-Hung Lin. Makespan minimization for two parallel machines with an availability constraint. *European Journal of Operational Research*, 160(2):445–456, 2005.

[56] Lu-Wen Liao and Gwo-Ji Sheen. Parallel machine scheduling with machine availability and eligibility constraints. *European Journal of Operational Research*, 184(2):458–467, 2008.

[57] Ying Ma, Chengbin Chu, and Chunrong Zuo. A survey of scheduling with deterministic machine availability constraints. *Computers & Industrial Engineering*, 58(2):199–211, 2010.

[58] Racem Mellouli, Imed Kacem, Chérif Sadfi, and Chengbin Chu. Lagrangian relaxation and column generation-based lower bounds for the pm, hj1||sum wici scheduling problem. *Applied Mathematics and Computation*, 219(22):10783–10805, 2013.

[59] Racem Mellouli, Cherif Sadfi, Chengbin Chu, and Imed Kacem. Identical parallel-machine scheduling under availability constraints to minimize the sum of completion times. *European Journal of Operational Research*, 197(3):1150–1165, 2009.

[60] Lars Mönch and Liji Shen. Parallel machine scheduling with the total weighted delivery time performance measure in distributed manufacturing. *Computers & Operations Research*, 127:105126, 2021.

[61] Rabia Nessah and Chengbin Chu. Infinite split scheduling: a new lower bound of total weighted completion time on parallel machines with job release dates and unavailability periods. *Annals of Operations Research*, 181(1):359–375, 2010.

[62] Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck, and Guido Tack. Minizinc: Towards a standard cp modelling language. In *International Conference on Principles and Practice of Constraint Programming*, pages 529–543. Springer, 2007.

[63] Olga Ohrimenko, Peter J Stuckey, and Michael Codish. Propagation via lazy clause generation. *Constraints*, 14(3):357–391, 2009.

[64] Jason Chao-Hsien Pan and Chi-Shiang Su. Two parallel machines problem with job delivery coordination and availability constraint. *Annals of Operations Research*, 235(1):653–664, 2015.

[65] Laurent Perron and Vincent Furnon. Or-tools.

[66] Cherif Sadfi, Imed Kacem, and Wei Liu. Lower bounds for total weighted completion scheduling problem with availability constraints. In *2009 International Conference on Computers & Industrial Engineering*, pages 159–163. IEEE, 2009.

[67] Eric Sanlaville and Günter Schmidt. Machine scheduling with availability constraints. *Acta Informatica*, 35(9):795–811, 1998.

[68] Günter Schmidt. Scheduling with limited machine availability. *European Journal of Operational Research*, 121(1):1–15, 2000.

[69] Wayne E Smith et al. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956.

[70] Petr Vilím. Global constraints in scheduling. 2007.

[71] Petr Vilím, Roman Barták, and Ondřej Čepek. Unary resource constraint with optional activities. In *International Conference on Principles and Practice of Constraint Programming*, pages 62–76. Springer, 2004.

[72] Guoqing Wang, Hongyi Sun, and Chengbin Chu. Preemptive scheduling with availability constraints to minimize total weighted completion times. *Annals of Operations Research*, 133(1-4):183–192, 2005.

[73] Dehua Xu and Dar-Li Yang. Makespan minimization for two parallel machines scheduling with a periodic availability constraint: mathematical programming model, average-case analysis, and anomalies. *Applied Mathematical Modelling*, 37(14-15):7561–7567, 2013.

[74] Zhijun Xu, Aihua Liu, and Qi Wang. Mixed 0–1 programming model for three parallel machines scheduling problem with machine-dependent unavailable constraints. In *2016 13th International Conference on Service Systems and Service Management (ICSSSM)*, pages 1–4. IEEE, 2016.

[75] Jaewook Yoo and Ik Sun Lee. Parallel machine scheduling with maintenance activities. *Computers & Industrial Engineering*, 101:361–371, 2016.

[76] Chuanli Zhao, Min Ji, and Hengyong Tang. Parallel-machine scheduling with an availability constraint. *Computers & Industrial Engineering*, 61(3):778–781, 2011.

Appendix A. GLPK/Gusek MILP Code for the makespan minimization problem

```
################  Parameters  ###################################
#number of machines
param m;
set M := 1..m;

#number of jobs
param n;
set N := 1..n;

#number of unavailability periods on machine i
param r{i in M};
set R{i in M} := 1..r[i];

#processing times
param p{j in N};

#resumability factor
param alpha, default 0;

#Starting time of unavailability periods
param s{i in M, q in R[i]};

 #Ending time of unavailability periods
param e{i in M, q in R[i]};

#Duration of unavailability periods
param D{i in M, q in R[i]} := e[i,q] - s[i,q];
```

```
#big M
param Z, default 10000;

############  Decision Variables  ################################
# 1 if job j is assigned to machine i, 0 otherwise
var x{i in M, j in N}, binary;

# 1 if jobs on machine i are completed before siq, 0 otherwise
var y{i in M, q in R[i]}, binary;

# amount of time after the last job has been completed before
# unavailability q
var w{i in M, q in R[i]}, >= 0;

# 1 if job j is completed before unavailability q on machine i,
# 0 otherwise
var b{i in M, j in N, q in R[i]}, binary;

#makespan
var Cmax, >= 0;

#################  Objective  ###################################

minimize f: Cmax;

#################  Constraints  #################################

s.t. c1{i in M, q in R[i]}:
```

```
sum{j in N}p[j]*x[i,j] + sum{k in 1..q-1}(D[i,k]
+ alpha*w[i,k]) <= s[i,q] + Z*(1-y[i,q]);
s.t. c2{i in M}:
sum{j in N}p[j]*x[i,j] + sum{q in R[i]}(D[i,q]*(1-y[i,q])
    + alpha*w[i,q]) <= Cmax;
s.t. c3{j in N}: sum{i in M}x[i,j] = 1;
s.t. c4{i in M, q in R[i]}:
w[i,q]>=s[i,q]*(1-y[i,q]) - sum{j in N}p[j]*b[i,j,q]
    - sum{k in 1..q-1}(D[i,k] + alpha*w[i,k]);
s.t. c5{i in M, q in R[i]}:
w[i,q] <= s[i,q] - sum{j in N}p[j]*b[i,j,q]
    - sum{k in 1..q-1}(D[i,k] + alpha*w[i,k]);
s.t. c6{i in M, j in N, q in R[i]}: b[i,j,q] <= x[i,j];
s.t. c7{i in M, j in N, q in 1..r[i]-1}: b[i,j,q+1] >= b[i,j,q];
s.t. c8{i in M, q in 1..r[i]-1}:
sum{j in N}(p[j]*b[i,j,q+1]-p[j]*b[i,j,q])
    <= s[i,q+1] - e[i,q] - alpha*w[i,q] + w[i,q];

solve;

display f, x, y, w, b, D;
data;

param m := 2;
param n := 10;
param alpha := 0.5;
param r :=
1 2
2 2;

param p :=
1 2
```

```
2 17
3 4
4 7
5 10
6 12
7 9
8 10
9 17
10 14
;


param s :=
1 1 1
1 2 4
2 1 3
2 2 7
;

param e:=
1 1 2
1 2 5
2 1 5
2 2 8
;

end;
```

Appendix B. GLPK/Gusek MILP Code for the weighted completion times minimization problem

```
# MILP Parallel Machine Scheduling Weighted Completion Times
# With Multiple Non-availability Periods

param m; #number of machines
set M := 1..m;

param n; #number of jobs
set N :=1..n;

#number of unavailability periods on machine i
param r{i in M};

set R{i in M} := 1..r[i];

param p{j in N}; #processing times
param w{j in N}; #weights

#Starting time of unavailability periods
param s{i in M, q in R[i]};

#Ending time of unavailabilty periods
param e{i in M, q in R[i]};

#Duration of unavailability periods
param D{i in M, q in R[i]} := e[i,q] - s[i,q];

param Z, default 500; #big M
```

```
###  Decision Variables  ###

# 1 if job j is assigned to machine i, 0 otherwise
var x{j in N, i in M}, binary;

# 1 if job j is scheduled before job k on machine,
# 0 otherwise
var z{j in N, k in N}, binary;

# 1 if job j is completed after unavailability q on machine i,
# 0 otherwise
var b{i in M, j in N, q in R[i]}, binary;

# start time of job j
var S{j in N}, >= 0;

#completion time of job j
var C{j in N}, >= 0;

###  Objective  ###

minimize f: sum{j in 1..n} w[j]*C[j];


###  Constraints  ###
s.t. ct1{j in 1..n}: sum{i in M} x[j,i] = 1;
s.t. ct2{j in 1..n, k in 1..n, i in M: j<>k}:
S[k] + p[k] -Z*(2-x[j,i]-x[k,i]+z[j,k]) <= S[j];
```

```
s.t. ct3{j in 1..n, k in 1..n, i in M: j<>k}:
S[j] + p[j] -Z*(3-x[j,i]-x[k,i]-z[j,k]) <= S[k];
s.t. ct4{j in 1..n, i in M}:
S[j] + p[j]*x[j,i] + sum{q in 1..r[i]} D[i,q]*b[i,j,q] <= C[j];
s.t. ct5{j in 1..n, i in M, q in 1..r[i]}:
C[j] <= s[i,q] + Z*b[i,j,q];
s.t. ct6{i in M, j in N, q in 1..r[i]-1}:
b[i,j,q+1] <= b[i,j,q];
solve;

display f, C, S, x, z, b, D;
data;

param m := 3;
param n := 10;
param r :=
1 1
2 1
3 1
;

param p :=
1 1
2 1
3 1
4 2
5 3
6 3
7 4
8 4
9 5
10 5
```

```
;

param w :=
1 1
2 1
3 1
4 1
5 1
6 1
7 1
8 1
9 1
10 1
;

param s :=
1 1 5
2 1 5
3 1 7
;

param e :=
1 1 7
2 1 8
3 1 8
;


end;
```