UNIVERSITY OF TECHNOLOGY SYDNEY
Faculty of Science

The Coupled Task Scheduling Problem: Models and Solution Methods

by

Mostafa Khatami

A Thesis Submitted
in Partial Fulfillment of the
Requirements for the Degree

Doctor of Philosophy

Sydney, Australia

2022

Certificate of Authorship/Originality

I, Mostafa Khatami, certify that the work in this thesis has not been previously submitted for a degree nor has it been submitted as a part of the requirements for other degree except as fully acknowledged within the text.

I also certify that this thesis has been written by me. Any help that I have received in my research and in the preparation of the thesis itself has been fully acknowledged. In addition, I certify that all information sources and literature used are quoted in the thesis.

This research is supported by an Australian Government Research Training Program.

Production Note:
Signature removed
prior to publication.

ABSTRACT

The Coupled Task Scheduling Problem:

Models and Solution Methods

by

Mostafa Khatami

The coupled task scheduling problem (CTSP) is studied in this thesis. The problem consists of scheduling a set of jobs on one or a set of machines, where each job consists of at least two tasks. The main characteristic of the problem is a fixed time-lag between the process of each two consecutive tasks of the same job, where its duration is fixed, i.e., the succeeding task cannot be started earlier or later than the time-lag is passed. The fixed time-lags were introduced to model radar tracking systems, and later extended to formulate problems in chemistry manufacturing systems and robotic cells. The motivation for studying the CTSP in this thesis is to model certain problems in healthcare scheduling with the same characteristics. One example is the scheduling of patients in a chemotherapy clinic, where each patient must undergo a number of consecutive treatments with time-lags in between. Meeting the fixed delays between the treatments of a patient is an important factor in gaining the best outcomes for them. To study the CTSP, a literature review is first conducted, followed by studying the problem in different scheduling environments, including the single-machine, parallel-machine, open-shop and flow-shop settings, where we propose several new complexity results and solution algorithms for different variants of the problem.

Regarding the single-machine coupled task problem, a new mathematical formulation and two matheuristic algorithms are proposed for the classical problem, as well as a dynamic programming algorithm for a variant of the problem with time-dependent processing times.

With regard to the parallel-machine environment, we first explore the complexity of the problem and propose NP-hardness proofs for certain cases, followed by approximation bounds for the two-machine problem. The latter result is then extended to the open-shop scheduling environment.

The problem in the flow-shop environment is then extensively investigated under the permutation setting, and also under the case of ordered processing times. A set of publicly available hard data set and state-of-the-art algorithms are proposed for the ordered flow-shops. Then, flow-shop problem with coupled tasks is studied and polynomial-time algorithms are proposed for various settings of the problem, including the ordered processing times.

Dissertation directed by Dr Amir Salehipour
School of Mathematical and Physical Sciences

Dedication

I would have never accomplished this degree without your wholehearted love, *Mahboubeh*. To you, my beloved wife, and our little *Ali*.

Acknowledgements

I would like to thank my principal supervisor Dr Amir Salehipour for his great and continued support during my degree. He has always been available to help me, and his strong support has been essential for my achievements.

I also thank my co-supervisors Professor Daniel Oron (The University of Sydney Business School) and Dr Hanyu Gu (UTS), and my former co-supervisor Dr Feng-Jang Hwang (UTS), for their helps and supports throughout my degree.

I would like to thank Professor Lance Lesley (UTS) and Professor Murray Elder (UTS), for their kind smiles and nice chats during my time at UTS. I also thank Mrs Julia Memar (UTS) for her great supports.

I thank UTS Graduate Research School for providing me with PhD scholarships, and UTS Faculty of Science for assisting me with conference grants.

Finally, I thank my parents for their support, and my wife for her all time love.

Mostafa Khatami

Sydney, Australia, 2022.

List of Publications

Journal Papers

J-1. Khatami, M., Salehipour, A., and Hwang, F. J. (2019). "Makespan minimization for the m-machine ordered flow shop scheduling problem". *Computers and Operations Research* 111, 400–414.

J-2. Khatami, M., Salehipour, A., and Cheng, T. C. E. (2020). "Coupled task scheduling with exact delays: Literature review and models". *European Journal of Operational Research* 282(1), 19 –39.

J-3. Khatami, M. and Salehipour, A. (2021a). "A binary search algorithm for the general coupled task scheduling problem". *4OR* 19(4), 593–611.

J-4. Khatami, M. and Salehipour, A. (2021b). "Coupled task scheduling with time-dependent processing times". *Journal of Scheduling* 24, 223–236.

Conference Papers

C-1. Khatami, M., Salehipour, A., and Hwang, F. J. (2018). "Single-machine coupled task scheduling with time-dependent processing times". *ASOR 2018*. Melbourne, Australia.

C-2. Khatami, M. and Salehipour, A. (2019). "A simple heuristic for the coupled task scheduling problem". *MODSIM 2019*. Canberra, Australia.

C-3. Khatami, M. and Salehipour, A. (2020). "A relax-and-solve algorithm for the ordered flow-shop scheduling problem". *IEEE IEEM 2020*. Singapore.

Preprints under review

P-1. Khatami, M. and Salehipour, A. (2021c). "The coupled task scheduling problem: An improved mathematical program and a new solution algorithm". *Submitted to International Transactions in Operational Research*.

P-2. Khatami, M., Salehipour, A., and Cheng, T. C. E. (2021b). "Flow-shop scheduling with exact delays to minimize makespan". *Submitted to Computers & Industrial Engineering*.

P-3. Khatami, M., Oron, D., and Salehipour, A. (2021a). "Scheduling coupled tasks on parallel identical machines". *Submitted to Annals of Operations Research*.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, definitions, scope and classification of the problem will be discussed (Section 1.1), followed by the research goals (Section 1.2). The chapter ends with presenting the organisation of the remainder of the thesis in Section 1.3.

1.1 Definitions, scope and classification

Scheduling is a resource allocation problem, and deals with the problem of allocating resources to some activities over specified time periods, where the goal is optimise one or more objective functions (Pinedo, 2012). In this thesis, we denote the resources as machines, and the activities as jobs. We formally formulate the single-machine coupled task scheduling problem (CTSP) as follows: There is a set $N = \{1, \ldots, n\}$ of jobs, indexed by j, where two tasks are associated with each job $j \in N$. The first (initial) task and its processing time are denoted by a_j while the second (completion) task and its processing time are denoted by b_j. As a result, there is a set $H = \{1, \ldots, 2n\}$ of tasks, indexed by h. Both tasks have known durations and the second task of a job must be started with a delay after the completion of the first task. We denote the delay duration between the tasks of job j as L_j, as illustrated in Figure 1.1. In the context of CTSP, it is typical to present the parameters for job j via a triple (a_j, L_j, b_j).

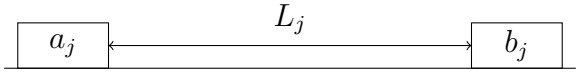

Figure 1.1 : A coupled-task job.

In the shop setting, the set N of jobs need to be processed on a set $M = \{1, \ldots, m\}$ of different machines, indexed by k. The coupled task assumption in the shop setting represents the situation where exact delays are considered between each two consecutive tasks of the same job. The processing time of job j's task on machine k is denoted by $p_{kj}, \forall k \in M, \forall j \in N$, and the delay duration of job j after being processed on machine k is represented by L_{kj}. For simplicity, in the two-machine setting, we replace p_{1j} and p_{2j} by a_j and b_j, respectively. Likewise, we replace L_{kj} by L_j. In both single-machine and shop settings we assume all processing times and delay durations take positive integers, unless otherwise specified.

A scheduling problem can be denoted with a three-field notation $\alpha|\beta|\gamma$ (Graham et al., 1979), where α field represents the scheduling environment, the field β presents details of the characteristics and constraints of the problem, and the field γ shows the objective function(s). The scheduling environments discussed in this thesis are the single-machine (1) environment, where there is a single-machine and all jobs are executed on that machine, the parallel identical machines (Pm) environment, where there are m identical machines and a job can be processed on any one of the machines, the open-shop (Om) environment, where there are m machines and each job needs to go through all machines, with no pre-specified order, the flow-shop (Fm) environment that restricts the open-shop environment in the sense that all jobs follow the same processing route, i.e., from machine 1 to machine m, and the job-shop (Jm) environment where every job has its own pre-specified route.

With respect to β field, the problems discussed in this thesis deal with the coupled task characteristic, i.e., existence of fixed delays between consecutive tasks of a job. Specifically, each job consists of at least two tasks, where there is a fixed amount of delay between every two consecutive tasks. The fixed delay therefore represents a strict time-lag between the completion time of the preceding task and the starting time of the succeeding task. In the single-machine setting (a_j, L_j, b_j) represents the coupled task characteristic, which can be modified based on the values of a_j, L_j and b_j. On the other hand, in the shop setting, the coupled task characteristic is represented by L_j.

With respect to the field γ, let d_j, C_j, E_j and T_j denote the due date, completion time, earliness and tardiness of job j, respectively, where $E_j = \max\{d_j - C_j, 0\}$ and $T_j = \max\{C_j - d_j, 0\}$. The most commonly used objective function for the CTSP is the total length of the schedule, i.e., the makespan, which is denoted as $C_{\max}$. Also note that we represent a general regular objective function by f. An objective function is called regular if it is non-decreasing in the completion times of the jobs.

The definition of the coupled task setting is close to some other related characteristics, that we do not study in this thesis. In the following we briefly define those related problems to specify the position of this thesis.

There is research on scheduling problems with "flexible delays", in which a lower and/or an upper bound on the duration of the delay is/are given. The lower bound implies that the delay between two consecutive tasks of a job must not be smaller than the minimum delay (this is denoted by finish-start time-lags in the literature) and the upper bound means the delay cannot be greater than the given bound. We can regard the CTSP as a special case of scheduling with flexible delays, where the minimum and maximum delays for each job are equal (see Figure 1.2). We refer the interested reader to Dell'Amico (1996), Potts and Whitehead (2007), and Zhang and Van De Velde (2010) for reviews of research on scheduling problems with flexible delays.

The CTSP in the shop environment is related to the well-known "no-wait" condition

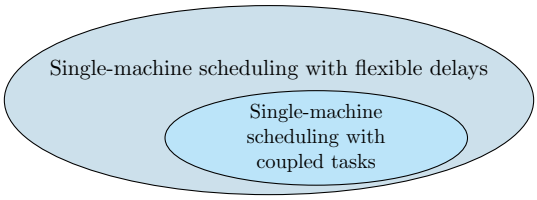

Figure 1.2 : Single-machine with flexible delays and single-machine coupled task scheduling.

in shop scheduling. The no-wait assumption implies that the jobs are processed on all the machines without any delay between them. Therefore, we can regard the no-wait shop scheduling problem as a special case of the CTSP, where the value of the delay between every pair of tasks is equal to zero (see Figure 1.3). Allahverdi (2016) discussed research on no-wait shop scheduling.

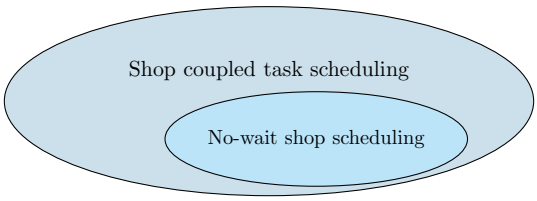

Figure 1.3 : Shop coupled task scheduling and no-wait shop scheduling.

1.2 Research aims

The main aims of this thesis include:

[**i**]. Conducting a review of the available results on the CTSP;

[**ii**]. Presenting new publicly available benchmark data sets for the CTSP;

[**iii**]. Exploring the computational complexity of different variants of the CTSP;

[**iv**]. Proposing new efficient and effective solution methodologies for the studied variants;

[**v**]. Investigating how the CTSP can be utilised to model real-world problems, mainly in the healthcare scheduling.

1.3 Thesis Organisation

The rest of this thesis is organised as follows:

- *Chapter 2*: This chapter presents a complete review of the available literature on the couple task scheduling problem, that is classified by the scheduling environment and different objective functions. The classical applications of the problem are then discussed, followed by the newly explored healthcare applications of the CTSP.

- *Chapter 3*: In this chapter, new benchmarks are proposed for different variants of the CTSP. The available mathematical models for the problem are then discussed, and some corrections/improvements are provided. The chapter ends with a thorough comparison of the mathematical models based on extensive computational experiments conducted on the proposed data sets.

- *Chapter 4*: This chapter is devoted to the single-machine CTSP under the objective of minimising the makespan. First, we discuss the single-machine problem under a special setting where processing times are time-dependent. Then, the general problem is studied, where a binary search matheuristic is proposed for the problem, that is based on an existing mathematical formulation of the problem. Then, a new improved formulation is proposed based on the existing model. The chapter continuous with proposing a second matheuristic for the problem, that benefits from the new proposed mathematical formulation. The quality of the proposed model is then discussed as a stand-alone method within a commercial solver, and also within the framework of the proposed matheuristic algorithms.

- *Chapter 5*: The CTSP on parallel identical machines with the makespan criterion in studied in this chapter. First, the complexity of the problem under different special cases is discussed, where most of the cases are shown to be strongly NP-hard. Then, an approximation bound is presented for the two-machine case of the problem. The latter result is then extended to the two-machine open-shop problem with coupled tasks.

- *Chapter 6*: The CTSP in the flow-shop environment is discussed in this chapter. Firstly, we explore an special setting in the flow-shop scheduling problem, denoted as ordered flow-shops. For that problem, we propose a new hard benchmark and state-of-the-art algorithms. Then, CTSP is studied and some polynomially solvable cases are presented, including the ordered flow-shop case.

- *Chapter 7*: A brief summary of the problems studied in this thesis and the main contributions are provided in this chapter. Some recommendations for future researches are also presented in this chapter.

Chapter 2

Literature review and applications

In this chapter, we review and discuss the literature of the coupled task scheduling problem (CTSP) in Section 2.1. The literature review will be presented with regard to the scheduling environment, and also the performance criteria discussed in each environment. We will then discuss the real-world applications of the CTSP in detail in Section 2.2. Parts of the discussions presented in this chapter is published in:

- Khatami, M., Salehipour, A., and Cheng, T. C. E. (2020). "Coupled task scheduling with exact delays: Literature review and models". *European Journal of Operational Research* 282(1), 19 –39.

2.1 Literature review

A comprehensive literature review of the CTSP on single-machine and shop scheduling problems is presented in the following. We present the papers available up to the year 2021, where we review 45 published papers. The first study on the problem appeared in 1980. A few papers on the problem were published between 1980 and 2006, and almost 76% of the related studies were published after 2006. This is an indication of the current research trend for the problem. Figure 2.1 shows the number of papers published over the years. Table 2.1 summaries the outlets that publish the available studies on the CTSP. According to the table, the majority of the papers were published in conference proceedings and *Journal of Scheduling*.

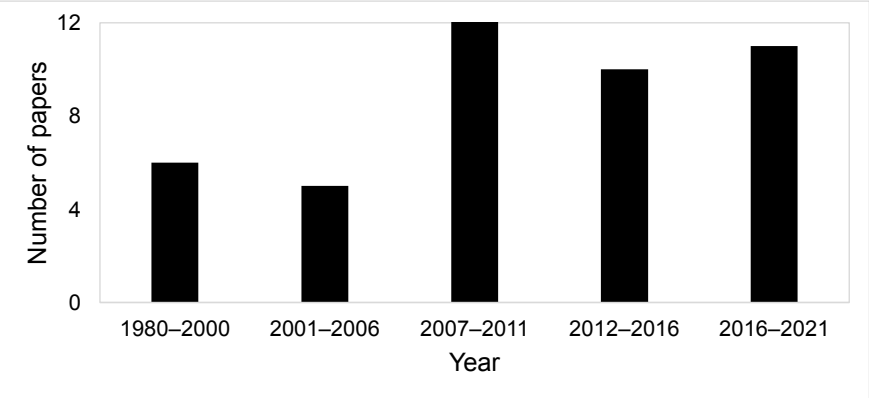

Figure 2.1 : Number of papers published in different time periods (last updated on 01/08/2021).

Table 2.1 : Distribution of papers by journals and conference proceedings.

Journal/Proceeding	Number of papers (%)
Proceedings	7 (15.56)
Journal of Scheduling	6 (13.33)
Discrete Applied Mathematics	4 (8.89)
Computers & Operations Research	3 (6.67)
RAIRO Operations Research	3 (6.67)
Computers & Industrial Engineering	2 (4.44)
European Journal of Operational Research	2 (4.44)
International Journal of Production Research	2 (4.44)
Journal of Combinatorial Optimization	2 (4.44)
Mathematical Methods of Operations Research	2 (4.44)
Naval Research Logistics	2 (4.44)
Operations Research Letters	2 (4.44)
Other outlets with one paper (eight outlets)	8 (17.76)

2.1.1 The single-machine scheduling problem

The majority of the studies on the CTSP are in the context of exact delays in the single-machine environment. In the following we present the researches on the single machine under different objective functions.

Makespan

Most of the available studies in the single-machine environment consider the objective function of minimising the makespan. It should be noted that in the single-machine environment minimising the idle times of the machine is equivalent to minimising the makespan.

Shapiro (1980) discussed that the single-machine CTSP is equivalent to a two-machine job-shop scheduling problem with the following characteristics: (1) every job requires three tasks, where the first task is performed on M_1, the second task, i.e., the delay period, on M_2, and the third task on M_1; (2) machine M_1 may only process one task at a time, however, M_2 has infinite processing capacity; and (3) no waiting time between every pair of tasks of a job is permitted. Hence, the NP-hardness of the problem can be derived from that of the two-machine job-shop problem. Shapiro (1980) also proposed the first heuristic algorithms for the problem. He labelled his three simple heuristics as "sequencing", "nesting" and "fitting" as follows.

The sequencing heuristic, also known as "interleaving", constructs an ordered subset of jobs such that the completion tasks are sequenced for processing in the same order as the initial tasks are scheduled, as shown in Figure 2.2a for two jobs j and j'. The nesting heuristic is similar to the sequencing heuristic; however, the completion tasks are processed in the reverse order of the initial tasks, as shown in Figure 2.2b. The fitting heuristic allows the user to specify a priority order for the jobs, that is applicable to

instances with a small number of jobs.

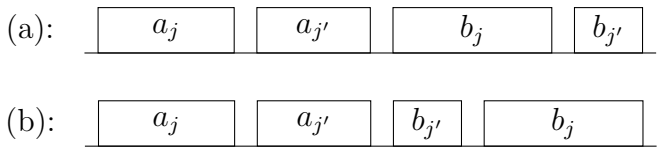

Figure 2.2 : Interleaving jobs j and j' (a) and nesting jobs j and j' (b).

Orman and Potts (1997) presented the symmetry property for the problem, meaning that the makespan minimisation problem defined by (a_j, L_j, b_j) is equivalent to the one defined by (b_j, L_j, a_j), where the latter is called the "reverse" of the former. Sherali and Smith (2005) studied two variants of the problem. The first variant aims at maximising the sum of the weights of the jobs that are completed before the time $\mathcal{T}_{\max}$. The second variant is the makespan minimisation problem. They showed that both variants are strongly NP-hard and proposed two methods to formulate the two variants, namely a discritised model and a continuous model. Condotta and Shakhlevich (2012) showed that a restricted version of the problem, where the sequence for either the initial or completion tasks of all the jobs is given, is NP-hard in the strong sense even if all the jobs have unit execution times (UET).

Ageev and Kononov (2007) proposed an algorithm that works by ordering the jobs in non-increasing order of $a_j + L_j$, i.e., $\text{LPT}_{(a_j+L_j)}$, and showed that it is a 3.5-approximation algorithm for the problem. This job ordering has a time complexity of $O(n \log n)$. Li and Zhao (2007) defined the "singleton" job as the job that can neither be interleaved nor nested. Particularly, job j is a singleton if it cannot be nested within any other job $j' \in N$, nor j' can be nested within j, nor they can be interleaved with each other. They observed that such jobs can be "appended" (appending refers to scheduling a job without any interleaving or nesting moves) one after another at the end of an optimal schedule of the remaining jobs. They also presented a lower bound on the makespan as follows:

$$C^*_{\max} \geq \max\left\{P_a + P_b, \max_{j\in N}\{a_j + L_j + b_j\}, P_a + \min_{j\in N} L_j, P_b + \min_{j\in N} L_j\right\}, \tag{2.1}$$

where $P_a = \sum_{j\in N} a_j$, $P_b = \sum_{j\in N} b_j$, and $C^*_{\max}$ is the optimal makespan.

Li and Zhao (2007) proposed a tabu search (TS) algorithm that applies the interleaving, nesting and appending operations for scheduling a job at the earliest possible time. The TS algorithm of Condotta and Shakhlevich (2012) includes a neighborhood that removes a job from its position and inserts it in another position, utilising a disjunctive graph, where the nodes represent tasks and the arcs show the precedence relations between the tasks. Their TS algorithm outperforms the adapted heuristic of joint decompose

local search proposed for the problem with flexible delays (Potts and Whitehead, 2007) and their own greedy dispatching rule.

Békési et al. (2014) proposed a branch-and-bound (B&B) algorithm. They developed two mathematical models based on linear ordering variables. They also implemented two additional models, a time-indexed model by utilising the discretised model of Elshafei et al. (2004) (that is originally proposed for a radar pulse interleaving problem) and the continuous model of Sherali and Smith (2005). They used the four models to evaluate the performance of their B&B algorithm. In the following, we discuss the studies with additional assumptions, e.g., on the duration or the job sequence.

In their study, Orman and Potts (1997) classified the problem with respect to the restrictions on the duration of the tasks and the delay periods. They showed that the case $(a_j = L_j = b_j)$ is NP-hard in the strong sense, so the cases $(a_j, L_j = b_j)$, $(a_j = b_j, L_j)$ and $(a_j = L_j, b_j)$ are also NP-hard in the strong sense. They also showed the strong NP-hardness of the case (a_j, L, b), implying the same result for the cases (a, L, b_j), (a_j, L_j, b), (a, L_j, b_j) and (a_j, L, b_j). We note that a, L and b indicate that all the jobs have a common initial task, a delay and a completion task, respectively. Even when the initial and completion tasks are equal, i.e., (p, L_j, p) for some constant p, they proved that the case is strongly NP-hard (the same applies to the case (a, L_j, b)). They proposed two optimal algorithms with an $O(n)$ time complexity for the cases (p, p, b_j) and (p, L, p). The former implies that the special cases (a_j, p, p), (p, p, b) and (a, p, p) can also be optimally solved by the same algorithm. There is one special case, however, whose complexity status has remained open to date, where all the jobs have the same tasks and delay, i.e., (a, L, b). This special case is called the "identical" problem.

Ahr et al. (2004) also studied the identical problem. They assumed two conditions. First, $a \geq b$ because the problem is equivalent if $b \geq a$, due to the reverse property discussed earlier. Second, $a < L < (n-1)a$; otherwise, a simple greedy heuristic yields an optimal solution. Under the two assumptions, they proposed a dynamic programming algorithm with an $O(nr^{2L})$ time complexity, where $r \leq \sqrt[a-1]{a}$ (note that $\sqrt[a-1]{a}$ approaches 1 when a increases). This algorithm has a linear time complexity in the number of jobs only when L is fixed; however, the complexity status of the problem still remains open. Studying the identical problem, Baptiste (2010) showed that it can be solved by an algorithm in $O(v^{2v+5})$, where $v \leq (a^{L/(a-1)})^{a^{L/(a-1)}}$. For fixed a, L and b, his algorithm has an $O(\log n)$ time complexity, so is linear in the number of jobs. This implies that the computational complexity of the identical problem still remains open for arbitrary inputs a, L and b. Also, Baptiste (2010) showed that if all the processing times take integer values, then an optimal schedule includes integer starting times.

Yu et al. (2004) showed the strong NP-hardness of the two-machine flow-shop CTSP and UET tasks. Based on the NP-hardness of the flow-shop problem, they implied that

the single-machine CTSP with UET tasks is also strongly NP-hard.

Several studies develop approximation algorithms and ratios for special cases of the problem. For example, Ageev and Kononov (2007) proposed an $O(n \log n)$ algorithm based on $\text{LPT}_{(a_j+L_j)}$, and showed that it is a 3-approximation when $a_j \leq b_j$ or $b_j \leq a_j$, and a 2.5-approximation if $a_j = b_j$. Importantly, they proved that, for any $\varepsilon > 0$, the existence of a $(2-\varepsilon)$-approximation algorithm for the problem implies $P = NP$, even if $a_j = b_j$. Ageev and Baburin (2007) studied the case of UET tasks. They proposed an $O(n \log n)$ algorithm based on non-decreasing ordering of L_j, i.e., $\text{SPT}_{(L_j)}$, and showed that it is a 1.75-approximation. Békési et al. (2009) later argued that there are some flaws in the approximation algorithm of Ageev and Baburin (2007). They re-calculated the ratio as $\frac{28}{19} \leq \rho \leq \frac{7}{4}$. Ageev and Ivanov (2016) studied the case with equal delays, i.e., (a_j, L, b_j). They proposed an $O(n \log n)$ constructive algorithm with a 3-approximation ratio. They showed that the $\text{LPT}_{(a_j+L_j)}$ algorithm leads to ratios of 2 when $a_j \leq b_j$, and 1.5 when $a_j = b_j$. Also, they showed that the 2-approximation algorithm provides the same ratio for the identical problem. In addition, for any $\varepsilon > 0$, the existence of a $(1.25-\varepsilon)$-approximation algorithm implies $P = NP$, even when $a_j = b_j$.

Li and Zhao (2007) studied the special case where the initial and completion tasks, and the delay have the same value for every job, i.e., (p_j, p_j, p_j). For this case they showed that no two jobs can be interleaved, and that any schedule without forced idle time yields a makespan value that is no greater than 1.5 times an optimal (a forced idle time is a time period over which the machine is forced to be idle because no job is available). For the problem (a_j, L, b), they concluded that no nesting is possible for any pair of jobs. They showed that a schedule with the makespan at most three times of an optimal one can be constructed in linear time.

There are a number of studies with restrictions on the job sequence, that include precedence relationships, compatibility and fixed-job-sequences. In the following we discuss research in those areas. Precedence constraints model jobs' dependency, i.e., when a job is required to be processed before another job. Precedence constraints are usually defined by using a precedence graph, which can be in a general form, or in a special form such as a chain or a star. The CTSP with precedence constraints is usually harder to solve. For example, the case (p, L, p) is polynomially solvable (Orman and Potts, 1997); however, if precedence constraints are included, the problem is not necessarily solvable in polynomial time.

Blazewicz et al. (2010) investigated the problem with UET tasks and two types of precedence constraints, namely strict and weak precedence constraints. For two jobs j and j', the strict precedence constraint $N_j \prec N_{j'}$ (N_j precedes $N_{j'}$) means $b_j \prec a_{j'}$, i.e., b_j must be completed prior to the start of $a_{j'}$. On the other hand, the weak precedence constraint $N_j \rightarrow N_{j'}$ means $a_j \prec a_{j'}$ and $b_j \prec b_{j'}$, i.e., a_j must be completed prior to

the start of $a_{j'}$ and b_j must be completed prior to the start of $b_{j'}$. They showed that the problem with strict precedence constraints is NP-hard in the strong sense, when L is arbitrary. However, for the special case where the delay duration is equal to two and the precedence graph is an in/out-tree (the in-tree (out-tree) precedence constraints occur when every job has at most one immediate successor (predecessor)), the problem can be solved in $O(n)$ time. For this reason, they developed an algorithm that operates by utilising the rule proposed by McNaughton (1959) for the parallel-machine scheduling problem.

Ecker and Tanaś (2012) studied the problem with UET tasks, equal delays, and strict precedence constraints in the form of chains. Based on the McNaughton's rule, they proposed an algorithm that solves the problem in $O(n \log n)$ time if L is a fixed constant. Blazewicz et al. (2012) studied a very special case of the problem where $L = 4$. They proposed a greedy heuristic and an optimal algorithm with an $O(n \log n)$ time complexity. As a generalisation of the two algorithms, they presented an approximation algorithm that works for any $L = 2r, r \in \mathbb{N}$, and runs in $O(n \log n)$ time. The solution delivered by the approximation algorithm is at most L times worse than that of an optimal.

Two jobs j and j' are compatible if at least one of the tasks of either of jobs can be processed during the idle time of the other job. The compatibility of jobs can be represented by a compatibility graph, in which an edge is associated with every two compatible jobs. Simonin (2009) showed that the identical CTSP with compatibility constraints is NP-hard. Later, Simonin et al. (2011a) showed that the identical problem can be solved in polynomial time if $L < a + b$. However, it is NP-hard if $L = a + b$. They proposed a ρ-approximation algorithm, where ρ depends on the values of a and b ($a > b$), $1.25 \leq \rho \leq 1.5$. Simonin et al. (2011b) investigated the identical problem with precedence constraints, incompatibility graph, and UET tasks. They showed that the problem is NP-hard when $L \geq 3$. They presented an approximation algorithm with a ratio of $\frac{L+6}{6} - \frac{1}{2(L+2)} + \frac{L+3}{6n(L+2)}$ (it is trivial though if $L = 1$). When $L = 2$, Simonin et al. (2013) studied the problem under the conditions of with and without "triangles" in the compatibility graph. They showed that the problem is NP-hard under both conditions, and presented a $\frac{10}{9}$ ($\frac{13}{12}$)-approximation algorithm for the existence (lack) of triangles. Darties et al. (2016) studied the case (p_j, p_j, p_j) and denoted p_j as the stretch factor of job j. They showed that the problem can be solved in $O(n^3)$ time when the compatibility graph is a chain and it is NP-hard when the graph is a star. For the problem with a chain or a 2-stage bipartite compatibility graph, they proposed $\frac{7}{6}$ and $\frac{13}{9}$-approximation algorithms (a 2-stage is a bipartite graph that has three sets of disjoint nodes. The first set is disjoint, but connected to the second set, which itself is also disjoint but connected to the third set).

Recently, Bessy and Giroudeau (2019) investigated the parameterised complexity of

the CTSP with compatibility constraints. Given a fixed due date d for all the jobs, the parameterised analysis includes whether a schedule in which at least a fixed number of jobs are completely processed before d exists. They proved that the problem is fixed-parameter tractable (FPT) when the total duration of every job is bounded by a constant, and also showed that the problem is W[1]-hard otherwise. Particularly, they showed that for every $\zeta \geq 4$, where ζ is an upper bound on the total duration of a job, i.e., $a_j + L_j + b_j \leq \zeta$, an FPT algorithm exists for the CTSP with a time complexity of $2^{O(k)}n^{2\zeta}\log^2 n$, where $k \leq n$.

Hwang and Lin (2011) studied the problem subject to a fixed-job-sequence (*fjs*), assuming weak precedence constraints. Note that the problem with the *fjs* assumption and strict precedence constraints is trivial. They argued that although *fjs* implies a preassigned job sequence, the problem remains a sequencing one due to the interleaving jobs. They developed a procedure to construct a schedule for a given sequence, and showed that if such a sequence is feasible, then the schedule has the minimum makespan among all the feasible sequences. The procedure also concludes the infeasibility of a given sequence in $O(n^2)$ time. While the complexity status of the problem is open, they proposed polynomial-time procedures for three special cases of the problem, namely (p_j, p_j, p_j), (p, p, b_j) and (p, L, p).

The problem with a convex resource function is also recently studied by Mosheiov et al. (2021). They investigated the case (p, p, b_j), where the processing time of the completion tasks is modelled as a convex function of the amount of the resource allocated to job j. While setting an upper bound on the resource availablity, they proposed an $O(n^2)$-time algorithm to minimise the makespan.

Total completion time

Recently, Chen and Zhang (2020) studied the single-machine CTSP with the objective function of minimising the total completion time. Their results indicate that the complexity of the problem under the total completion time criterion is analogous to that of the makespan criterion. Precisely, they showed that cases $(a_j = L_j = b_j)$, (p, L_j, p), (a_j, L, b), and (a, L, b_j) are strongly NP-hard. Note that the last two cases are equivalent (due to the reverse property) in the makespan setting. However, the reverse property does not hold for the total completion time criterion (Chen and Zhang, 2020). Similar to the makespan setting, they managed to propose polynomial-time algorithms for the cases (p, p, b_j), (a_j, p, p), and (p, L, p). Lastly, they showed that the case (a, L, b) is polynomially solvable only if a, L, and b are fixed values.

Cyclic scheduling

Ahr et al. (2004) investigated an interesting structure of an optimal schedule for the identical problem when the number of jobs approaches infinity. They argued that an

optimal schedule consists of three parts, namely an "initial" segment, followed by a certain number of repetition of cycles (the "middle" part), and a "finishing" segment. They stated that the middle segment repetitions must contain the minimum mean cycle. For the special case where $b = 1$, they conjectured a formula for the minimum mean cycle time. They also presented optimal ratios and mean cycle times for certain small instances. Later, Brauner et al. (2009) linked the problem to the one-machine no-wait robotic cell problem and adapted the algorithm of Ahr et al. (2004) to obtain production patterns.

Lehoux-Lebacque et al. (2015) continued work on the identical problem in the cyclic case. They considered the objective function of maximising the throughput, i.e., $\frac{T(C)}{N(C)}$, where $T(C)$ is the cycle time and $N(C)$ is the number of tasks in the cycle C. It should be noted that maximising the throughput is equivalent to minimising the mean cycle time $(\frac{N(C)}{T(C)})$. They studied the problem for $a > b$, as for $b > a$ the reverse property holds and for $a = b$ it is trivial. Likewise, the condition $L > a + b$ is imposed; otherwise, an optimal cycle can be easily obtained. They studied all the possible patterns that may occur in a cycle, and presented an algorithm with a time complexity of $O((\log L)^2)$ to find an optimal cycle. In addition, considering $\beta = \lfloor \frac{L}{a+b} \rfloor$, if $L - \beta(a + b) \leq a$, an optimal mean cycle is the triple $(a, \frac{L-\beta(a+b)}{\beta+1}, b)$, i.e., the mean cycle consists of one initial task a, an idle time, which is equal to $\frac{L-\beta(a+b)}{\beta+1}$, and one completion task b. Such results let them arrive at the conjectures of Ahr et al. (2004). The results of Lehoux-Lebacque et al. (2015) enable the finding of an optimal schedule for the middle segment when n is very large. However, obtaining an optimal schedule for the initial and finishing segments is not trivial.

2.1.2 The shop scheduling problem

Most of the published studies on the CTSP in the shop environment are conducted in the flow-shop context. Therefore, we first discuss the flow-shop environment, based on the difference in performance criteria. We then present research in job-shop and open-shop scheduling environments.

The classical flow-shop scheduling problem includes a set N of jobs to be processed on a set M of different machines. Job j consists of a set of m tasks to be processed in the "same" sequence on the m machines.

The CTSP in the flow-shop environment deals with the situation in which there are "exact" delays between every pair of consecutive tasks of a job. All the tasks have known processing times and all the delays have known durations. In addition, the delay durations are strict, i.e., once the delay period is elapsed, the processing of the next task must immediately start. No two tasks can be processed at the same time on one machine and preemption is not allowed. However, tasks of other jobs can be processed during the delay period. Figure 2.3 shows a schematic of the m-machine flow-shop CTSP.

The flow-shop CTSP is a generalisation of the no-wait flow-shop scheduling problem.

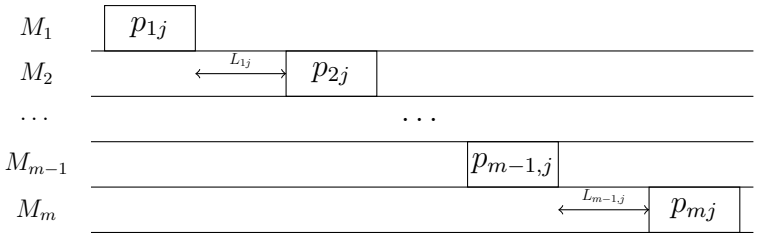

Figure 2.3 : Schematic of the m-machine flow-shop CTSP.

In other words, in case we allow for zero-time delays in the flow-shop CTSP, the no-wait flow-shop problem is a special case of the flow-shop CTSP, where $L_{kj} = 0, \forall k \in M \setminus \{m\}, \forall j \in N$. Recall that in the no-wait flow-shop, once the processing of a job is started, its tasks must be processed on all the machines (from the first to the last) without any interruption.

Makespan

The problem $F2||C_{\max}$ is one of the first and well studied scheduling problems. Johnson (1954) proposed an $O(n \log n)$ algorithm to find an optimal schedule for the problem, which is a permutation one. A permutation schedule is any schedule in which the processing orders of the jobs on all the m machines are the same. For the problem $F2|L_j|C_{\max}$ there exists an optimal schedule, which is also a permutation one, if $L_j = L$, and an optimal schedule is obtained in $O(n \log n)$ time (Leung et al., 2007). However, they did not discuss how the algorithm works. We observe that the algorithm proposed by Gilmore and Gomory (1964) for the two-machine no-wait flow-shop problem is optimal for $F2|L_j = L|C_{\max}$ as well. This is because in any sequence for $F2|L_j = L|C_{\max}$, the delay duration of the first job leads to shifting the completion tasks of all the jobs forward, i.e., $C_j^{L_j=L} = C_j^{\text{no-wait}} + L$. Moreover, there is no job that can be swapped in order to use this delay duration because all the jobs have an identical delay. Hence, an optimal solution for the no-wait case yields an optimal solution for the problem $F2|L_j = L|C_{\max}$, where $C_{\max}^{L_j=L} = C_{\max}^{\text{no-wait}} + L$.

However, the problem is not trivial if arbitrary delays are considered. For example, Yu et al. (2004) showed that even if two distinct values are considered for the delays, the two-machine flow-shop CTSP to minimise the makespan is strongly NP-hard. This result holds even for the case with UET tasks. It should be noted that an optimal schedule for the problem with arbitrary delays is not necessarily a permutation one. Leung et al. (2007) discussed an example for this, which is shown in Figure 2.4. For two jobs $N_1 = (1, 8, 1)$ and $N_2 = (2, 1, 3)$, the figure shows that the optimal makespan is 10. However, the makespan of the best permutation schedule is 12.

Leung et al. (2007) also presented a simple lower bound for $F2|L_j|C_{\max}$ as follows:

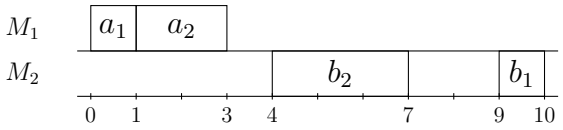

Figure 2.4 : An example showing that an optimal schedule for $F2|L_j|C_{\max}$ is not necessarily a permutation one.

$$C^*_{\max} \geq \max\Big\{P_a, P_b, \max_{j\in N}\{a_j + L_j + b_j\}\Big\}, \tag{2.2}$$

where $P_a = \sum_{j\in N} a_j$ and $P_b = \sum_{j\in N} b_j$.
For the m-machine case $Fm|L_j|C_{\max}$, Hamdi and Loukil (2017) proposed a lower bound as follows:

$$C^*_{\max} \geq \max_{k\in M\setminus\{m\}}\Big\{\sum_{j\in N} p_{kj} + \sum_{k'\in M\setminus\{m\}} \min_{j\in N}\{p_{k'j} + L_{k'j}\}\Big\}. \tag{2.3}$$

The lower bound indicates that the minimum completion time of a machine is equal to the sum of its workload (all the processing times) plus its earliest starting time.

For the two-machine flow-shop CTSP, Ageev and Kononov (2007) presented a two-phase algorithm. The first phase sequences the jobs in $\mathrm{SPT}_{(a_j+L_j)}$. The second phase constructs the schedule for that sequence. They showed that this simple algorithm provides a 3-approximation ratio for the problem, and that the ratio is relatively small because, for any $\varepsilon > 0$ the existence of a $(1.5-\varepsilon)$-approximation algorithm for the problem implies $P = NP$, even if $a_j = b_j$. They discussed the same algorithm provides a 2-approximation ratio for the special case where $a_j \leq b_j$ or $b_j \leq a_j$. The algorithm can be implemented in $O(n \log n)$ time.

Ageev and Baburin (2007) proposed a two-phase algorithm for the two-machine flow-shop CTSP with UET tasks. Here, the first phase includes sequencing the jobs in $\mathrm{SPT}_{(L_j)}$. They showed that the algorithm provides a 1.5-approximation ratio for the problem with a time complexity of $O(n^2 \log n)$. They did not analyze the tightness of the ratio, which remains as an open question. Both of the algorithms of Ageev and Kononov (2007) and Ageev and Baburin (2007) are essentially the same when UET tasks are considered because $a_j = 1, \forall j \in N$, and sequencing the jobs in $\mathrm{SPT}_{(a_j+L_j)}$ leads to the same order obtained by $\mathrm{SPT}_{(L_j)}$. To conclude, for the two-machine flow-shop CTSP, sorting the jobs in $\mathrm{SPT}_{(a_j+L_j)}$ leads to a 3-approximation algorithm for the general case, to a 2-approximation if $a_j \leq b_j$ or $b_j \leq a_j$, and to a 1.5-approximation for UET tasks.

A common assumption in scheduling research is that the processing times take positive integers. Leung et al. (2007) considered a variant in which at least one positive task is given for every job. In other words, there may be jobs with only one positive task. For

convenience, we use F^g to denote this variant and F to denote the classical case. It should be noted that F^g is a generalisation of F. In their study, if one of the tasks of a job has a zero processing time, the delay of the job also has a zero value. On the other hand, if the delay duration of a job takes a non-zero value, both tasks of the job have positive processing times. However, there might be a job with positive processing times for its two tasks and a zero-duration delay.

Leung et al. (2007) showed that the problem $F2^g|L_j \in \{\lambda_1, \lambda_2\}|C_{\max}$ is strongly NP-hard because the F^g variant of the two-machine no-wait flow-shop problem is strongly NP-hard (Sahni and Cho, 1979). They proposed a 3-approximation algorithm, which is applicable to the $F2$ and $F2^g$ problems in $O(n \log n)$ time. The algorithm sequences the jobs in $\text{SPT}_{(L_j)}$ and then obtains the schedule for the sequence. They showed that when $a_j \geq b_j$, sequencing the jobs in $\text{SPT}_{(a_j+L_j)}$ leads to a 2-approximation algorithm. The same argument applies to the $F2^g$ variant. Likewise, sequencing the jobs in $\text{LPT}_{(L_j+b_j)}$ provides a 2-approximation algorithm for the case $a_j \leq b_j$, for both the F and F^g variants. They showed that the ratios are tight. Recently, Ageev (2019) developed an $O(n \log n)$-time 2-approximation algorithm for $F2|L_j \in \{\lambda_1, \lambda_2\}|C_{\max}$, and showed that the existence of a $(1.25 - \varepsilon)$-approximation algorithm for the problem implies $P = NP$, if $\lambda_1 = 0$.

Total completion time

Leung et al. (2007) studied the flow-shop CTSP to minimise the total completion time. They applied the NP-hardness of the two-machine no-wait flow-shop scheduling problem to minimise the total completion time (Röck, 1984a) to imply that the following two-machine problems are strongly NP-hard: (1) the F variant with equal delays and (2) the F^g variant with arbitrary delays.

They observed that a schedule is optimal if (1) the initial tasks are scheduled in non-decreasing order, and from time zero with no idle time between them, or (2) the completion tasks are scheduled in non-decreasing order, and from time $\min_{j \in N}\{a_j + L_j\}$ with no idle time between them. Based on these observations, they proposed a lower bound on the total completion time on two machines. Let $a_1 \leq a_2 \leq \cdots \leq a_n$ denote the processing times of the tasks on the first machine, sorted in non-decreasing order of the values. Similarly, let $b_1 \leq b_2 \leq \cdots \leq b_n$ denote the processing times of the tasks on the second machine, again sorted in non-decreasing order of their values. Let $P'_a = \sum_{i \in N}(n-i+1)a_i$, $P'_b = \sum_{i \in N}(n-i+1)b_i$ and $L' = \sum_{j \in N} L_j$, and the lower bound is as follows:

$$C^* \geq \max\left\{P'_a + L' + P_b, \min_{j \in N}\{a_j + L_j\} + P'_b\right\}, \tag{2.4}$$

where C^* is the optimal value of the total completion time.

Another interesting result of their study includes generating an optimal schedule for a special case of the F variant where $a_j = a, b_j = b$, and $a \geq b$ by sequencing the jobs in

$\text{SPT}_{(L_j)}$. This algorithm, however, is not optimal if $a < b$ and provides a 2-approximation ratio instead.

Huo et al. (2009) provided the major results for the total completion time criterion. They studied the permutation version of the flow-shop CTSP. They showed that, as for the makespan minimisation criterion, an optimal schedule for the two-machine flow-shop to minimise the total completion time is not necessarily a permutation schedule. This can be shown by the following example. Consider three jobs $N_1 = (1, 5, 3)$, $N_2 = (1, 9, 2)$ and $N_3 = (3, 3, 3)$, where a non-permutation schedule generates the minimum total completion time of 35, whereas that of a permutation one is 38. They argued that forced idle time will never improve a permutation schedule under arbitrary delays. Therefore, they assumed no forced idle time in any feasible schedule for the problem. Because any feasible schedule for the two-machine no-wait flow-shop scheduling problem, which itself is NP-hard, is a permutation schedule, they implied that the permutation flow-shop CTSP is strongly NP-hard.

They investigated several special cases as well. For the first case, let the jobs be ordered in such a way that $a_j \leq a_{j+1}$. If $a_{j+1} + L_{j+1} \geq L_j + b_j$ for all $1 \leq j \leq n-1$, the problem is optimally solved by sequencing the jobs in $\text{SPT}_{(a_j)}$. For the second case, assume that the jobs are ordered in such a way that $b_j \leq b_{j+1}$. If $a_1 + L_1 = \min_{j \in N}\{a_j + b_j\}$ and $a_{j+1} + L_{j+1} < L_j + b_j$ for all $1 \leq j \leq n-1$, the problem is also optimally solved by sequencing the jobs in $\text{SPT}_{(b_j)}$. The third case is when $\max_{j \in N} a_j \leq \min_{j' \in N} b_{j'}, L_j = L, \forall j \in N$, which is also polynomially solvable by a $\text{SPT}_{(b_j)}$-based algorithm that obtains the minimum total completion time from a total of n schedules. Finally, for $F2|perm, L_j, a_j = a, b_j = b|\sum C_j$, sequencing the jobs in $\text{SPT}_{(a_j + L_j + b_j)}$ is optimal. The same rule also leads to an optimal solution for $F2|perm, L_j = L, a_{j'} < a_j \Rightarrow b_{j'} < b_j|\sum C_j$.

Huo et al. (2009) proposed two meta-heuristics, namely simulated annealing and tabu search, for the general case of the problem. According to the computational results, SA performs slightly better than TS in terms of both solution quality and computing time. In addition, they showed that a greedy heuristic, which inserts the jobs one by one such that the completion time of the inserted job has the smallest value, performs better than sorting the jobs in non-decreasing order of any of the following: a_j, b_j, $a_j + L_j$, $L_j + b_j$ and $a_j + L_j + b_j$.

It should be noted that the problem to minimise the total completion time is computationally less demanding than that to minimise the makespan. For example, $F2|L_j, a_j = a, b_j = b, a \geq b|\sum C_j$ and so $F2|L_j, a_j = b_j|\sum C_j$ are polynomially solvable, however, $F2|L_j, UET|C_{\max}$ is NP-hard.

Maximum lateness

Fondrevelle et al. (2009) studied the m-machine flow-shop CTSP to minimise the maximum lateness. They first showed that permutation schedules are not dominating even if only one job has a delay greater than 0.

They defined the lateness of a job based on its completion time on any machine $k < m$ and distinguished three different forms of the jobs. They presented a special case of the problem for which the earliest due date (EDD) dispatching rule provides an optimal schedule. In addition, they developed several dominance rules for the two-machine problem and proposed a B&B algorithm. They used the well-known algorithm of Nawaz, Enscore and Ham (NEH) (Nawaz et al., 1983) to build upper bounds, and the EDD rule and an extension of the algorithm of Gilmore and Gomory (1964) (for the no-wait flow-shop problem) to generate lower bounds.

Earliness and tardiness penalties

Hamdi and Loukil (2017) studied permutation schedules for the flow-shop coupled task problem to minimise the total earliness and tardiness penalties. They proposed three mathematical formulations, namely completion time-based, idle time-based and starting time-based. They also discussed three lower bounds. The first is a linear relaxation bound and the second one is calculated by summing two bounds on the values of total earliness and tardiness. The third bound is obtained by relaxing the processing times and delays of the jobs. Also, they used simple sequencing rules to produce upper bounds.

Other variants of coupled task flow-shops

As reviewed, the flow-shop CTSP with an exact delay period between every two consecutive tasks is an important variant. In this section we discuss two additional variants with exact delays in the flow-shop environment.

Two-machine problem with coupled tasks on the first machine. Meziani et al. (2018) studied the two-machine flow-shop problem to minimise the makespan, in which the coupled tasks with an exact delay are only considered on the first machine. Specifically, every job consists of three tasks, two coupled tasks on the first machine, followed by a single task on the second machine. The processing times of the tasks of job j are represented by (a_j, L_j, b_j, c_j), in which a_j and b_j are the coupled tasks with an exact delay L_j between them, and c_j represents the third task. The task on the second machine can start if the tasks on the first machine are finished. Figure 2.5 shows a schematic presentation of this problem. It should be noted that the problem is NP-hard because the one on the first machine is NP-hard. Meziani et al. (2018) observed that the makespan is at least equal to (1) the sum of all the tasks on M_1 plus at least one task on M_2, (2) the smallest job on M_1 plus the sum of all the tasks on M_2, (3) a lower bound on M_1,

where $b_j = 0, \forall j \in N$, plus at least one task on M_2 and (4) a lower bound on M_1, where $a_j = 0, \forall j \in N$, plus at least one task on M_2. Based on these observations, they proposed a lower bound for the problem as follows:

$$LB = \max\Big\{ \sum_{j\in N}(a_j + b_j) + \min_{j\in N} c_j, \min_{j\in N}\{a_j + L_j + b_j\} + \sum_{j\in N} c_j, \\ \sum_{j\in N} a_j + \min_{j\in N} L_j + \min_{j\in N} c_j, \sum_{j\in N} b_j + \min_{j\in N} L_j + \min_{j\in N} c_j \Big\}. \tag{2.5}$$

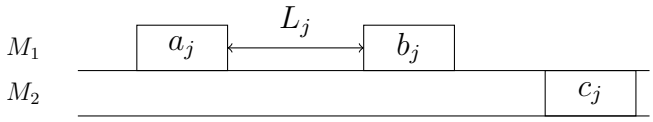

Figure 2.5 : Schematic of the two-machine flow-shop problem with coupled tasks on the first machine, and a single task on the second machine.

They also proposed four SPT- and LPT-based heuristics and two meta-heuristics, namely particle swarm optimisation (PSO) and simulated annealing (SA). They further designed a hybrid algorithm, in which SA guides PSO to avoid being trapped in low-quality local optima. Meziani et al. (2019) extended their previous study and examined the complexity status of several cases. For example, they proved the NP-hardness of the cases (a_j, p, p, c_j) and (p, p, b_j, c_j). For the cases (a, p, p, c_j), (p, p, b, c_j) and (p, L, p, c_j), they proposed optimal solutions by extending the algorithm of Johnson (1954), which has an $O(n \log n)$ time complexity. They also proposed an $O(n^2 \log n)$-time algorithm, by modifying the algorithm of Mitten (1959), to solve the cases (a_j, p, p, c_j) and (p, p, b_j, c_j).

The chain re-entrant flow-shop problem. Amrouche and Boudhar (2016) studied a variant in which the coupled tasks are considered in the re-entrant flow-shop scheduling environment. In the re-entrant flow-shop problem, the jobs may visit any machine more than once. There are various types of the re-entrant shop, such as the chain re-entrant and V-shop. In the chain re-entrant problem, the jobs visit M_1, followed by the rest of the $m-1$ machines, and return back to M_1 for their terminating task. In the V-shop problem, the jobs follow the route $M_1, M_2, \ldots, M_{m-1}, M_m, M_{m-1}, \ldots, M_2, M_1$. Note that for the two-machine flow-shop, the chain re-entrant and V-shop are identical. Amrouche and Boudhar (2016) studied the two-machine chain re-entrant problem to minimise the makespan, i.e., the jobs pass in the order M_1, M_2, M_1. Here, the processing times of the tasks of job j can be represented by the triple (a_j, b_j, c_j), where a_j and c_j are processed on the first machine and b_j is processed on the second machine. We call this problem the "Ch-R". Figure 2.6 illustrates this.

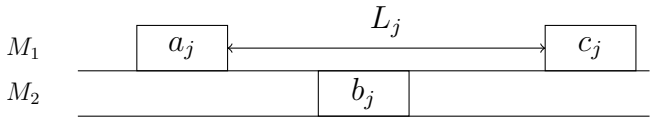

Figure 2.6 : The two-machine chain re-entrant flow-shop CTSP.

First, they considered $L_j = L$ between two tasks of every job on the first machine. They showed that even $a_j = c_j = p$ leads to a strongly NP-hard problem. Therefore, the general form of the problem is also strongly NP-hard. They showed that the special case where $a_j > \frac{L}{2}, c_j > \frac{L}{2}$ reduces to the maximum weight matching problem, so is solvable in $O(n^{2.5})$ time. They also showed that the case where $b_j = L_j = L \geq a_j + c_j$ is polynomially solvable.

Amrouche et al. (2017) showed that when $b_j = L_j$, the problem is NP-hard in the strong sense. In addition, they showed that the special case of the problem where $b_j = L_j = L$, $a_j + c_j > L$ reduces to the maximum weight matching problem, so is solvable in $O(n^{2.5})$ time. The case where $a_j = a \geq c_j$ and $b_j + b_{j'} \leq 2a = L_j$ is also polynomially solvable. For the special case with identical delays, they proposed a heuristic procedure with nine job arrangement rules. Eight rules include sorting the jobs in non-increasing and non-decreasing order of a_j, b_j, c_j and $a_j + c_j$. The ninth one is a heuristic that generates the schedule with some best fitted batches of the jobs, where each batch consists of a subset of interleaving jobs, and a union of all the batches form a complete solution. Their experiments concluded that both the heuristic and sequencing rule in non-increasing order of $a_j + c_j$ perform the best.

Liu et al. (2017a) extended the setting of Amrouche and Boudhar (2016) and modelled the time-in-use electricity prices for energy consumption as a bi-criteria re-entrant flow-shop. They investigated the two objective functions of minimising the makespan and minimising the total energy consumption. They developed a mathematical formulation and proposed two solution methods.

The job-shop problem

The classical job-shop scheduling problem includes a set N of jobs to be processed on a set M of different machines. Each job has a pre-specified processing order on the machines. The job-shop CTSP is similar to its flow-shop counterpart with the only difference that each job has its own pre-specified order in which its tasks are processed on the machines.

Gröflin and Klinkert (2007) first considered the job-shop CTSP. They studied an optimal job insertion in the job-shop environment, where the delays can also take negative values, and the objective is to minimise the makespan. The insertion issue arises when additional tasks or jobs are inserted into a given schedule. They showed that the job insertion problem is NP-hard in the classical job-shop environment; however, it is poly-

nomially solvable in the job-shop CTSP.

Bürgy and Gröflin (2013) studied a similar job insertion problem, where sequence-dependent setup times between consecutive tasks on a machine are present. They proposed an $O(n^2 \max\{n, m\})$-time algorithm to solve the problem. They also developed a heuristic based on their optimal job insertion algorithm. Later, Bürgy and Gröflin (2017) generalised their earlier results to the case of minimising a general regular objective function.

The open-shop problem

The classical open-shop scheduling problem includes a set N of jobs to be processed on a set M of different machines. Job j consists of m tasks, which can be processed in any order on the m machines. The open-shop CTSP is similar to its flow-shop counterpart with the only difference that the order in which the tasks of a job are processed on the machines is immaterial.

Due to the NP-hardness of the two-machine no-wait open-shop problem (Giaro, 2001), Ageev (2018) discussed that the two-machine open-shop CTSP is also NP-hard. They also proved that the existence of a $(1.5 - \varepsilon)$-approximation algorithm for the special case where $a_j = b_j$ implies $P = NP$, for any $\varepsilon > 0$. They further showed that when the delays can take only two values, there is no approximation algorithm with a ratio better than $(1.5 - \varepsilon)$ for any $\varepsilon > 0$, unless $P = NP$.

2.2 Applications

Shapiro (1980) introduced the CTSP based on the following application: In a radar tracking system, pulses are transmitted and reflections are received once every specified update period. The radar cannot transmit a pulse at the same time when a reflected pulse is arriving; also, two reflected pulses cannot overlap. The radar devices transmit certain pulses to calculate the sizes, shapes and speeds of the tracked objects. Hence, one can model the transmission and reflection of pulses as two tasks with a fixed duration of delay between them. It is important that the idle time of the radar system is minimised. In a similar context, Simonin (2009) made use of the CTSP to improve the performance of submarine torpedoes. Various environmental data must be processed by sensors located on the torpedo. Here, the initial task is the transmission of a pulse from the torpedo to the water and the completion task consists of receiving the echo, while there is an exact duration of idle time between the two tasks.

In the flow-shop setting, Ageev and Baburin (2007) modelled a chemistry manufacturing process as the CTSP. Specifically, two different operators execute the operations successively, where there is an exact technological delay between the finishing time of the first task and the starting time of the second one. Brauner et al. (2009) showed that

minimising the makespan for the single-machine identical CTSP is equivalent to maximising the throughput rate in a one-machine no-wait robotic cell with the characteristics of having an input station, an output station and one machine. Raw material and finished products can be stored in input and output stations with infinite capacity. Any number of products can be treated simultaneously by the machine. A single transporter with a unit capacity carries materials between the stations and the machine.

Recently, the CTSP has been utilised to model certain problems in the health care domain. As an example, for chemotherapy appointment scheduling, once the medicine is prescribed, the patient visits the health centre on treatment days separated by a fixed number of rest days (Condotta and Shakhlevich, 2014). Some other health care environments with multi-stage characteristics can also be modelled as the CTSP. Consider a pathology laboratory, where minimising the patients' waiting times is the performance measure (Marinagi et al., 2000; Azadeh et al., 2014). Certain blood tests, e.g., fasting blood sugar testing, require multiple tests and there is an exact time delay between a pair of tests. Another problem with multi-stage characteristics includes patient appointment scheduling in nuclear medicine clinics. Due to the very strict multi-stage sequential procedures, the problem in the nuclear medicine clinic is more complex than its counterpart in the typical medical imaging clinic (Pérez et al., 2011; Pérez et al., 2013). Here, a single treatment requires multiple steps and each step needs to be completed within a strict time window. Maximising the number of treated patients is well justified in this context due to the costs of the required resources and the short half-lives of the radio-pharmaceuticals needed for the treatments. Strict time window requirements also exist in the hemodialysis treatment services (Liu et al., 2019).

In the context of healthcare applications, the tiredness of the staff, that typically occurs in practice and impacts the processing time and therefore, the starting time of the next job, can be modelled as a time-dependent event. For example, Pérez et al. (2013) highlights the importance of modelling human resource fatigue within the patient appointment scheduling in nuclear medicine clinics or in similar environments, proposing therefore studies on the CTSP with time-dependent processing times.

There are two other motivations incorporating delays in shop scheduling problems. First, it can be used to model a no-wait problem, where there is a transport time (or any processing that does not require a processing machine) between two consecutive tasks of the same job. That time requirement can indeed be represented by the delay duration in the CTSP. Chu and Proth (1996) presented a class of problems in which a transport time between successive tasks of a job occurs. Fondrevelle et al. (2009) discussed similar situations arising in the manufacturing process of thermic paper that involves chemical processing, where temporal constraints are imposed on the process. Likewise, such constraints must be satisfied in every processing step of a pharmaceutical plant,

from raw materials and resource preparation to packaging. Second, it can be used to model the no-wait condition involving bottleneck machines. According to Mitten (1959), the two-machine flow-shop problem with delays can present the industrial environment involving two bottleneck machines, where the jobs must undergo a number of tasks on some intermediate machines between the two bottleneck machines. The elapsed time on those intermediate machines can be represented by the delay durations, where the delays are exact in the no-wait situation.

Chapter 3

Benchmarks and mathematical models

This chapter is devoted to the benchmark instances and mathematical formulations of the coupled task scheduling problem (CTSP). Precisely, in Section 3.1 we discuss the utilised data sets in the literature, and then propose new sets of benchmarks. All available mathematical formulations of the problem are then presented and discussed in Section 3.2. Some corrections and improvements to the existing models, and a new formulation are proposed. The chapter ends in Section 3.3 with comparing the performance of the models based on the results of extensive computational experiments. The results presented in this chapter are published in:

- Khatami, M., Salehipour, A., and Cheng, T. C. E. (2020). "Coupled task scheduling with exact delays: Literature review and models". *European Journal of Operational Research* 282(1), 19 –39.

3.1 Benchmark instances

Several studies generated their own instances to evaluate their proposed algorithms. Often, those instance generation schemes include small- and medium-sized instances, and they may only be applicable to certain special cases. Also, because previously generated instances were not published in the literature, there is a demand for standard test beds; particularly, for evaluation and comparison purposes. Therefore, we propose comprehensive sets of instances for the CTSP, and for both the single-machine and shop environments.

3.1.1 Previous instance generation schemes

Shapiro (1980) proposed the first instance generation scheme. He generated a set of problem instances by simulating a radar process. The instances set includes a total number of 250 instances with 20 to 500 jobs, where the task processing times are in the range of 10 and 100, and the delay durations in the range of 300 and 1,300. Ahr et al. (2004), Li and Zhao (2007), Brauner et al. (2009), and Blazewicz et al. (2012) utilized a discrete uniform interval as well.

The instances in Sherali and Smith (2005) consist of 8 to 14 jobs, with a_j taken from the normal distribution with a mean of 30 time units and a standard deviation of 5 time units, i.e., $N(30, 5)$. Likewise, b_j and L_j are taken from $N(120, 30)$ and $N(600, 100)$, respectively.

Condotta and Shakhlevich (2012) defined three types of delay in their instance generation. Delays in the first type are in a small range, while in the second and third types, delays take considerably different values. In a set of data in Békési et al. (2014), a_j and b_j take values in the range of 1 and 100, and the values of L_j are generated such that the jobs are perfectly "fitted", i.e., the jobs can be scheduled in such a way that the makespan is at most one unit greater than the trivial lower bound $P_a + P_b$.

Considering the studies in the shop setting, the discrete uniform distribution is used by all the works with a data generation scheme, including Huo et al. (2009), Fondrevelle et al. (2009), Hamdi and Loukil (2017), Amrouche et al. (2017), and Meziani et al. (2018).

The previous studies generate different instances in order to evaluate their proposed methods, though, none of them are published. Instead, the instance generation scheme is designed and discussed in those studies, which are not useful for comparing various algorithms and solution methods, because even the same instance generation scheme may lead to different values, e.g., (a_j, L_j, b_j), so different instances. To fulfill the limitation, we propose a complete set of benchmark instances for the CTSP, which is available online in the Mendeley data repository at `http://dx.doi.org/10.17632/dd7ht5k5pn.1`.

3.1.2 The single-machine CTSP

We consider the set $\{5, 10, 15, 20, 25, 40, 50, 100\}$ for the number of jobs. We investigate jobs with small, medium and large durations, where the values of (a_j, L_j, b_j) are randomly generated from the discrete uniform distribution in the ranges as follows:

- Small jobs: $a_j, b_j \sim U(1, 20), L_j \sim U(10, 80)$
- Medium jobs: $a_j, b_j \sim U(1, 50), L_j \sim U(25, 200)$
- Large jobs: $a_j, b_j \sim U(1, 100), L_j \sim U(50, 400)$

Having eight values for the number of jobs and three alternatives for their sizes results in a total of 24 combinations. Generating ten instances for each combination will result in a set with 240 instances, i.e., per each number of jobs n, 30 instances are generated, where every ten instances have the same characteristics as discussed above.

We label this set the "general set". For illustration purposes, we show the name of an instance in the format `n-x-y-gen`, where `n` denotes the number of jobs, `x` denotes the instance ID (number), ranging from 1 to 10, and `y` denotes the sizes of the jobs that can be `S`, `M`, or `L`, denoting small, medium, or large jobs' attributes as discussed above. The `gen` denotes the general set. For example, $n = 5$ and small jobs result in a `5-1-S-gen` instance with the following job attributes

$$N_1 = (5, 30, 14), N_2 = (17, 10, 9), N_3 = (15, 52, 6), N_4 = (7, 33, 18), N_5 = (19, 41, 8).$$

Although there is no prior research considering the single-machine CTSP with a due date related criterion, we generate a set of due dates to complete the proposed data set. Specifically, we follow the approach proposed by Potts and Van Wassenhove (1982), which is also used in Fondrevelle et al. (2009) and Hamdi and Loukil (2017). The method is to generate the due dates in the interval $[Px, Py]$, where P is a lower bound for the problem. We use the bound $P = P_a + P_b$, and set $x = 0.05$ and $y = 0.95$.

We also generate the second set, i.e., the "restricted set" based on the identical problem. The set consists of 240 instances, where all the jobs of a particular instance are identical, i.e., $(a, L, b) \sim U(\alpha, \beta)$ with the same characteristics and values of α and β as those of the general set. Our naming convention for this set is `n-x-y-res`, where `n`, `x`, and `y` are as before, and `res` denotes the restricted set. For example, `5-1-S-res` may represent an instance with 5 small jobs as follows:

$$N_1 = N_2 = N_3 = N_4 = N_5 = (16, 47, 9).$$

Note that having the two sets allows us to generate instances for every special case, as well as every restriction on the task/delay durations. As an example, an instance with five small jobs for the case (a, L_j, b_j), where all the initial tasks taking an identical processing time can be generated by using the first values from the restricted set, and the second and third values from the general set. This results in

$$N_1 = (16, 30, 14), N_2 = (16, 10, 9), N_3 = (16, 52, 6), N_4 = (16, 33, 18), N_5 = (16, 41, 8).$$

We use `n-x-y-a` to denote this special case, where `a` stands for identical initial tasks. Benchmark instances for the other special cases can also be generated in the same way. For example, the special case $(p, L_j, p), p > 0$, where the initial and completion tasks of every job are equal to p, can be generated from the instances for (a, L_j, b) and setting $b_j = a, \forall j \in N$. The special case $a_j \geq b_j$ can be generated from the general set by interchanging the processing times of the tasks of job j with $a_j < b_j$. Similarly, instances for the problem with UET tasks, i.e., $(1, L_j, 1)$, can be generated by using the general set instances and setting all the task processing times to 1. Table 3.1 shows the naming convention for the instances generated for those special cases. In total, we generate 11 sets, each with 240 instances, i.e., 2,640 problem instances for the single-machine CTSP.

3.1.3 The shop CTSP

In order to generate benchmark instances for the two-machine flow-shop CTSP, we use the instance generation parameters of the single-machine problem (see Section 3.1.2). Particularly, the completion task durations can be considered as the processing times of the tasks on the second machine. Hence, all the sets discussed in Section 3.1.2 can easily be used for the two-machine flow-shop CTSP.

Table 3.1 : Naming convention for instances of the single-machine CTSP.

Type	Description
`gen`	general set
`res`	restricted set (a, L, b)
`a`	restricted set (a, L_j, b_j)
`L`	restricted set (a_j, L, b_j)
`b`	restricted set (a_j, L_j, b)
`aL`	restricted set (a, L, b_j)
`ab`	restricted set (a, L_j, b)
`Lb`	restricted set (a_j, L, b)
`p`	restricted set (p, L_j, p)
`a>b`	restricted set (a_j, L_j, b_j), $a_j \geq b_j$
`UET`	restricted set $(1, L_j, 1)$

However, for the m-machine flow-shop scheduling problem the numbers of jobs n and machines m need to be specified. Most of previous studies studied problems with up to five machines. An exception is Hamdi and Loukil (2017), who solved instances with up to ten machines. In addition, the number of jobs is set to at most 20 in the previous studies. To cover a broader range of instances, we set $n = \{10, 20, 50, 100\}$ and $m = \{5, 10, 20\}$, which results in 12 combinations. For each combination, ten instances are generated, so a total of 120 instances are produced. The tasks' processing times and delay durations are randomly taken from a discrete uniform distribution with the following parameters: $p_{kj} \sim U(1, 100)$ and $L_{kj} \sim U(1, 20)$. Note that the values of the delay duration are not from a wide range as in the single-machine case. This is to reflect the true nature of delays in the flow-shop, which are mainly related to the transport of the tasks. We also show the name of an instance in the format `n-m-x`, where `n` denotes the number of jobs, `m` denotes the number of machines and `x` denotes the instance number, ranging from one to ten. Also, we generate due dates for the jobs following the same method discussed in the single-machine setting, where the lower bound follows Equation (2.3), and the same values are used for x and y.

We note that the same data set can be used for the open-shop CTSP, with the only change that we do not consider the pre-specified sequence for the tasks of the jobs. To utilise the proposed data set for the job-shop setting, however, we need to generate a sequence for the tasks of each job.

3.2 Mathematical models

In this section we discuss the mathematical programming formulations for the CTSP documented in the literature. We first discuss the models for the single-machine CTSP and then consider those for the flow-shop setting.

3.2.1 The single-machine models

Let N indexed by j, and $H = \{1, \ldots, 2n\}$ indexed by h, denote sets of jobs and tasks, respectively, where H_{2j-1} and H_{2j} represent the initial and completion tasks of job j.

Elshafei et al. (2004) proposed the first mathematical formulation for the single-machine CTSP. For this time-indexed model, they discretised the time horizon into a set $\Theta = \{1, \ldots, \mathcal{T}\}$ of unit time slots. The binary decision variable $x_{jt}, \forall j \in N, \forall t \in \Theta$, takes the value of 1 if job j starts its processing at time slot t, and 0 otherwise. They considered the objective function of minimising the makespan. Model S1 in the following is their formulation.

Model S1

$$z = \min C_{\max} \tag{3.1}$$

subject to

$$C_{\max} \geq \sum_{t=1}^{\tau_j} (t + P_j - 1) x_{jt}, \quad j \in N, \tag{3.2}$$

$$\sum_{t=1}^{\tau_j} x_{jt} = 1, \quad j \in N, \tag{3.3}$$

$$\sum_{j \in N} \sum_{\substack{v \in s_{2j-1} \\ 1 \leq v \leq \tau_j}} x_{jv} + \sum_{j \in N} \sum_{\substack{v \in s_{2j} \\ 1 \leq v \leq \tau_j}} x_{jv} \leq 1, \quad t \in \Theta, \tag{3.4}$$

$$x_{jt} \in \{0, 1\}, \quad j \in N, \quad t \in \Theta. \tag{3.5}$$

The objective function (3.1) minimises the makespan, where the makespan is enforced to be at least as large as the finishing time of every job (constraints (3.2)). Here, $\tau_j = \mathcal{T} - P_j + 1$ is the latest time that if a_j is scheduled by τ_j, processing of b_j will finish by time $\mathcal{T}$ (end of the time horizon), where $P_j = a_j + L_j + b_j$ is the total time that job j remains in the system after its initial task starts processing. Constraints (3.3) ensure that every job starts at exactly one time slot, and constraints (3.4) indicate that at most one job can occupy each time slot. In other words, time slot t is occupied by a_j, if the starting time of job j is in $s_{2j-1} \in \{t - a_j + 1, \ldots, t\}$, or it is occupied by b_j, if the starting time of job j is in $s_{2j} \in \{t - P_j + 1, \ldots, t - a_j - L_j\}$. Note that the value of v must always remain between 1 and τ_j and any other value is ignored.

Elshafei et al. (2004) also proposed the weighted version of model S1, in which the objective is to maximise the total weight of the completed jobs within a time limit $\mathcal{T}_{\max}$ of the time horizon, where $\Theta' = \{1, \ldots, \mathcal{T}_{\max}\}$. Model W1 shows their formulation.

Model W1

$$z = \max \sum_{j \in N} \sum_{t=1}^{\tau_j} w_j x_{jt} \tag{3.6}$$

subject to

$$\sum_{t=1}^{\tau_j} x_{jt} \leq 1, \quad j \in N, \tag{3.7}$$

$$\sum_{j \in N} \sum_{\substack{v \in s_{2j-1} \\ 1 \leq v \leq \tau_j}} x_{jv} + \sum_{j \in N} \sum_{\substack{v \in s_{2j} \\ 1 \leq v \leq \tau_j}} x_{jv} \leq 1, \quad t \in \Theta', \tag{3.8}$$

$$x_{jt} \in \{0, 1\}, \quad j \in N, \quad t \in \Theta'. \tag{3.9}$$

The objective function (3.6) maximises the total weight of the completed jobs, where w_j is the weight of job j. Recall that this model does not require all the jobs to be scheduled, so constraints (3.7) reflect this. In other words, the constraints ensure that a job should be started at exactly one time slot t, if it is scheduled within the time limit $\mathcal{T}_{\max}$.

The numbers of decision variables and constraints in the time-indexed formulations, i.e., models S1 and W1, depend on the number of time slots. Hence, it is very likely that it grows quickly as the size of the problem increases. Sherali and Smith (2005) attempted to overcome this issue by proposing alternative formulations with continuous time horizon. They considered five possible relative configurations for every pair of jobs j and j':

1. processing of job j is finished before job j' starts;
2. processing of job j' is finished before job j starts;
3. the initial task of job j', i.e., $a_{j'}$, is interleaved between the two tasks of job j (this setting is only possible if $a_{j'} \leq L_j$ and $b_j \leq L_{j'}$);
4. the initial task of job j, i.e., a_j, is interleaved between the two tasks of job j' (this setting is only possible if $a_j \leq L_{j'}$ and $b_{j'} \leq L_j$); and,
5. job j is nested between the two tasks of job j' (if $L_{j'} \geq P_j$), or job j' is nested between the two tasks of job j (if $L_j \geq P_{j'}$; note that at most one of the two cases is possible).

Those five configurations are depicted in Figure 3.1. Configurations 1 and 2 are always possible if the time horizon is relatively large, i.e., if $\mathcal{T} \geq P_j + P_{j'}, \forall (j, j') \in N$ ($P_j = a_j + L_j + b_j$). According to those five configurations, Sherali and Smith (2005) defined a binary decision variable $y_{jj'\nu}$, which takes 1 if jobs j and $j', j < j'$, are scheduled relative to each other based on configuration $\nu = 1, \ldots, 5$. For a configuration ν and jobs j and j', $l_{jj'\nu}$ and $r_{jj'\nu}$ denote two constants representing the earliest and latest times that job j' can start its processing relative to job j, i.e., $l_{jj'\nu} \leq s_{j'} - s_j \leq r_{jj'\nu}$, where s_j is the starting time of job j.

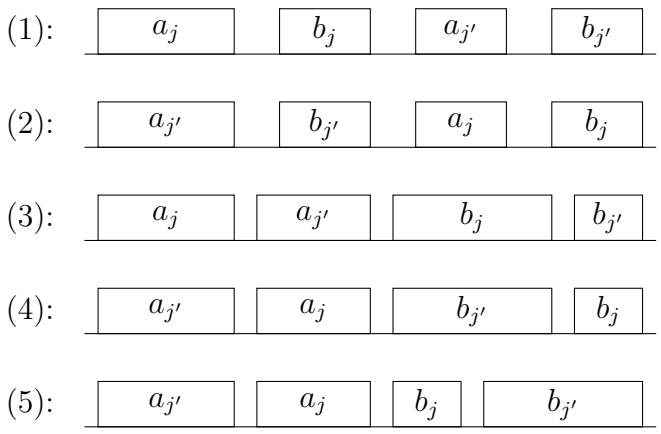

Figure 3.1 : Interleaving jobs j and j' (a) and nesting jobs j and j' (b).

The following example illustrates the calculation of those constants. Consider job j with $(a_j = 1, L_j = 4, b_j = 3)$ and job j' with $(a_{j'} = 1, L_{j'} = 10, b_{j'} = 2)$. If the time horizon is equal to 40 time units, configurations 1 and 2 are both possible and we have $l_{jj'1} = 8, r_{jj'1} = 27, l_{jj'2} = -32$ and $r_{jj'2} = -13$. Configuration 3 is possible as $a_{j'} \leq L_j$ and $b_j \leq L_{j'}$, so we have $l_{jj'3} = 1$ and $r_{jj'3} = 4$. Configuration 4 is possible as $a_j \leq L_{j'}$ and $b_{j'} \leq L_j$, so $l_{jj'4} = -10$ and $r_{jj'4} = -8$. Figure 3.2 shows the relative positions of jobs j and j' under configuration 4. Lastly, configuration 5 is possible for the nesting of job j between the two tasks of job j', so we have $l_{jj'5} = -3$ and $r_{jj'5} = -1$.

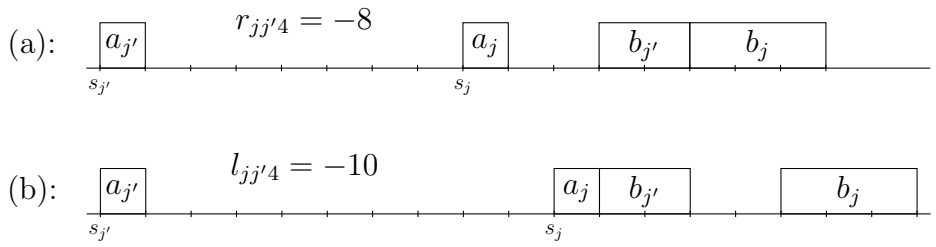

Figure 3.2 : Calculation of (a) $r_{jj'4}$ and (b) $l_{jj'4}$ for two jobs j $(1, 4, 3)$ and j' $(1, 10, 2)$.

Following the above definitions, Sherali and Smith (2005) formulated the problem to minimise the makespan into model S2 as follows:

Model S2

$$z = \min C_{\max} \tag{3.10}$$

subject to

$$C_{\max} \geq s_j + P_j, \quad j \in N, \tag{3.11}$$

$$\sum_{\nu=1}^{5} y_{jj'\nu} = 1, \quad (j,j') \in N, j < j', \tag{3.12}$$

$$s_{j'} - s_j \geq \sum_{\nu=1}^{5} l_{jj'\nu} y_{jj'\nu}, \quad (j,j') \in N, j < j', \tag{3.13}$$

$$s_{j'} - s_j \leq \sum_{\nu=1}^{5} r_{jj'\nu} y_{jj'\nu}, \quad (j,j') \in N, j < j', \tag{3.14}$$

$$s_j \geq 0, \quad j \in N, \tag{3.15}$$

$$y_{jj'\nu} \in \{0,1\}, \quad (j,j') \in N, j < j', \quad 1 \leq \nu \leq 5. \tag{3.16}$$

The objective function (3.10) minimises the makespan, where the makespan is enforced to be at least as large as the finishing time of every job (constraints (3.11)). Constraints (3.12) ensure that exactly one configuration is selected for every pair of jobs. Constraints (3.13) and (3.14) ensure that the difference between the starting time of every pair of jobs is within the permissible upper and lower bounds, under their selected configuration. Note that if there is a configuration ν that is not possible for two jobs j and j', the relative decision variable $y_{jj'\nu}$ is set to zero.

Sherali and Smith (2005) also proposed the weighted version of model S2. They defined a binary decision variable x_j that is equal to 1 if both tasks of job j are processed before $\mathcal{T}_{\max}$. The weighted formulation maximises the total weight of the completed jobs within the time limit $\mathcal{T}_{\max}$. Model W2 is as follows:

Model W2

$$z = \max \sum_{j \in N} w_j x_j \tag{3.17}$$

subject to

$$s_j + P_j \leq UB - x_j(UB - \mathcal{T}_{\max}), \quad j \in N, \tag{3.18}$$

(3.12) to (3.16),

$$x_j \in \{0,1\}, \quad j \in N, \tag{3.19}$$

where UB is an upper bound for the problem, which can be set to $\sum_{j \in N} P_j$. Constraints (3.18) state that if job j is selected to be scheduled, it must finish before the time limit. In model W2, the values of $l_{jj'\nu}$ and $r_{jj'\nu}$ are the same as those in model S2.

Békési et al. (2014) proposed two linear ordering-based formulations of the problem to minimise the makespan. In these formulations, a sequence is an ordered set of tasks, rather than jobs. For any pair of tasks h and h', a binary variable $x_{hh'}$ is defined, which

takes the value of 1 if task h' is started after task h in the sequence. Model S3 shows their first formulation.

Model S3

$$z = \min C_{\max} \tag{3.20}$$

subject to

$$C_{\max} \geq s_{2j} + b_j, \quad j \in N, \tag{3.21}$$

$$x_{2j-1,2j} = 1, \quad j \in N, \tag{3.22}$$

$$x_{hh'} + x_{h'h} = 1, \quad (h, h') \in H, h < h', \tag{3.23}$$

$$x_{hh'} + x_{h'h''} + x_{h''h} \leq 2, \quad (h, h', h'') \in H, h \neq h', h \neq h'', h' \neq h'', \tag{3.24}$$

$$s_{2j} = s_{2j-1} + a_j + L_j, \quad j \in N, \tag{3.25}$$

$$s_{2j} \leq UB - b_j, \quad j \in N, \tag{3.26}$$

$$s_{h'} \geq s_h + p_h - UB(1 - x_{hh'}), \quad (h, h') \in H, \quad h \neq h', \tag{3.27}$$

$$s_h \geq 0, \quad h \in H, \tag{3.28}$$

$$x_{hh'} \in \{0, 1\}, \quad (h, h') \in H, \quad h \neq h', \tag{3.29}$$

where s_{2j} is the starting time of the completion task of job j. Constraints (3.22) indicate that the initial task of each job should be scheduled before its completion task. The relative order of any pair of tasks is considered by constraints (3.23). Constraints (3.24) are the so-called "3-dicycle inequalities" for any triple distinct tasks. The relation between the starting times of the tasks of a job is defined by constraints (3.25), where an upper bound on the starting times of the completion tasks is set by constraints (3.26). Constraints (3.27) relate the starting time variables to the linear ordering variables, where p_h is the processing time of task h. Specifically, the constraints ensure that if task h' is scheduled after task h, i.e., $x_{hh'} = 1$ and the constraint is as $s_{h'} \geq s_h + p_h$, the starting time of task h', i.e., $s_{h'}$, must be at least as large as the finishing time of task h, i.e., $s_h + p_h$. On the other hand, in case $x_{hh'} = 0$, the constraints turn to $s_{h'} \geq s_h + p_h - UB$, that is always true. The reason is that the finishing time of any task, including task h, cannot be larger than UB.

Békési et al. (2014) proposed a second formulation as an alternative for model S3.

The model does not use the starting time variables. The model includes variables y_h, which are the idle times of the machine after processing task h, and positive constants C_h, which are upper bounds on the values of y_h. A knapsack constraint is used in order to express that the sum of the processing times of tasks that are executed between the two tasks of job j must not exceed its delay duration L_j. Because such a constraint is initially stated in a non-linear form, additional variables $y_{jh}, j \in N, h \in H \setminus \{2j-1, 2j\}$ are added for linearisation (see Békési et al. (2014) for more details on the non-linear constraint). Model S4 is the formulation as follows:

Model S4

$$z = \min \sum_{j \in N} (y_{2j-1} + y_{2j}) \tag{3.30}$$

subject to

$$y_{2j-1} + \sum_{h \notin \{2j-1, 2j\}} (p_h(x_{h,2j} - x_{h,2j-1}) + y_{jh}) = L_j, \quad j \in N, \tag{3.31}$$

$$y_{jh} \leq y_h, \quad j \in N, \quad h \in H \setminus \{2j-1, 2j\}, \tag{3.32}$$

$$y_{jh} \leq C_{jh}(x_{h,2j} - x_{h,2j-1}), \quad j \in N, \quad h \in H \setminus \{2j-1, 2j\}, \tag{3.33}$$

$$y_{jh} \geq y_h - C_j(x_{2j,h} + x_{h,2j-1}), \quad j \in N, \quad h \in H \setminus \{2j-1, 2j\}, \tag{3.34}$$

(3.22) to (3.24), (3.29),

$$y_j \geq 0, \quad j \in N, \tag{3.35}$$

$$y_{jh} \geq 0, \quad j \in N, \quad h \in H, \quad h \notin \{2j-1, 2j\}. \tag{3.36}$$

The objective function (3.30) minimises the total idle time of the machine, which is equivalent to minimising the makespan. The knapsack constraints are presented in (3.31), indicating that the delay duration of each job consists of the tasks that are processed between its initial and completion tasks, plus the idle time between them. Constraints (3.32) to (3.34) are used to linearise the knapsack constraints. Here, the constants $C_{jh} = \max\{0, L_j - p_h\}$ are upper bounds on the values of y_{jh}. In addition, $C_h = L_j$, if h is the first task of job j, or $C_h = \max_{j \in N}\{C_{jh} | h \notin \{2j-1, 2j\}\}$, if h is the second task of a job.

3.2.2 The flow-shop models

Hamdi and Loukil (2017) proposed three mathematical formulations for the (permutation) flow-shop CTSP. The objective function of the models is to minimise the total earliness and tardiness. The first model, which we denote by F1, is a completion time-

based formulation. It uses the decision variable x_{ij}, which takes the value of 1 if job j is assigned to the sequence position $i \in N$, and 0 otherwise. Also, the non-negative variable C_{ik} denotes the completion time of job in the sequence position i on machine k. Model F1 shows this formulation as follows:

Model F1

$$z = \min \sum_{i \in N} (E_i + T_i) \tag{3.37}$$

subject to

$$\sum_{i \in N} x_{ij} = 1, \quad j \in N, \tag{3.38}$$

$$\sum_{j \in N} x_{ij} = 1, \quad i \in N, \tag{3.39}$$

$$T_i \geq C_{im} - \sum_{j \in N} d_j x_{ij}, \quad i \in N, \tag{3.40}$$

$$E_i \geq \sum_{j \in N} d_j x_{ij} - C_{im}, \quad i \in N, \tag{3.41}$$

$$C_{i,k+1} = C_{ik} + \sum_{j \in N} (p_{j,k+1} + L_{jk}) x_{ij}, \quad i \in N, \quad k \in M \setminus \{m\}, \tag{3.42}$$

$$C_{i+1,k} \geq C_{ik} + \sum_{j \in N} p_{jk} x_{i+1,j}, \quad i \in N \setminus \{n\}, \quad k \in M, \tag{3.43}$$

$$C_{ik} \geq 0, \quad i \in N, \quad k \in M, \tag{3.44}$$

$$x_{ij} \in \{0, 1\}, \quad (i, j) \in N, \tag{3.45}$$

$$E_i, T_i \geq 0, \quad i \in N. \tag{3.46}$$

The objective function (3.37) minimises the total earliness and tardiness. The assignment constraints (3.38) and (3.39) ensure that each job is assigned to exactly one sequence position and each sequence position is assigned to exactly one job. Constraints (3.40) and (3.41) define the tardiness and earliness of the job in the sequence position i. Constraints (3.42) define the completion time of each job on consecutive machines. Likewise, constraints (3.43) calculate the completion times of any pair of consecutive jobs on every machine. We notice that Hamdi and Loukil (2017) missed a constraint in model F1, i.e., they did not explicitly define the completion time of the first job on the first machine. In particular, the omission of such a constraint may result in starting the first job on the first machine before the time horizon, i.e., an infeasible solution. To avoid this, we add the following constraint to model F1.

$$C_{11} \geq \sum_{j \in N} p_{j1} x_{1j}. \tag{3.47}$$

We express constraint (3.47) in the form of greater than or equal to because the objective function of model F3, i.e., the total earliness and tardiness is a non-regular one, and forced idle times may therefore improve the objective. In case of a regular objective function, e.g., minimisation of the makespan, forced idle times will not be beneficial, so it suffices to express constraint (3.47) as $C_{11} = \sum_{j \in N} p_{j1} x_{1j}$.

Their second model is an idle time-based formulation, in which the variable I_{ik} is the idle time of machine k after processing the job in the sequence position i. We notice that their formulation does not generate a valid solution. We investigate their formulation and observe two flaws. First, the calculation of jobs' earliness needs fixing. Second, by defining variable I_{ik} as the idle time of machine k "after" processing the job in the sequence position i, there is no possibility to have idle time before the processing of the first job on the first machine. Note that because the objective function is non-regular, it might be beneficial to have idle time before the processing of the first job on the first machine. For example, in a one-job two-machine instance with $a = 2$, $L = 3$, $b = 2$ and $d = 9$, the job should start at time 2 to minimise the total earliness and tardiness (see Figure 3.3).

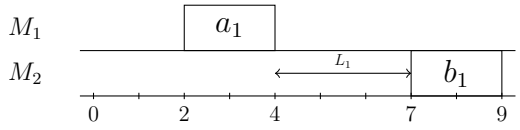

Figure 3.3 : An example to clarify the correct definition of idle times.

Consequently, we define variable I_{ik} as the idle time of machine k "before" processing the job in the sequence position i. We also correct the earliness calculation. The corrected formulation, denoted as model F2, is as follows:

Model F2

(3.37) to (3.39),

$$\sum_{j \in N} (p_{jk} + L_{jk}) x_{1j} + I_{1k} = I_{1,k+1}, \quad k \in M \setminus \{m\}, \tag{3.48}$$

$$\begin{aligned} &\sum_{j \in N} (p_{j,k+1} + L_{jk}) x_{ij} + I_{i+1,k+1} \\ &\quad = \sum_{j \in N} (p_{jk} + L_{jk}) x_{i+1,j} + I_{i+1,k}, \quad i \in N \setminus \{n\}, \quad k \in M \setminus \{m\}, \end{aligned} \tag{3.49}$$

$$T_i \geq \sum_{s=1}^{i} \sum_{j \in N} p_{jm} x_{sj} + \sum_{s=1}^{i} I_{sm} - \sum_{j \in N} d_j x_{ij}, \quad i \in N, \tag{3.50}$$

$$E_i \geq \sum_{j \in N} d_j x_{ij} - \sum_{s=1}^{i} \sum_{j \in N} p_{jm} x_{sj} - \sum_{s=1}^{i} I_{sm}, \quad i \in N, \tag{3.51}$$

$$I_{ik} \geq 0, \quad i \in N, \quad k \in M, \tag{3.52}$$

(3.45) and (3.46).

Constraints (3.48) model the idle time before the job in the first position. Similarly, constraints (3.49) define the idle time between the remaining jobs on all the machines. In particular, they relate the processing times, delays and idle times of every two consecutive jobs on any pair of consecutive machines. Constraints (3.50) and (3.51) define the tardiness and earliness, respectively.

The third model proposed by Hamdi and Loukil (2017) is a starting time-based formulation, which uses the variable s_{ik} to express the starting time of the job in position i on machine k. Model F3 in the following shows this formulation.

Model F3

(3.37) to (3.39),

$$T_i \geq s_{im} + \sum_{j \in N} p_{jm} x_{ij} - \sum_{j \in N} d_j x_{ij}, \quad i \in N, \tag{3.53}$$

$$E_i \geq \sum_{j \in N} d_j x_{ij} - s_{im} - \sum_{j \in N} p_{jm} x_{ij}, \quad i \in N, \tag{3.54}$$

$$s_{i,k+1} = s_{ik} + \sum_{j \in N} (p_{jk} + L_{jk}) x_{ij}, \quad i \in N, \quad k \in M \setminus \{m\}, \tag{3.55}$$

$$s_{i+1,k} \geq s_{ik} + \sum_{j \in N} p_{jk} x_{ij}, \quad i \in N \setminus \{n\}, \quad k \in M, \tag{3.56}$$

$$s_{ik} \geq 0, \quad i \in N, \quad k \in M, \tag{3.57}$$

(3.45) and (3.46).

Constraints (3.53) and (3.54) define the tardiness and earliness of jobs. Constraints (3.55) relate the starting times of a job on every pair of consecutive machines. Finally, constraints (3.56) relate the starting times of any pair of consecutive jobs on a machine. Their original model also includes the constraint $s_{11} \geq 0$. This constraint is redundant because it has already been implied by constraints (3.57). Therefore, we do not include it in model F3.

Also, we provide an additional formulation for the flow-shop CTSP, which Arabameri and Salmasi (2013) originally proposed for the no-wait flow-shop scheduling problem. We choose this model because it is relatively new and uses the sequential ordering variables (the first three have positional variables). The model proposed by Arabameri and

Salmasi (2013) includes a binary variable $x_{jj'}$, which takes 1 if job j' is placed immediately after job j in a sequence, and 0 otherwise. Two dummy jobs are considered, where their processing times and delays on all the machines are equal to zero. The due date of the first dummy job is also set to zero so as to assign it to the first position in a sequence, and that of the second dummy job is set to a large positive number in order to assign it to the last position in the sequence. Assuming the set $N' = \{1, \ldots, n+2\}$ of jobs, model F4 is as follows:

Model F4

$$z = \min \sum_{j \in N' \setminus \{n+2\}} (E_j + T_j) \tag{3.58}$$

subject to

$$\sum_{\substack{j \in N' \setminus \{n+2\} \\ j \neq j'}} x_{jj'} = 1, \quad j' \in N' \setminus \{1\}, \tag{3.59}$$

$$\sum_{\substack{j' \in N' \setminus \{1\} \\ j' \neq j}} x_{jj'} = 1, \quad j \in N' \setminus \{n+2\}, \tag{3.60}$$

$$C_{j'k} + \mathcal{M}(1 - x_{jj'}) \geq C_{jk} + p_{jk}, \quad (j, j') \in N', \quad j \neq j', \quad k \in M, \tag{3.61}$$

$$C_{jk} = C_{j,k-1} + p_{jk} + L_{j,k-1}, \quad j \in N', \quad k \in M \setminus \{1\}, \tag{3.62}$$

$$T_j \geq C_{jm} - d_j, \quad j \in N', \tag{3.63}$$

$$E_j \geq d_j - C_{jm}, \quad j \in N', \tag{3.64}$$

$$C_{jk} \geq 0, \quad j \in N', \quad k \in M, \tag{3.65}$$

$$x_{jj'} \in \{0, 1\}, \quad (j, j') \in N', \quad j \neq j', \tag{3.66}$$

$$E_j, T_j \geq 0, \quad j \in N'. \tag{3.67}$$

The objective function (3.58) minimises the total earliness and tardiness of jobs 1 to $n+1$. The last job is not included as it is the second dummy job. Constraints (3.59) and (3.60) ensure that all the jobs are sequenced exactly once. Constraints (3.61) and (3.62) relate the completion time variables to the sequential ordering variables. Specifically, they set the completion times of every two consecutive jobs on every machine, and every job on any pair of consecutive machines. Here C_{jk} is the completion time of job j on machine k and $\mathcal{M}$ is a sufficiently large constant. Constraints (3.63) and (3.64) define the tardiness

and earliness of the jobs.

3.3 Performance evaluation of models

In this section we evaluate the performance of all ten models discussed in Section 3.2. For this purpose, we carried out comprehensive computational experiments. Recall that the mathematical models can be categorised into three groups: (1) single-machine models to minimise the makespan, i.e., models S1 to S4; (2) single-machine models to maximise the weighted sum of the completed jobs, i.e., models W1 and W2; and (3) flow-shop models to minimise the total earliness and tardiness, i.e., models F1 to F4. Table 3.2 summarises the number of decision variables and constraints of each model. In the table, n and m denote the numbers of jobs and machines, respectively, and $\mathcal{T}$ denotes the number of discretised time units.

Table 3.2 : Numbers of decision variables and constraints in the models.

Model	Number of binary variables	Number of continuous variables	Number of constraints
S1	$n\mathcal{T}$	1	$2n + \mathcal{T}$
S2	$(5n^2 - 5n)/2$	$n + 1$	$(3n^2 - n)/2$
S3	$4n^2 - 2n$	$n + 1$	$\frac{8}{3}n^3 + 2n^2 + \frac{7}{3}n$
S4	$4n^2 - 2n$	$2n^2$	$\frac{8}{3}n^3 + 4n^2 - \frac{11}{3}n$
W1	$n\mathcal{T}$	-	$n + \mathcal{T}_{\max}$
W2	$(5n^2 - 3n)/2$	n	$(3n^2 - n)/2$
F1	n^2	$nm + 2n$	$2nm + 3n–m + 1$
F2	n^2	$nm + 2n$	$nm + 3n$
F3	n^2	$nm + 2n$	$2nm + 3n–m + 1$
F4	$(n + 2)^2 - n - 2$	$(n + 2)m + 2(n + 2)$	$(n+2)^2m+3(n+2)-2$

For the first and second groups, which are models for the single-machine CTSP, we tested 240 instances of general set (see Section 3.1.2). For the weighted models, we randomly generated the job weights from $\sim U(1, 10)$. We set the parameter $\mathcal{T}_{\max}$ to $\frac{1}{3}\sum_{j\in N} P_j$, where $P_j = a_j + L_j + b_j$. For the third group, i.e., the flow-shop CTSP models, we studied 120 benchmark instances generated in Section 3.1.3. We used the solver Gurobi version 8.0.0 (Gurobi Optimization, 2018) to solve the models. We coded all the models in Python version 2.7. We performed all the computational experiments on a PC with an Intel® Core™ i5-7500 CPU clocked at 3.40GHz with 8GB of memory under the Linux Ubuntu 16.04 operating system. The machine has four threads and we ran Gurobi in the parallel mode, i.e., we used all the four threads. We only changed one Gurobi parameter, i.e., the time limit, for which we changed its default value to 900 seconds. Therefore, this was the stopping criterion for the solver. For the remaining parameters of Gurobi, we used their default values.

Table 3.3 : Comparison of the performance of the studied mathematical models.

Model	Feas	Best	Opt	Gap (%)	Time (sec)
S1	**210**	73	34	30.9	788.2
S2	126	85	**61**	4.2	**674.8**
S3	192	**148**	60	**3.3**	677.3
S4	166	68	58	15.5	751.7
W1	**240**	**199**	102	**0.4**	578.4
W2	205	153	**113**	1.1	**497.5**
F1	106	32	30	5.5	675.5
F2	107	47	**36**	5.1	**645.0**
F3	109	32	30	5.6	675.5
F4	**120**	**106**	30	**0.6**	742.7

Table 3.3 summarises the results of the computational experiments. We use five criteria to evaluate each model. These include "feas", which shows the number of generated feasible solutions by the model within the time limit; "best", which is the number of best solutions obtained by the model; "opt", i.e., the number of optimal solutions delivered by the model; and "gap (in %)", which is calculated as $\frac{z-z^*}{z^*} \times 100$, where z is the objective function value obtained by the model and z^* is the best objective function value among all the models, i.e., the best available solution, and is calculated over the number of feasible solutions for the model. The last metric "Time (sec)" represents the average computing time in seconds for the model. We highlight the criterion with the best value across the models (per group).

Among the single-machine models to minimise the makespan, model S1 generates the largest number of feasible solutions. In addition, as Table 3.5 shows, model S1 is the only model that produces feasible solutions for instances with 100 jobs. Despite this, model S1 is not the outperforming model in terms of solution quality because it only delivers the best solution for 73 (almost 30%) and optimal for 34 (less than 15%) of instances. The model's gap (from the best solution) is also the highest among all the models. Notice that a larger gap value implies that the solution is further from the best available one.

Model S2 delivers feasible solutions for slightly more than half of instances (52.5%). Nevertheless, it obtains the highest number of optimal solutions. Also, its gap is the second best, reflecting the quality of its solutions. Table 3.5 details the results and shows that the majority of high-quality solutions obtained by model S2 are for instances with up to 20 jobs. Model S2 was originally proposed to overcome the exponential growth in the numbers of variables and constraints in the time-indexed model. However, it is evident that it is unable to deliver feasible solutions for instances with 40 jobs and more.

Model S3 obtains the largest number of best solutions. It also delivers optimal solutions for a quarter of the instances. The quality of the delivered solutions is very promising because the gap is lowest among all the models. The model generates feasible solutions

for 80% of the instances. The details presented in Table 3.5 indicate that this model has the capability of models S1 and S2 since it generates quality solutions for a broad range of the instances. Model S4, which is an alternative for model S3, under-performs model S3 across all the metrics.

The average computation times reported in Table 3.5 reveal that models S2 and S3 are able to solve slightly larger instances in a short time, because their computational times do not quickly rise as for models S1 and S4.

Among the weighted models, model W1 generates feasible solutions for all the instances, while model W2 produces feasible solutions for 205 (almost 86%) instances. Model W1's better performance is further recognised by its larger number of best solutions and lower gap values. Particularly, the average gap for model W1 is less than half of that for model W2. Regarding the number of optimal solutions, however, model W2 obtains more optimal solutions, and in the order of 11 more instances. Based on the detailed results presented in Table 3.4, we draw the conclusion that model W2 is a better choice to solve the small instances, due to its shorter computing times and greater numbers of optimal solutions. On the contrary, model W1 is the best choice when dealing with large instances, due to its competitive gap and number of best solutions obtained.

Between the flow-shop models, model F4 performs significantly better than the other models. In particular, it delivers feasible solutions for all the instances, with 106 (nearly 88%) of which being the best available solutions. As a result, the gap value of this model is small, i.e., 0.6%. Models F1 and F3 behave very closely, which is not surprising since they are very similar. Model F2, i.e., the idle time-based model, is slightly better than models F1 and F3 because it has a lower average gap, as well as greater number of best and optimal solutions. Model F2 also reports the largest number of optimal solutions among all the models. The average computing times of models F1 to F3 are less than that of model F4; particularly, model F2 has the shortest average time. To conclude, the detailed results presented in Table 3.6 reveal that model F2 is the best option to solve instances with up to 10 jobs. However, model F4 is the top performer and the best choice to deal with large instances.

Table 3.4 : Detailed performance of the single-machine weighted models (a "-" denotes that the model cannot produce an outcome within the time limit).

n	W1					W2				
	Feas	Best	Opt	Gap (%)	Time (sec)	Feas	Best	Opt	Gap (%)	Time (sec)
5	30	30	30	0.0	1.5	30	30	30	0.0	0.0
10	30	23	20	0.8	363.8	30	30	30	0.0	6.1
15	30	25	20	0.6	390.7	30	30	22	0.0	274.7
20	30	19	12	0.8	589.2	30	27	14	0.1	517.0
25	30	21	8	0.6	690.1	30	19	9	1.1	651.3
40	30	24	6	0.5	750.5	30	12	5	3.0	802.6
50	30	27	6	0.3	785.1	25	5	3	4.2	828.3
100	30	30	0	0.0	1056.3	0	0	0	-	900.1

Table 3.5 : Detailed performance of the single-machine makespan models (a "-" denotes that the model cannot produce an outcome within the time limit).

n	S1					S2					S3					S4				
	Feas	Best	Opt	Gap (%)	Time (sec)	Feas	Best	Opt	Gap (%)	Time (sec)	Feas	Best	Opt	Gap (%)	Time (sec)	Feas	Best	Opt	Gap (%)	Time (sec)
5	30	30	30	0.0	5.0	30	30	30	0.0	0.0	30	30	30	0.0	0.0	30	30	30	0.0	0.2
10	30	5	4	4.7	843.5	30	30	30	0.0	15.0	30	30	30	0.0	17.8	30	30	28	0.0	596.8
15	30	0	0	12.1	900.1	30	10	1	1.0	883.1	30	21	0	0.2	900.0	30	0	0	7.5	900.0
20	30	1	0	28.6	900.1	27	8	0	8.9	900.0	30	22	0	0.4	900.1	30	0	0	13.4	900.0
25	30	7	0	60.2	900.1	9	7	0	28.8	900.0	30	16	0	1.8	900.2	25	0	0	48.8	900.0
40	30	11	0	70.6	956.9	0	0	0	-	900.0	26	15	0	21.3	900.0	16	4	0	42.2	900.0
50	20	9	0	60.1	900.1	0	0	0	-	900.0	16	14	0	0.4	900.1	5	4	0	10.8	900.0
100	10	10	0	0.0	900.1	0	0	0	-	900.0	0	0	0	-	900.6	0	0	0	-	916.8

Table 3.6 : Detailed performance of the flow-shop models.

n	F1					F2					F3					F4				
	Feas	Best	Opt	Gap (%)	Time (sec)	Feas	Best	Opt	Gap (%)	Time (sec)	Feas	Best	Opt	Gap (%)	Time (sec)	Feas	Best	Opt	Gap (%)	Time (sec)
10	30	30	30	0.0	1.8	30	30	30	0.0	1.3	30	30	30	0.0	1.6	30	30	30	0.0	270.6
20	30	1	0	1.4	900.0	30	7	6	0.9	778.8	30	2	0	1.2	900.0	30	27	0	0.0	900.0
50	26	0	0	10.4	900.1	30	0	0	13.7	900.0	29	0	0	9.2	900.1	30	30	0	0.0	900.0
100	20	1	0	13.7	900.1	17	10	0	6.2	900.0	20	0	0	15.2	900.1	30	19	0	2.3	900.1

Chapter 4

Coupled tasks on a single-machine

In this chapter we study the single-machine CTSP. First, in Section 4.1 we study a special case of the single-machine couple task problem where processing times are time-dependent. Then, we study the general problem, where a binary search matheuristic algorithm is proposed in Section 4.2, the best available mathematical model of the problem is improved in Section 4.3, and a second matheuristic algorithm, i.e., a relax-and-solve algorithm, is proposed in Section 4.4. Section 4.5 ends this chapter where we evaluate the new mathematical formulation and compare the performance of the proposed methods based on the new formulation. The results presented in Section 4.1 and Section 4.2 are published in:

- Khatami, M. and Salehipour, A. (2021a). "A binary search algorithm for the general coupled task scheduling problem". *4OR* 19(4), 593–611.
- Khatami, M. and Salehipour, A. (2021b). "Coupled task scheduling with time-dependent processing times". *Journal of Scheduling* 24, 223–236.

In addition, the results presented in Section 4.3 are submitted for possible publication as:

- Khatami, M. and Salehipour, A. (2021c). "The coupled task scheduling problem: An improved mathematical program and a new solution algorithm". *Submitted to International Transactions in Operational Research*.

4.1 Time-dependent scheduling

The healthcare applications discussed in Section 2.2 become complex when staff tiredness, that typically occurs in practice and impacts the treatment times, is considered. In this section we propose solution methods for the single-machine CTSP where processing times are variables of their starting times, i.e., time-dependent processing times. Those variable processing times can address human fatigue within the CTSP. We also note that Mosheiov (1994) and Gawiejnowicz (2008) studied the time-dependent processing times to model the time of performing medical services as the time increases under deteriorating health conditions. It is true that in the coupled task applications in healthcare, particularly, in chemotherapy and radiotherapy services, the symptoms of the patient rapidly grow over time, i.e., the patient needs longer treatment if that treatment is started late.

Orman and Potts (1997) proved that the case (p, p, b_j) is polynomially solvable for the objective function of minimising the makespan. We investigate the same case, however, with a time-dependent processing time characteristic for the completion tasks. Under this setting, the processing time of the completion task depends on its starting time. Gupta and Gupta (1988) introduced the scheduling problems with time-dependent processing times. Under this setting, job j has a normal processing time $\alpha_j \geq 0$ and a processing rate (also called "deterioration rate") $\beta_j \geq 0$. The actual processing time of job j depends on its starting time s_j, and is typically shown as $p_j = \alpha_j \pm \beta_j s_j$. A variant of this model, which is called the "simple linear processing times" (also called "simple linear deterioration"), assumes $\alpha_j = 0$, and therefore, $p_j = \beta_j s_j$. We investigate the simple linear processing times model for the completion tasks. Therefore, with considering the three-field scheduling notation proposed by Graham et al. (1979), the present study investigates the problem $1|(p, p, b_j = \beta_j s_j)|C_{\max}$. In this chapter, we first present a mathematical formulation for the problem in Section 4.1.1. In Section 4.1.2, we discuss optimal properties for the problem. A dynamic program and a heuristic are also proposed. The results of numerical experiments are presented in Section 4.1.4.

4.1.1 Problem definition

Given a set of coupled task jobs $N = \{1, 2, \ldots, n\}$, each with two tasks and there is an exact delay period between two consecutive tasks, to be processed on a single-machine, a job $j \in N$ is represented by $(p, p, b_j = \beta_j s_j)$, where p is a positive integer, and $\beta_j, s_j > 0, \forall j \in N$. Therefore, parameter $b_j, \forall j$ is a time dependent variable defined by a simple linear processing time. The goal is to develop a schedule for $(p, p, b_j = \beta_j s_j)$, so to minimise the makespan.

As discussed in Section 3.3, model S3 is computationally among the top performing models for the CTSP. Hence, we extend that formulation for the problem of this study. Model S3-T below shows the formulation for $(p, p, b_j = \beta_j s_j)$, where $\beta_{s_{2j}}$ is substituted for b_j.

Model S3-T

$$z = \min C_{\max} \tag{4.1}$$

subject to

(3.22)-(3.25), (3.27)-(3.29),

$$s_h \geq s_{2j-1} + p - UB(1 - x_{2j-1,h}), \quad 1 \leq j \leq n, \quad 1 \leq h \leq 2n, \quad h \notin \{2j-1, 2j\}, \tag{4.2}$$

$$s_h \geq s_{2j} + \beta_j s_{2j} - UB(1 - x_{2j,h}), \quad 1 \leq j \leq n, \quad 1 \leq h \leq 2n, \quad h \notin \{2j-1, 2j\}, \tag{4.3}$$

$$C_{\max} \geq s_{2j} + \beta_j s_{2j}, \quad 1 \leq j \leq n, \tag{4.4}$$

$$s_{2j} \leq UB - \beta_j s_{2j}, \quad 1 \leq j \leq n. \tag{4.5}$$

The objective function (Equation (4.1)) minimises the makespan. Constraints (4.2) and (4.3) relate the starting time of tasks, and also relate the starting time variables to the linear ordering variables. Constraints (4.4) ensure that the makespan is larger than the completion time of any job. An upper bound (UB) is considered for the starting time of the completion tasks in constraints (4.5).

4.1.2 Minimising the makespan

We show that model S3-T can be easily solved for these two cases: (1) $\beta_j > 0.5, \forall j \in N$, and (2) a two-job instance. In addition, under the condition that jobs are grouped into a few classes we propose a dynamic program for model S3-T that delivers an optimal schedule in polynomial time. For the general case, we propose an efficient heuristic algorithm.

Optimal schedule

Orman and Potts (1997) showed that for problem (p, p, b_j) under general processing times, the nesting of jobs is not possible. For a pair of jobs j and k, if $b_j \leq p$, it is possible to interleave jobs j and k, where j is the first job and k is the second job of the pair. The contribution of this setting to the makespan is equal to $3p + b_k$, as illustrated in Figure 4.1a. On the other hand, any job j with $b_j > p$ contributes $2p + b_j$ to the makespan, as shown in Figure 4.1b. Therefore, an optimal schedule is derived when as many jobs as possible are interleaved. We will investigate whether this is the case in problem $(p, p, \beta_j s_j)$.

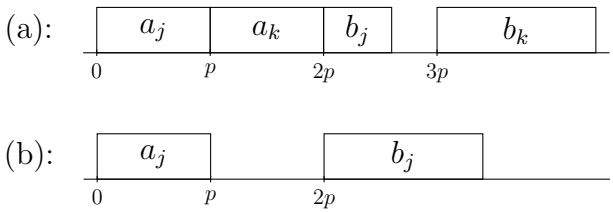

Figure 4.1 : Contribution of an interleaving pair of jobs (a), and a single job (b) to the makespan.

In the classical single-machine setting, an optimal schedule for both simple linear and linear time-dependent processing times exists. Under the simple linear condition Mosheiov (1994) showed that all schedules lead to the same makespan, which is equal to $s_1 \times \prod_j (1 + \beta_j), s_1 > 0$, where s_1 is the starting time of the schedule. Under the linear processing times, Gupta and Gupta (1988) proved that the optimal makespan is obtained when jobs are sequenced in non-decreasing order of α_j/β_j.

The results of Gupta and Gupta (1988) may be extended for problem $(p, p, \beta_j s_j)$. We note that the combination of the initial task and the delay period of job j can be considered

as the normal processing time of job j, i.e., $\alpha_j = p + p = 2p$. If no interleaving is possible, problem $(p, p, \beta_j s_j)$ reduces to the single-machine scheduling with linear time-dependent processing times, for which sequencing jobs in non-decreasing order of α_j/β_j leads to the optimal makespan. Therefore, it suffices to investigate if interleaving is possible.

We can assume that the first job starts at time 0 because all jobs are available at time 0. Then, the processing time of its completion task will be $b_j = \beta_j \times 2p$. Two cases are possible: (1) $\beta_j > 0.5, \forall j \in N$, and (2) $\beta_j \leq 0.5, \exists j \in N$. The following theorem leads to an optimal schedule if $\beta_j > 0.5, \forall j$.

Theorem 1. *If $\beta_j > 0.5, \forall j$, an optimal solution for model S3-T is obtained when jobs are sorted in non-increasing order of β_j.*

Proof. Without loss of generality let the first job start at time 0. Therefore, its completion task starts at time $2p$ and $b_j = \beta_j \times 2p$. Note that $b_j > p$, since $\beta_j > 0.5, \forall j$. Recall that there is no possibility for jobs interleaving if $b_j > p$. Hence, an optimal sequence is obtained by ordering jobs in non-decreasing order of α_j/β_j, or equivalently in non-increasing order of β_j since $\alpha_j = 2p > 0, \forall j \in N$. □

The proof of Theorem 1 shows that even though the actual processing time of jobs depends on the starting time of the completion tasks, this does not impact an optimal sequence because $\alpha_j = 2p, \forall j \in N$. The result of Theorem 1 may also be utilised to locate jobs that cannot be the first of an interleaving pair. This leads to the following lemma.

Lemma 1. *Under arbitrary values of β the jobs in set $\bar{J} \subset J$, where $\bar{J} = \{j | \beta_j > 0.5\}$ appear in an optimal schedule in non-increasing order of $\beta_j, j \in \bar{J}$.*

Proof. Let $\beta_j > \beta_k > 0.5$ for jobs $j, k \in \bar{J}$. Assume that job k precedes job j in an optimal schedule. It is easy to see that swapping jobs j and k decreases the makespan, which implies that job k cannot precede job j in an optimal schedule. We note that the jobs in $\bar{J}$ cannot be the first of an interleaving pair, and swapping jobs j, k does not therefore change the order of the other jobs. □

We now investigate the case of $\beta_j \leq 0.5, \exists j \in N$. There might be some possibility for interleaving of jobs, as shown in the following scenario. Let $l = (p, p, b_l = \beta_l s_l)$ and $k = (p, p, b_k = \beta_k s_k)$ be a two-job instance of problem $(p, p, \beta_j s_j)$. Also, let $\beta_l \leq 0.5$ and $\beta_k > 0.5$. Assume that the schedule starts at time 0 and the first completion task therefore starts at time $2p$. Since $\beta_l \leq 0.5$ and thus $\beta_l(2p) \leq p$, the jobs can be interleaved if the schedule starts with job l. On the contrary, because $\beta_k > 0.5$, and therefore $\beta_k(2p) \nleq p$, the interleaving of jobs is not possible if the schedule starts with job k. The Gantt chart of these two cases are illustrated in Figure 4.2. The makespan for those cases can be derived as follows.

$$(l,k) : C_{(1)} = p + p + p + 3p\beta_k = 3p + 3p\beta_k, \tag{4.6}$$

$$(k,l) : C_{(2)} = p + p + 2p\beta_k + p + p + (4p + 2p\beta_k)\beta_l = 4p + 2p\beta_k + (4p + 2p\beta_k)\beta_l. \tag{4.7}$$

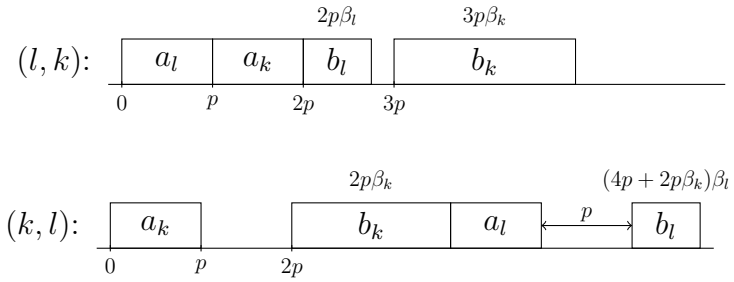

Figure 4.2 : Two possible schedules for a two-job instance: (l,k), where interleaving occurs, and (k,l), where interleaving is not possible.

Obviously, we are interested in finding the values of β_l and β_k such that $C_{(1)} \leq C_{(2)}$:

$$\begin{aligned}
&3p + 3p\beta_k \leq 4p + 2p\beta_k + (4p + 2p\beta_k)\beta_l \Longrightarrow \\
&\quad p\beta_k \leq p + 4p\beta_l + 2p\beta_l\beta_k \Longrightarrow \\
&\quad \beta_k \leq 1 + 4\beta_l + 2\beta_l\beta_k \Longrightarrow \\
&\quad \beta_k - 2\beta_l\beta_k \leq 1 + 4\beta_l \Longrightarrow \\
&\quad\quad \beta_k(1 - 2\beta_l) \leq 1 + 4\beta_l.
\end{aligned} \tag{4.8}$$

Note that if $\beta_l = 0.5$, Inequality (4.8) always holds, i.e., interleaving is beneficial. Following this, we propose Lemma 2.

Lemma 2. *If there exists a job l with $\beta_l = 0.5$, and the remaining jobs with $\beta_j > 0.5, \forall j \in N \setminus \{l\}$, an optimal schedule is obtained among the following two schedules: (1) interleaving job l with the job with the largest value of $\beta_j, j \in N \setminus \{l\}$, and sequencing the remaining jobs in non-increasing order of their β values, and (2) scheduling all jobs in non-increasing order of their β values.*

Proof. The largest improvement in the makespan by performing an interleaving is obtained when interleaving job l with the job with the largest value of β. An optimal sequence for the remaining jobs can be determined by Theorem 1. In case job l is not scheduled first, it cannot be interleaved anymore, and hence the problem is solved by Theorem 1. □

Lemma 2 further shows that it is only enough to investigate the potential of interleaving when $0 < \beta_l < 0.5$. Given a pair of jobs l, k, Inequality (4.9) calculates a threshold for $\beta_k > 0.5$ such that an interleaving improves the makespan:

$$\beta_k \leq \frac{1 + 4\beta_l}{1 - 2\beta_l}. \tag{4.9}$$

This leads to the following theorem.

Theorem 2. *In a two-job (l, k) instance of problem $(p, p, \beta_j s_j)$, an interleaving reduces the makespan if* $0.5 < \beta_k \leq \frac{1+4\beta_l}{1-2\beta_l}, 0 < \beta_l < 0.5$.

Proof. As discussed above. □

We note that Theorem 2 does not necessarily hold when $n \geq 3$. A counter example is shown in Figure 4.3. The optimal schedule for a three-job instance with $\beta_1 = 0.1$, $\beta_2 = 1$, $\beta_3 = 1.5$ and $p = 1$ does not follow Theorem 2, because the theorem implies that we may schedule an interleaving pair of jobs 1 and 3 at the beginning of the schedule since $\beta_3 < \frac{1+4\beta_1}{1-2\beta_1}$ (prioritising job 3 to job 2 since $\beta_3 > \beta_2$ due to applying Lemma 1), followed by job 2. However, the optimal sequence is $(3, 1, 2)$.

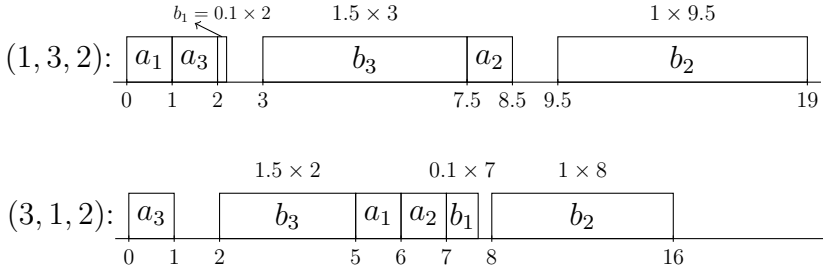

Figure 4.3 : Counter example for generalising the result of Theorem 2.

Groups of identical jobs

Although the computational complexity of problem $(p, p, \beta_j s_j)$ under arbitrary values of β remains open, we now investigate a polynomially solvable case, in which jobs are partitioned into a set of $G = \{1, \ldots, \mu\}, |G| = \mu$ groups. An important characteristic of group $g \in G$ is that all of its jobs share the same processing rate denoted by β_g.

The simplest case includes only one group of jobs, i.e., $\mu = 1$, implying that all jobs are identical and in the form of $(p, p, \beta s_j)$. The problem can easily be solved because the sequence is immaterial. We further show this in Section 4.1.3. When $\mu > 1$, however, the number of all possible permutations of jobs grows exponentially. As an example, consider $\mu = 2$. For simplicity, let n be an even number and let each group have an equal number of jobs. Therefore, $\frac{n}{2}$ jobs have a processing rate of β_1 and the remaining $\frac{n}{2}$ jobs have a processing rate of β_2. The number of all possible permutations of jobs is equal to $\frac{n!}{\frac{n}{2}!\frac{n}{2}!}$. Next, we present a dynamic program for problem $(p, p, \beta_j s_j)$.

The dynamic programming algorithm

Let $i = 1, \ldots, n$ denote the current stage of the algorithm, where the total number of stages is equal to the number of jobs. At stage i the set of i first jobs is scheduled. Let π

denote the set of job-groups at stage i and j denote the last job scheduled in stage i. We denote by $z^i_{\pi,j} = (c^i_{\pi,j}, t^i_{\pi,j})$ the state of the system at stage i, where $t^i_{\pi,j}$ represents two operations of "interleaving" (int) or "appending" (app) for the next job in the sequence. In case $t^i_{\pi,j} = app$, $c^i_{\pi,j}$ represents the completion time of i first jobs, and in case $t^i_{\pi,j} = int$, $c^i_{\pi,j}$ shows the earliest time the second task of the next job can start. The recursive formula for $z^i_{\pi,j}, 1 \leq i \leq n-1$ is shown in Equation (4.10).

$$z^i_{\pi,j} = (c^i_{\pi,j}, t^i_{\pi,j}) = \begin{cases} (c^{i-1}_{\pi\setminus\{j\}} + b_j, app) & \text{if } t^{i-1}_{\pi\setminus\{j\}} = int, \\ (c^{i-1}_{\pi\setminus\{j\}} + 2p + b_j, app) & \text{if } t^{i-1}_{\pi\setminus\{j\}} = app \wedge b_j > p, \\ (c^{i-1}_{\pi\setminus\{j\}} + 3p, int), (c^{i-1}_{\pi\setminus\{j\}} + 2p + b_j, app) & \text{if } t^{i-1}_{\pi\setminus\{j\}} = app \wedge b_j \leq p. \end{cases} \tag{4.10}$$

where

$$b_j = \begin{cases} \beta_j(c^{i-1}_{\pi\setminus\{j\}}) & \text{if } t^{i-1}_{\pi\setminus\{j\}} = int, \\ \beta_j(c^{i-1}_{\pi\setminus\{j\}} + 2p) & \text{if } t^{i-1}_{\pi\setminus\{j\}} = app. \end{cases} \tag{4.11}$$

Given that $c^{i-1}_{\pi\setminus\{j\}}$ represents the completion time of the $i-1$ first jobs and $s^{i-1}_{\pi\setminus\{j\}}$ and $t^{i-1}_{\pi\setminus\{j\}}$ denote the starting time and the operations "int" or "app" for the last job in the sequence of $i-1$ first jobs, then at each stage $i > 1$, we show the state of the system at the previous stage by $z^{i-1}_{\pi\setminus\{j\}}$.
The initial state is $(c^0_\varnothing, t^0_\varnothing) = (0, app)$, and the final state is

$$z^n_{\pi,j} = c^n_{\pi,j} = \begin{cases} c^{n-1}_{\pi\setminus\{j\}} + b_j & \text{if } t^{n-1}_{\pi\setminus\{j\}} = int, \\ c^{n-1}_{\pi\setminus\{j\}} + 2p + b_j & \text{if } t^{i-1}_{\pi\setminus\{j\}} = app. \end{cases} \tag{4.12}$$

The four possible cases for $z^i_{\pi,j}$ in Equation (4.10) are as follows. If $t^{i-1}_{\pi\setminus\{j\}} = int$, job j is interleaved with the last job in the sequence and the next job should be in the form of appending (case 1). If $t^{i-1}_{\pi\setminus\{j\}} = app$, job j will be appended, but the possibilities for the next job depend on the value of b_j. If $b_j > p$, the next job is also in the form of appending since it cannot be interleaved with job j (case 2). However, if $b_j \leq p$, the next job can be interleaved with job j. We point that both interleaving and appending (cases 3 and 4, respectively) must be considered for the next job. Consider a three-job instance, where $\beta_1 = 0.25, \beta_2 = 0.20, \beta_3 = 0.17$ and $p = 1$. Figure 4.4 shows that although job 2 can be interleaved with job 1, appending it leads to the optimal schedule (Figure 4.4a).

We note that at most three cases, out of four, need to be considered in any stage. Also, at each stage i, for any π, from the cases with similar $t^i_{\pi,j}$ the one with smaller $c^i_{\pi,j}$ is stored for the next stage. Therefore, in each stage at most two options of appending and interleaving may be possible.

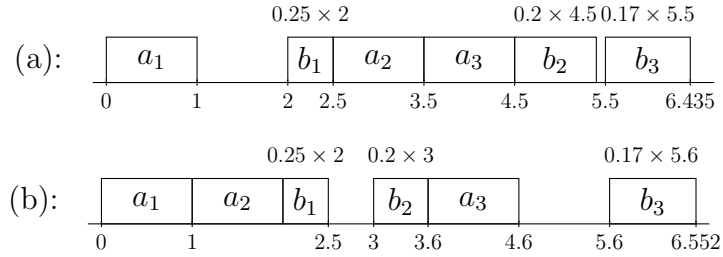

Figure 4.4 : A three-job example showing that appending job 2 (a) leads to a smaller makespan than interleaving jobs 1 and 2 (b).

Complexity of the proposed dynamic program

We assume that jobs are partitioned into μ groups. Let first consider the case that each group contains an equal number of jobs, which is $\frac{n}{\mu}$.

In stage i there is a number of candidates for π. Each candidate includes exactly i jobs. The number of candidates depends on both i and μ because we only distinguish jobs by their group(s). In stage $i, 1 \leq i \leq \frac{n}{\mu}$, the number of candidates, which we denote by η, is equal to the number of solutions of Equation (4.13):

$$\sum_{\mu'=1}^{\mu} x_{\mu'} = i, \quad 0 \leq x_{\mu'} \leq \frac{n}{\mu}, \forall \mu' \in \{1, \ldots, \mu\}. \tag{4.13}$$

Since both i and $x_{\mu'}$ are bounded from above by $\frac{n}{\mu}$, η is equal to $\binom{i+\mu-1}{i}$. In stage $i, \frac{n}{\mu} < i \leq n$, η is still derived by using Equation (4.13); however, out of the total number of $\binom{i+\mu-1}{i}$ some are invalid. More precisely, because $i > \frac{n}{\mu}$ the solutions including $x_{\mu'} > \frac{n}{\mu}, \exists \mu' \in \{1, \ldots, \mu\}$ are not considered, leading to a smaller value of η. Therefore, η in any stage i is not greater than $\binom{i+\mu-1}{i}$. Now consider the case where the groups do not contain an equal number of jobs. We can still derive η by using Equation (4.13), however, again some of the solution are invalid, and hence, η in stage i is never greater than $\binom{i+\mu-1}{i}$.

Additionally, in stage i there are at most μ candidate jobs to occupy the last position because μ groups of jobs exist. Also, at most three cases of appending or interleaving need to be considered. Hence, the time complexity of stage i is in the order of $O(\mu\binom{n+\mu-1}{n})$, implying that the time complexity of the proposed dynamic program is $O(n\mu\binom{n+\mu-1}{n})$, which is shown by Theorem 3 to be polynomial for fixed values of μ.

Theorem 3. *The proposed dynamic program solves model S3-T with n jobs and μ groups of identical jobs in $O(\mu n^{\mu})$.*

Proof. The proof is by induction:

If $\mu = 2$, then $2n\binom{n+2-1}{n} = 2n(n+1) \approx O(n^2)$,

If $\mu = 3$, then $3n\binom{n+3-1}{n} = 3n(n+2)(n+1)/2 \approx O(n^3)$,

If $\mu = 4$, then $4n\binom{n+4-1}{n} = 4n(n+3)(n+2)(n+1)/6 \approx O(n^4)$,

....

In general, if μ groups of identical jobs exist, the time complexity is $O(\mu n^{\mu})$. We note that when $\mu = n$, the complexity of the dynamic program is $O(n^{n+1})$. □

The heuristic algorithm

In Section 4.1.2, we showed that in problem $(p, p, \beta_j s_j)$ the first priority must be given to jobs with greater values of β_j (implied by Theorem 1). However, an interleaving of jobs may potentially decrease the makespan if Theorem 2 holds. Therefore, when constructing a schedule the only two available options at any point include (1) appending a single job, or (2) interleaving a pair of jobs. We utilise those principles and develop a heuristic algorithm for model S3-T (see Algorithm 2), as follows.

Let $T = J$ be the set of unscheduled jobs and $S = ()$ be the sequence of performing jobs. Each iteration of the constructive heuristic consists of identifying a single job to be appended, or a pair of jobs to be interleaved. Let us assume that the jobs can start at time 0. Therefore, the starting time of the completion task in the first iteration is $s_1 = 2p$. At every iteration $i \geq 1$, a threshold on β is calculated: $\beta_{thr} = \frac{p}{s_i}$ (the threshold is used to identify jobs with $b_j \leq p$). The subset of jobs with $\beta_j \leq \beta_{thr}$ are identified as the jobs that can be the first of a potential interleave. From those, the job with the largest value of β is selected. Let l denote this job. The other job, say k, is then selected such that it has the largest value of β among all jobs.

Then, it is checked whether the bound $\beta_k \leq \frac{1+4\beta_l}{1-2\beta_l}$ is satisfied by job k. If yes, the interleaving pair of jobs l and k is scheduled, where job l is the first job of the interleaving pair. Otherwise, job k is appended to S. At the end of each iteration, the starting time of the next completion task, and S and T are updated. The procedure continues until all jobs are scheduled, or no interleaving is possible, i.e., $b_j \nleq p, j \in T$. In this case, the remaining jobs are appended to S in non-increasing order of β_j.

By utilising the delay periods, Algorithm 1 constructs as many interleaving pairs as possible, while it gives higher priority to the jobs with larger value of β. The total number of iterations performed by the algorithm is at most equal to the number of jobs. Because finding jobs l and k in each iteration requires $O(n)$ time, the algorithm therefore has a time complexity of $O(n^2)$.

The following numerical example is presented to clarify the operation of Algorithm 1. There are four jobs with β values of $\{0.1, 0.15, 0.18, 3.0\}$ and $p = 1$. We initialise $T = \{1, 2, 3, 4\}$ and $S = ()$. Assuming that we start at time 0, then $s_1 = 2$. Table 4.1 shows that the algorithm appends job 4 in the first iteration. In the second iteration, jobs 1 and 3 are interleaved, where job 1 is the first job of the interleaving pair. Then, because the condition in line 5 of Algorithm 1 is not satisfied, the loop is terminated and the

Algorithm 1: The construction procedure of the heuristic algorithm.

Input: $S = (), T = J, p, \beta_j, \forall j \in N, s_1 = 2p$.
Output: A sequence S with makespan C_S.
for $i = 1$ *to* n **do**
  $\beta_{thr} = \frac{p}{s_i}$;
  if $\exists j \in T, \beta_j \leq \beta_{thr}$ **then**
    $l \leftarrow \arg\max_{j \in T}(\beta_j | \beta_j \leq \beta_{thr})$;
    $k \leftarrow \arg\max_{j \in T}(\beta_j)$;
    if $\beta_k \leq \frac{1+4\beta_l}{1-2\beta_l}$ **then**
      Interleave jobs l and k adjacently;
      $s_{i+1} = s_i + (\beta_k)(s_i + p) + 3p$;
      $S \leftarrow S \cup \{l, k\}$;
      $T \leftarrow T \setminus \{l, k\}$;
    else
      Append job k adjacently;
      $s_{i+1} = s_i + (\beta_k)s_i + 2p$;
      $S \leftarrow S \cup \{k\}$;
      $T \leftarrow T \setminus \{k\}$;
    end
  else
    Break;
  end
end
Adjacently append the remaining jobs in T to S, in non-increasing order of β_j;
return S;

remaining job, i.e., job 2 is appended. The Gantt chart depicted in Figure 4.5 shows the schedule delivered by Algorithm 1, that is the optimal schedule.

Table 4.1 : The operation of Algorithm 1 for a four-job instance.

Step i	β_{thr}	l	k	S	T	s_{i+1}
1	$\frac{1}{2}$	3	4	(4)	$\{1, 2, 3\}$	10
2	$\frac{1}{10}$	1	3	(4, 1, 3)	$\{2\}$	14.98
3	$\frac{1}{14.98}$	-	-	-	-	-

The schedule obtained by Algorithm 1 can further be improved. In that regard, we iteratively apply swap moves, as presented in Algorithm 2. We implement the "first improvement" criterion, i.e., once an improving solution is obtained it is accepted and the schedule is updated. The run time of Algorithm 2 is $O(n^2)$, and hence the run time of the proposed heuristic is $O(n^2)$.

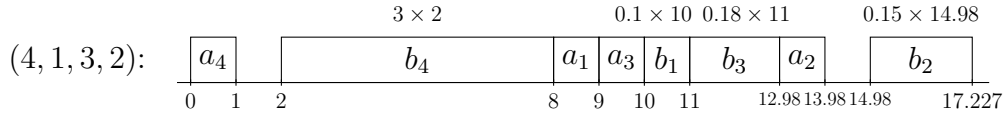

Figure 4.5 : The schedule for a four-job instance delivered by Algorithm 1.

Algorithm 2: The improvement procedure of the heuristic algorithm.

Input: $\beta_j, \forall j \in N$, S_0, C_{S_0}, $j = 1$.
Output: A sequence S with makespan C_S.
$S = S_0$;
$C_S = C_{S_0}$;
while $j \leq n - 1$ **do**
  $Improve = 0$;
  for $k = j + 1 : n$ **do**
    $S' \leftarrow \text{swap}(j, k)$;
    $C_{S'} \leftarrow \text{makespan}(S')$;
    if $C_{S'} < C_S$ **then**
      $S = S'$;
      $C_S = C_{S'}$;
      $Improve = 1$;
    end
  end
  if $Improve = 0$ **then**
    $j = j + 1$;
  end
end
return S;

4.1.3 Lower bound

We derive a lower bound for model S3-T by letting $\beta_j = \min_{j \in N} \beta_j, \forall j \in N$, i.e., all jobs have an identical processing rate. Theorem 4 shows this.

Theorem 4. *Optimising model S3-T under the setting $\beta_j = \beta_{\min} = \min_{j \in N} \beta_j, \forall j \in N$ leads to a makespan, which is never greater than the makespan under arbitrary values for β_j.*

Proof. Assume that the makespan under the setting $\beta_{\min} = \min_{j \in N} \beta_j$ is larger than the makespan under the arbitrary values for β. Then, there exits an optimal makespan where $\beta_j > \beta_{\min}, \exists j \in N$. Since the initial task and the delay period take identical values for all jobs, the makespan under $\beta_j > \beta_{\min}, j \in N$ is never smaller than the one under $\beta_{\min}$, implying that the initial assumption is contradicted. □

Next, we show that obtaining this lower bound is trivial.

Lemma 3. *Under the setting $\beta_j = \beta_{\min}, \forall j \in N$ the makespan for model S3-T is minimised if the sequence includes a number of adjacent interleaving pairs followed by appending the remaining jobs once no interleaving is possible (sequence 1), or if the sequence includes appending a single job at the beginning, followed by a number of adjacent interleaving pairs and then appending the remaining jobs (sequence 2).*

Proof. We note that the schedule with the minimum makespan consist of as many interleaving pairs (of jobs) as possible. Intuitively, this implies that a number of adjacent interleaving pairs followed by appending the remaining jobs once no interleaving is possible must lead to the minimum makespan. We show that in some cases a better makespan (with smaller value) is obtained if we first append a single job, and then add a set of interleaving pairs followed by a set of appending jobs. We note that interleaving is possible as long as the completion task of the first job (of an interleaving pair) starts no later than $\frac{p}{\beta_{\min}}$.
Let illustrate sequences 1 and 2 by a three-job example, where $p = 1$ and $\beta_{\min} = 0.1$. Sequence 1 consists of one interleaving pair, followed by appending the third job. This results in $C_{\max} = 5.83$ (see Figure 4.6a). In sequence 2, the third job is scheduled before the interleaving pair. This results in $C_{\max} = 5.72$, i.e., the minimum makespan (see Figure 4.6b).

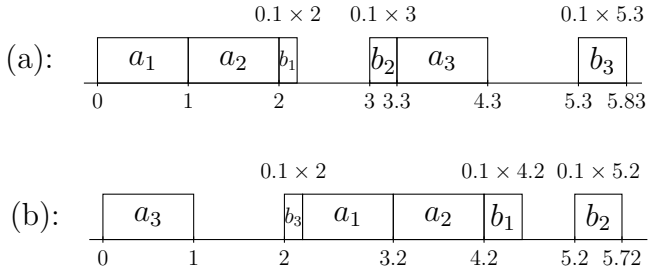

Figure 4.6 : An instance to illustrate calculation of the lower bound for model S3-T.

There is no possibility to append two (or more) jobs at the beginning of the sequence because an interleaving pair of those jobs would complete earlier than appending them adjacently. □

4.1.4 Computational experiments

We evaluate the performance of the proposed heuristic on a set of 120 randomly generated instances. The instances include 5, 10, 20, 50, 75 and 100 jobs (n). We set the parameter β in a way to allow some interleaving in the schedule. Since large values of β result in less possibility for interleaving, and hence, easier instances, we therefore consider two settings. For the first setting, we randomly select β from the continuous uniform distribution such that $\beta_j \in (0, 0.1), \forall j \in N$, and for the second setting $\beta_j \in$

$(0, 0.2), \forall j \in N$. We generated 10 instances for each combination of n and β. This results in 120 instances in total. We set $p = 1$ for all instances.

We also solve the instances by optimising model S3-T with the solver Gurobi Optimizer version 8.0.0 (Gurobi Optimization, 2018). We perform all computational experiments on the same PC mentioned in Section 3.3. We set a time limit of 3,600 seconds for the solver Gurobi. We utilise one processor (thread) for the heuristic algorithm, however, we run the Gurobi by using one processor and four processors (denoted as Gurobi^1 and Gurobi^4). For the remaining parameters of the solver Gurobi we used the default values.

Table 4.2 reports the outcomes of the heuristic algorithm, denoted as "Heur$_{cons}$" and Gurobi. We use two criteria of feas and opt (see Section 3.3). According to the results, Gurobi^1 and Gurobi^4 generate feasible solution for only 71 instances, out of 120 (i.e., for almost 59%). Within 3,600 seconds of running, Gurobi reports feasible solution for only one instance with 75 jobs and $\beta_j \in (0, 0.1)$; it also does not report feasible solution for the instances with 50 jobs and $\beta_j \in (0, 0.2)$. For the instances with 100 jobs, Gurobi runs out of memory. The performance of Gurobi^4 is slightly better than that of Gurobi^1 since it obtains three additional optimal solutions. The proposed heuristic, however, delivers feasible solutions for all instances. Interestingly, the heuristic produces the same best solutions for 18 of those instances, i.e., for 45%.

Table 4.2 : Number of feasible and optimal solutions delivered by Heur$_{cons}$ and Gurobi.

n	Setting for β	Feas			Opt		
		Heur$_{cons}$	Gurobi^1	Gurobi^4	Heur$_{cons}$	Gurobi^1	Gurobi^4
5	(0, 0.1)	10	10	10	0	10	10
	(0, 0.2)	10	10	10	2	10	10
10	(0, 0.1)	10	10	10	9	7	10
	(0, 0.2)	10	10	10	7	10	10
20	(0, 0.1)	10	10	10	0	0	0
	(0, 0.2)	10	10	10	0	0	0
50	(0, 0.1)	10	10	10	0	0	0
	(0, 0.2)	10	0	0	0	0	0
75	(0, 0.1)	10	1	1	0	0	0
	(0, 0.2)	10	0	0	0	0	0
100	(0, 0.1)	10	0	0	0	0	0
	(0, 0.2)	10	0	0	0	0	0
Total		120	71	71	18	37	40

Table 4.3 reports two criteria of gap (%) and time (sec) (see Section 3.3). Consistent with earlier findings, for small instances with 5 and 10 jobs the solver Gurobi outperforms the proposed heuristic. For larger instances, however, the heuristic delivers improved solutions. Particularly, we note that both versions of Gurobi have a gap of 75.42% and 77.16% for instances with 50 and 75 jobs, not to mention that because Gurobi is not able

to report any feasible solution for four groups of instances, the value of gap cannot be calculated for those instances ("-" in Table 4.3 shows this).

Table 4.3 : Gap (in %) and the computation time for Heur_{cons} and Gurobi.

n	Setting for β	Gap (in %)			Time		
		Heur_{cons}	Gurobi^1	Gurobi^4	Heur_{cons}	Gurobi^1	Gurobi^4
5	(0, 0.1)	1.08	0.00	0.00	< 0.01	0.20	0.18
	(0, 0.2)	1.13	0.00	0.00	< 0.01	0.18	0.16
10	(0, 0.1)	0.01	0.00	0.00	< 0.01	3167.36	1526.39
	(0, 0.2)	1.22	0.00	0.00	< 0.01	443.31	199.38
20	(0, 0.1)	0.00	6.52	4.86	0.02	3600.00	3600.03
	(0, 0.2)	0.14	4.45	3.81	0.02	3600.01	3600.03
50	(0, 0.1)	0.00	75.42	75.42	0.25	3600.01	3600.07
	(0, 0.2)	0.00	-	-	0.31	3600.02	3600.02
75	(0, 0.1)	0.00	77.16	77.16	0.92	3600.15	3600.26
	(0, 0.2)	0.00	-	-	1.17	3600.28	3600.39
100	(0, 0.1)	0.00	-	-	2.36	-	-
	(0, 0.2)	0.00	-	-	2.79	-	-

Table 4.4 summarises the outcomes of Heur_{cons} and Gurobi^1 and Gurobi^4. The highlighted values denote the superiority of the method with respect to the criterion. As the table shows, the proposed heuristic performs very well, and obtains high quality solutions: its average gap is 0.30%, while its worst gap is nearly 1.22%. In addition, it is very efficient since it solves even the problems with 100 jobs within three seconds. The average time of both Gurobi^1 and Gurobi^4 is almost 40 minutes, and significantly increases with the number of jobs.

Table 4.4 : Overall results for Heur_{cons} and Gurobi.

Method	Feasible	Optimal	Gap (%)		Time (sec)	
			Ave	Max	Ave	Max
Heur_{cons}	**120**	18	**0.30**	**1.22**	**0.65**	**2.79**
Gurobi^1	71	37	13.25	77.16	2521.18	3600.28
Gurobi^4	71	**40**	12.93	77.16	2332.69	3600.39

To further evaluate the performance of the proposed heuristic, i.e., Heur_{cons}, we compare the values of its gap to the lower bound and those of the solver Gurobi. We report the outcomes in Table 4.5, where the values of gap are averaged over 10 instances per setting. The gap is calculated as $\frac{z-lb}{lb} \times 100$, where z is the objective function value, i.e., the makespan delivered by the method, and lb is the lower bound obtained via procedure explained in Section 4.1.3. We report the results only for instances with $n = 5, 10$ because proven optimal solutions are available only for these instances. The results indicate that

Table 4.5 : Gap to the lower bound for Heur_{cons} and Gurobi.

n	Setting for β	Heur_{cons}	Gurobi[1]	Gurobi[4]
5	(0, 0.1)	2.55	1.50	1.50
	(0, 0.2)	6.73	5.67	5.67
10	(0, 0.1)	3.72	3.71	3.71
	(0, 0.2)	23.34	22.58	22.58
Average		9.08	8.36	8.36

Heur_{cons} performs very closely to Gurobi because its average values of gap to the lower bound is very close to those of Gurobi.

Because Gurobi cannot deliver feasible solutions for large instances, we further assess the performance of the proposed heuristic, i.e., Heur_{cons} by solving the instances with two new settings and comparing the outcomes of Heur_{cons} and those of the two settings. For this purpose, we generate initial solutions via sorting the jobs in non-increasing and non-decreasing orders of their β values that results in two new variants for the heuristic, denoted as "Heur_{LPT}", "Heur_{SPT}", respectively. We summarise the results in Table 4.6, where the metric best denotes the number of best solutions obtained by each setting. The results show that Heur_{cons} obtains significantly better solutions than those two variants of Heur_{LPT} and Heur_{SPT}. Indeed, Heur_{cons} obtains the best solution in 108 instances. The average gap of Heur_{cons} over all instances is almost 0.44%, that is much lower than the average gap of the two settings of Heur_{LPT} and Heur_{SPT}. Those results further indicate the quality of solutions produced by the proposed heuristic.

Table 4.6 : Assessing the performance of Heur_{cons}, Heur_{LPT} and Heur_{SPT}.

n	Setting for β	Heur_{cons}		Heur_{LPT}		Heur_{SPT}	
		Gap (in %)	Best	Gap (in %)	Best	Gap (in %)	Best
5	(0, 0.1)	1.08	10	1.08	10	1.08	10
	(0, 0.2)	2.01	10	2.01	10	2.01	10
10	(0, 0.1)	0.01	9	0.01	9	0.04	9
	(0, 0.2)	1.22	8	3.10	5	2.48	5
20	(0, 0.1)	0.52	6	1.11	7	1.91	2
	(0, 0.2)	0.40	9	3.13	2	9.80	0
50	(0, 0.1)	0.02	8	2.23	2	10.95	0
	(0, 0.2)	0.00	10	5.71	0	6.88	0
75	(0, 0.1)	0.00	10	2.86	0	10.23	0
	(0, 0.2)	0.00	9	4.94	1	10.43	0
100	(0, 0.1)	0.07	9	2.87	1	7.09	0
	(0, 0.2)	0.00	10	4.61	0	12.61	0
Average / total		**0.44**	**108**	2.81	47	6.29	36

4.2 Binary search algorithm

In this section we consider the classical single machine CTSP, i.e., where all processing times and delay durations are integral, with the objective of minimising the makespan. As discussed in Section 3.3, the standard exact solvers can only optimally solve small instances in a reasonable amount of time. Though, their performance is not guaranteed for larger instances. We aim at proposing more efficient solution methods for the problem, and that utilising the available formulations. We utilise model S3 as the mathematical formulation.

The general idea of the proposed algorithm is as following. First, a lower bound (LB) and an upper bound (UB) on the value of the optimal makespan are calculated. A point in the interval $[LB, UB]$ is selected as the binary bound bb. Then, the heuristic checks whether it is possible (feasible) to schedule all jobs such that the makespan is bounded from above by the binary bound, i.e., $C_{\max} \leq bb$. For this reason, the heuristic utilises an exact solver to solve a feasibility problem associated with Model S3. A feasible solution means that bb is a valid makespan, implying that UB can be lowered to bb, i.e., $UB = bb$. If there is no feasible solution, the lower bound is updated to bb, i.e., $LB = bb$. Therefore, at each iteration either LB or UB is tightened. If the length of interval between the lower and upper bounds is 1, i.e., $UB - LB = 1$, the upper bound is equal to the optimal makespan because we deal with integer values for all problem input data. We note that because problem $1|(a_j, L_j, b_j)|C_{\max}$ is strongly NP-hard it is less likely that we observe a quick convergence of the proposed binary search heuristic in a reasonably short time. Therefore, we set the stopping criterion of the heuristic as either $UB - LB = 1$ or when a time limit is reached, whichever occurs the first. Under the latter condition the most recent upper bound is a valid makespan for the problem. Algorithm 3 summarises the proposed binary search heuristic.

Note that the binary search algorithm is also known as dichotomous search, and has been successfully applied to various optimisation problems including the traveling salesman problem (França et al., 1995), the project scheduling problem (Carlier and Néron, 2003) and the job-shop scheduling problem (Grimes and Hebrard, 2015).

4.2.1 Lower bound

In the single-machine CTSP, A trivial LB on $C_{\max}$ can be obtained by scheduling all tasks without any idle time between them (Li and Zhao, 2007). We denote this lower bound by lb_0, where $lb_0 = \sum_{j \in N} (a_j + b_j)$. However, exclusion of parameter $L_j, j \in N$ from lb_0 results in a loose lower bound. As a result, we present two remedies for this. Note that not always the values of L_j can be included in lb_0, for example, when $a_j = b_j = L_j = p, p \in \mathbb{Z}^+$.

Algorithm 3: The binary search heuristic for the CTSP.

```
Input: lb (a lower bound), ub (an upper bound), time_limit.
Output: A schedule.
UB := ub;
LB := lb;
elapsed_time := 0;
while UB − LB > 1 and elpased_time < time_limit do
    Calculate bb, bb ∈ [LB, UB];
    Set C_max ≤ bb;
    if a feasible solution exists then
        UB := bb;
    else
        LB := bb;
    end
end
return UB;
```

As the first remedy, lb_0 can be improved by checking whether there are some singleton jobs (see Section 2.1.1). For the singleton jobs the delay period cannot be utilised to schedule any other task. Hence, L_j associated with the singleton jobs can be included in the lower bound. We let lb_1 denote this, that can be calculated as Equation (4.14).

$$lb_1 = \sum_{j \in N} (a_j + b_j) + \sum_{\substack{j \in N \\ L_j < p_{\min}}} L_j. \tag{4.14}$$

Note that $lb_1 \geq lb_0$ for any instance of the problem.

As the second remedy, lb_0 can be improved if we are able to locate the jobs whose delay periods cannot be completely utilised to schedule other tasks. Assume that we aim to concatenate as much tasks as possible in the delay period of job j, which has a length of L_j. This can be modelled as a 0-1 knapsack problem. In other words, the initial and completion tasks of all jobs other than job j are eligible to be inserted into this delay period. We define p_h the processing time of task h. We note that if task h is an initial task for some job j, then we have $p_h = a_j$. Similarly, if task h is a completion task for some job j, then we have $p_h = b_j$. A binary decision variable y_h is also defined that takes the value of 1 if task h is selected. Such a concatenation problem can be formulated as problem K.

Problem K

$$z = \max \sum_{\substack{h \in H \\ h \notin \{2j, 2j-1\}}} p_h y_h \tag{4.15}$$

subject to

$$\sum_{\substack{h \in H \\ h \notin \{2j, 2j-1\}}} p_h y_h \leq L_j, \tag{4.16}$$

$$y_{2j'-1} + y_{2j'} \leq 1, \quad \forall j' \in N \setminus \{j\}, \quad \text{if} \quad a_{j'} + L_{j'} + b_{j'} > L_j, \tag{4.17}$$

$$y_{2j'-1} - y_{2j'} \geq 0, \quad \forall j' \in N \setminus \{j\}, \quad \text{if} \quad L_{j'} < a_j, \tag{4.18}$$

$$y_{2j'} - y_{2j'-1} \geq 0, \quad \forall j' \in N \setminus \{j\}, \quad \text{if} \quad L_{j'} < b_j, \tag{4.19}$$

where, the objective function (4.15) maximises the total processing time of the selected tasks, and it is forced to be no larger than the delay period of job j (constraint (4.16)). Constraints (4.17) imply that only either of tasks of job j' can be selected if the nesting of job j' inside job j is not possible. Constraints (4.18) (constraints (4.19)) ensure that if the delay period of job j' is not as large as the initial (completion) task of job j, the completion (initial) task of job j' would only be selected if its initial (completion) task is selected as well.

Solving problem K for job j results in whether L_j can be completely filled with some tasks. If not, it means that there is an idle time I_j within L_j, which is equal to $L_j - z_K^*$. This implies that in an optimal schedule of model S3 there will be an idle time (inside L_j) at least equal to I_j. If we solve problem K for all jobs $j \in N$ and add the maximum idle time among all found idle times, i.e., $\max_{j \in N}\{I_j\}$ to lb_0, a tighter lower bound can be obtained, denoted as lb_2 (see Equation (4.20)).

$$lb_2 = \sum_{j \in N} (a_j + b_j) + \max_{j \in N}\{I_j\}. \tag{4.20}$$

Although the knapsack problem is NP-hard in the ordinary sense, it can be efficiently solved even for large inputs by the pseudo-polynomial time dynamic program of Martello et al. (1999), that is able to solve instances with up to 10,000 items in less than a second. Note that lb_2 is a tighter lower bound than lb_0 since $lb_2 \geq lb_0$ for any instance of the problem. Also, we cannot add the summation of the idle times (instead of their maximum value) to lb_0 because the idle times may be overlapped. Given $lb_1 \geq lb_0$ and $lb_2 \geq lb_0$, Equation (4.21) follows directly:

$$C_{\max} \geq \max\{lb_1, lb_2\}. \tag{4.21}$$

Next, we propose an upper bound for the problem.

4.2.2 Upper bound

A trivial upper bound on $C_{\max}$ can be calculated as $\sum_{j \in N} (a_j + L_j + b_j)$, i.e., scheduling the jobs one by one without any interleaving or nesting. This bound, however, is expected to be of a large value, and might not therefore be tight because interleaving, nesting and pairwise interchange operations are excluded from the calculation of the bound. We may tighten this bound by applying a local search algorithm. The proposed local search algorithm, which is summarised in Algorithm 4, starts with a given sequence π_0 of jobs, and then iteratively improves it by performing adjacent pairwise interchanges. We set $\pi_0 = (1, \ldots, n)$, i.e., we add the jobs to π_0 in increasing order of their indices. In addition, we implement the first improvement criterion in the adjacent pairwise interchanges.

For a sequence π, the algorithm generates a feasible schedule with nesting, interleaving and appending operations as follows. For a pair of consecutive (adjacent) jobs $j, j' \in \pi$, where j comes before j', if nesting of job j' inside job j is possible, i.e., $a_{j'} + L_{j'} + b_{j'} \leq L_j$, then job j' is inserted inside job j. However, if nesting is not possible but interleaving, i.e., $L_j \geq a_{j'} \wedge L_{j'} \geq b_j$, then the interleaving is performed where job j is the first job in the pair. If neither nesting nor interleaving of a pair of jobs is possible, job j' is adjacently appended after job j. The algorithm performs this procedure for all consecutive (adjacent) pairs of jobs until all jobs are scheduled. The worst-case time complexity of the local search algorithm is $O(n^2)$.

4.2.3 The feasibility problem

Given lower and upper bounds on the value of the makespan, the next step is to systematically tighten the gap between the bounds until the stopping criterion is met, i.e., either no further tightening is possible or the computation time limit is reached.

We tighten the gap between the lower and upper bounds by iteratively solving a feasibility problem associated with Model S3. We generate such a feasibility problem by changing the objective function of Model S3 into a constant and bounding $C_{\max}$ from above by bb. Problem S3-F in the following represents the feasibility problem.

Problem S3-F

$$z = \min \zeta \tag{4.22}$$

subject to

(3.21), (3.23) to (3.29),

$$C_{\max} \leq bb, \tag{4.23}$$

where ζ is a constant. Constraints (3.22) are not included in Model S3-F because they do

Algorithm 4: The local search algorithm.

```
Input: π_0 = (1, ..., n), C_{π_0}, j = 1.
Output: A sequence π with makespan C_π.
π := π_0;
C_π := C_{π_0};
while j ≤ n − 1 do
    improve := false;
    k := j + 1;
    for k ≤ n do
        π' ← swap(j, k);
        S_{π'} ← Generate a feasible schedule for π';
        C_{π'} ← makespan(S_{π'});
        if C_{π'} < C_π then
            π := π';
            C_π := C_{π'};
            improve := true;
        end
    end
    if improve is false then
        j := j + 1;
    end
end
return C_π;
```

not impact the feasibility of the problem. In other words, if we exclude constraints (3.22), the feasibility model will return the same solution as the original model where this set of constraints are included. Constraint (4.23) sets the binary bound as an upper bound on the value of the makespan.

Solving problem S3-F is equivalent to identifying a feasible solution for model S3. Also, if problem S3-F does not have a feasible solution, neither does model S3. We may use standard optimisation solvers for solving problem S3-F. Even though problem S3-F might be easier to solve than model S3 because at least no "optimisation" phase is performed, it might be still challenging to find a feasible solution for problem S3-F, implying that the process may be computationally expensive, particularly for large problem instances. We propose two speed-up techniques to improve the computation burden of solving problem S3-F.

Speed-up 1. The first speed-up that we propose benefits from the "call-back" functionality of the solver. A call-back provides additional information during the solve process of the solver. We can use such additional information to stop the solver as soon as a feasible solution is found.

Speed-up 2. The second speed-up focuses on modifying the "solve focus" of the solver.

Various solvers have different name for this feature. For example, the solver Gurobi names the feature "MIPfocus". Changing the focus of the solver towards obtaining a feasible solution, rather than, e.g., an optimal solution (which focuses on proving the optimality of the current solution), is an effective computation burden reduction strategy.

4.2.4 Computational experiments

We perform an extensive computational experiment to evaluate the performance of the proposed binary search heuristic. We test the algorithm on the "general set" instances of the CTSP proposed in Section 3.1.2.

We utilise Gurobi version 8.0.0 (Gurobi Optimization, 2018), as the solver within the binary search heuristic algorithm, to solve problem S3-F. We use the same solver Gurobi as the stand-alone solver to solve the same instances by optimising model S3. Also, we utilise Gurobi to solve problem K (for delivering LB). Unless otherwise stated we perform all computational experiments on the same PC mentioned in Section 3.3, and we set the time limit to 3,600 seconds for both the binary search heuristic and the stand-alone Gurobi and we utilise four processors. We use the default value for the remaining parameters of Gurobi.

Recall that the binary bound is in the ranges $[LB, UB]$, i.e., $bb \in [LB, UB]$. We investigate two strategies to choose bb: (1) $bb = \frac{lb+ub}{2}$, i.e., it is equal to the midpoint of the interval, and (2) $bb = \frac{lb+3ub}{4}$, i.e., it is equal to the three fourth of the interval. We solve the 240 instances with both strategies, each with setting 3,600 seconds time limit. We let $bb_{1/2}$ and $bb_{3/4}$ denote these two strategies.

We use the four criteria of feas, best, opt and gap (in %) (see Section 3.3), to compare the proposed binary search heuristic and Gurobi. We report the outcomes of the computational experiments in Tables 4.7 to 4.10, where a table represents one evaluation criterion.

Table 4.7 summarises the performance of the methods for the criterion feas. The table shows that the binary search heuristic under both $bb_{1/2}$ and $bb_{3/4}$ strategies obtain feasible solution for all 240 instances, whereas Gurobi is unable to find feasible solution for 43 instance, for which $n = 40, 50, 100$. In particular, Gurobi cannot deliver feasible solution for any of the instances with 100 jobs. This shows that Gurobi is not able to even generate acceptable solution for large instances. Also, Gurobi's performance deteriorates for medium sized instances.

Table 4.8 shows that Gurobi is slightly superior to the binary search in obtaining an optimal solution for the instances with a small number of jobs. We note that the binary search heuristic under both strategies still obtains an optimal solution for all instances with 5 jobs but one. Both the binary search heuristic and Gurobi find an optimal solution for all instances with 10 jobs. For the instances with 15 jobs, Gurobi finds an optimal

Table 4.7 : The number of feasible solutions obtained by the binary search and Gurobi.

n	Job category	$bb_{1/2}$	$bb_{3/4}$	Gurobi
5	S	10	10	10
	M	10	10	10
	L	10	10	10
10	S	10	10	10
	M	10	10	10
	L	10	10	10
15	S	10	10	10
	M	10	10	10
	L	10	10	10
20	S	10	10	10
	M	10	10	10
	L	10	10	10
25	S	10	10	10
	M	10	10	10
	L	10	10	10
40	S	10	10	9
	M	10	10	10
	L	10	10	10
50	S	10	10	4
	M	10	10	6
	L	10	10	8
100	S	10	10	0
	M	10	10	0
	L	10	10	0
Total		240	240	197

solution for only two instances. Overall, Gurobi delivers optimal solution for 62 instances, i.e., only for three more instances than the binary search heuristic.

We report the results of the criterion best, i.e., the number of best obtained solutions, in Table 4.9. Regarding this criterion, the strategy $bb_{3/4}$ outperforms both the strategy $bb_{1/2}$ and Gurobi, particularly for large instances. We note that Gurobi is not comparable to the binary search heuristic when the number of jobs increases, more precisely, when $n = 25, 40, 50, 100$. A similar outcome is observed for the criterion gap, which is reported in Table 4.10. Indeed, Gurobi performs well only for instances with a small number of jobs. The Gurobi's solutions are of poor quality when $n \geq 25$. The strategy $bb_{3/4}$ performs better than $bb_{1/2}$ and also than Gurobi, because it has the smallest average gap. In Table 4.10, the "-" indicates that the criterion gap cannot be calculated because not a single feasible solution was reported by Gurobi.

We summarise the outcomes of both tested strategies of the binary search heuristic and Gurobi in Table 4.11, where the highlighted values denote the superior ones. The table shows that all methods perform close to one another for $n = 5, 10, 15, 20$. Once the number of jobs increases, Gurobi cannot even deliver a feasible solution for any problem

Table 4.8 : The number of optimal solutions obtained by the binary search and Gurobi.

n	Job category	$bb_{1/2}$	$bb_{3/4}$	Gurobi
5	S	10	10	10
	M	10	10	10
	L	9	9	10
10	S	10	10	10
	M	10	10	10
	L	10	10	10
15	S	0	0	1
	M	0	0	0
	L	0	0	1
20	S	0	0	0
	M	0	0	0
	L	0	0	0
25	S	0	0	0
	M	0	0	0
	L	0	0	0
40	S	0	0	0
	M	0	0	0
	L	0	0	0
50	S	0	0	0
	M	0	0	0
	L	0	0	0
100	S	0	0	0
	M	0	0	0
	L	0	0	0
Total		59	59	62

instance. When Gurobi does find a feasible solution, however, it is usually of poor quality. The binary search heuristic reports superior solutions for large instances. The results also indicate that the binary search heuristic under the strategy $bb_{3/4}$ outperforms the one under the strategy $bb_{1/2}$.

4.3 Proposed new formulation

In this section, we propose several enhancements to improve the computational performance of model S3. Particularly, we propose two directions to improve the computational performance of model S3. First, we identify constraints that can be removed from model S3 without an impact on its validity; that is, we identify constraints that removing them from model S3 will not change the model's solution. Second, we introduce new constraints that may improve the computational performance of model S3. In the following, we explore those directions.

4.3.1 Removing existing constraints

The three sets of constraints (3.23), (3.25) and (3.27) can be removed from model S3 without violating the validity of it. In other words, the solution of the model while those

Table 4.9 : The number of best solutions obtained by the binary search and Gurobi.

n	Job category	$bb_{1/2}$	$bb_{3/4}$	Gurobi
5	S	10	10	10
	M	10	10	10
	L	9	9	10
10	S	10	10	10
	M	10	10	10
	L	10	10	10
15	S	0	1	9
	M	0	0	10
	L	1	0	9
20	S	0	4	8
	M	1	2	8
	L	1	4	5
25	S	6	3	1
	M	1	7	2
	L	3	6	1
40	S	7	3	0
	M	7	2	1
	L	6	3	1
50	S	1	8	1
	M	3	5	2
	L	2	4	4
100	S	7	10	0
	M	9	10	0
	L	10	10	0
Total		124	141	122

three sets of constraints are removed will be the same as the solution of the original model, based on the following reasons. From constraints (3.26), it follows that constraints (3.23) can be removed because the relation between two tasks of the same job is modelled in constraints (3.26). Constraints (3.25) can also be removed because the relative order of each pair of tasks is established in constraints (3.24), enforcing therefore a similar order for any triple distinct tasks. Finally, we can remove constraints (3.27) since they only enforce an upper bound on starting time of the completion tasks.

4.3.2 Introducing new constraints

We propose two sets of constraints to be added to model S3. The first set of constraints is based on an implicit relationship between the tasks of every pair of jobs. It follows that if the initial task of job j precedes the initial task of job j', i.e., if $x_{2j-1,2j'-1} = 1$, the initial task of job j also precedes the completion task of job j'. Therefore, we may set $x_{2j-1,2j'} = 1$. Constraints (4.24) show that relationship.

$$x_{2j-1,2j'-1} \leq x_{2j-1,2j'}, \quad j, j' \in N. \tag{4.24}$$

Table 4.10 : The gap (in %) from the best solution.

n	Job category	$bb_{1/2}$	$bb_{3/4}$	Gurobi
5	S	0.0	0.0	0.0
	M	0.0	0.0	0.0
	L	0.0	0.0	0.0
10	S	0.0	0.0	0.0
	M	0.0	0.0	0.0
	L	0.0	0.0	0.0
15	S	1.8	1.6	0.0
	M	2.6	1.5	0.0
	L	2.2	1.7	0.0
20	S	3.0	1.0	0.2
	M	3.3	1.1	0.2
	L	3.1	0.6	0.5
25	S	0.8	0.5	2.1
	M	1.8	0.4	1.8
	L	1.5	0.8	2.1
40	S	1.2	3.1	37.4
	M	2.5	5.9	28.3
	L	3.9	3.5	30.8
50	S	5.6	0.8	7.8
	M	5.5	1.7	43.4
	L	4.4	5.8	23.2
100	S	3.5	0.0	-
	M	1.3	0.0	-
	L	0.0	0.0	-
Average		2.0	1.25	7.48

The second set of constraints follows from the following observation. If two jobs cannot be interleaved, and nor can be nested, then one of the jobs completes before the other job is started.

Consider two jobs j and j' that cannot be nested into each other (i.e., $L_j < p_{2j'-1} + L_{j'} + p_{2j'}$ and $L_{j'} < p_{2j-1} + L_j + p_{2j}$), neither can job j be interleaved in job j', and nor job j' can be interleaved in job j, i.e., $p_{2j-1} > L_{j'}$ or $p_{2j'} > L_j$, and $p_{2j'-1} > L_j$ or $p_{2j} > L_{j'}$. Then, either both tasks of job j precede both tasks of job j', i.e., $x_{2j-1,2j'-1} = x_{2j,2j'-1} = x_{2j-1,2j'} = x_{2j,2j'} = 1$, or both tasks of job j' precede both tasks of job j, i.e., $x_{2j-1,2j'-1} = x_{2j,2j'-1} = x_{2j-1,2j'} = x_{2j,2j'} = 0$. Such relationships can be modelled by constraints (4.25):

$$x_{2j-1,2j'-1} = x_{2j,2j'-1} = x_{2j-1,2j'} = x_{2j,2j'}, \quad j, j' \in N, \tag{4.25}$$

if $(L_j < p_{2j'-1} + L_{j'} + p_{2j'})$, and $(L_{j'} < p_{2j-1} + L_j + p_{2j})$, and $(p_{2j-1} > L_{j'}$ or $p_{2j'} > L_j)$, and $(p_{2j'-1} > L_j$ or $p_{2j} > L_{j'})$.

We note that the two sets of new constraints are valid inequalities as tested over

Table 4.11 : An overview of the outcomes of the binary search heuristic and Gurobi.

Criterion	$bb_{1/2}$	$bb_{3/4}$	Gurobi
Feas	**240**	**240**	197
Opt	59	59	**62**
Best	124	**141**	122
Gap (in %)	2.00	**1.25**	7.48

various instances.

Next, we conduct a full experimental design to assess the effectiveness of various combinations of constraints removal and addition.

4.3.3 The enhanced mixed-integer program

As we discussed in Sections 4.3.1 and 4.3.2, a number of constraints can be removed from model S3, and some constraints may be added to model S3 so that the computational performance of model S3 may be improved. It is well-known that removing or adding redundant constraints may lead to computational advantages. For example, Naeni and Salehipour (2019) showed that certain redundant constraints significantly improve performance of the MIPs for the traveling repairman problem. Therefore, we aim to determine the best combination of constraints to be removed from, and to be added to model S3.

For that purpose, we design a full factorial experiment, in which there are five factors corresponding to constraints (3.23), (3.25), (3.27), (4.24), and (4.25), and each factor has two levels of being included or not. The number of all combinations is equal to $2^5 = 32$, resulting therefore in 32 different MIPs for the problem. We choose 48 instances from the "general set" benchmark instances. That benchmark comprises of 240 instances in 24 categories of different sizes, where there are 10 instances per category. We select two instances from each category. That results in a total of 48 instances to be solved by 32 MIPs, meaning that we test 1,536 settings in total, i.e., solving 48 instances each by 32 models. Because the original model S3 (before removing or adding constraints) is not capable of obtaining even feasible solutions for large instances, we therefore utilise the binary search heuristic to guide the optimisation process and to solve the instances. We set the time limit to 900 seconds for solving each setting, resulting in a total computation time of 16 days.

We perform a statistical analysis of the results of experiments with Minitab (Minitab Statistical Software, 2020). At the significance level of 5%, the analysis indicates that constraints (3.25) and (4.24), and the interaction of constraints (3.25) and (4.24) significantly affect performance of the model, with the p-values equal to 0.000, 0.014, and 0.022, respectively. We present the main effects and interactions plots for constraints (3.25) and (4.24) in Figures 4.7 and 4.8, respectively. In the figures, level 1 stands for including the constraints in the model, and level 0 means that the constraints are not included in

the model. From the results, we can observe that removing constraints (3.25) from, and adding constraints (4.24) to the model considerably improve the performance.

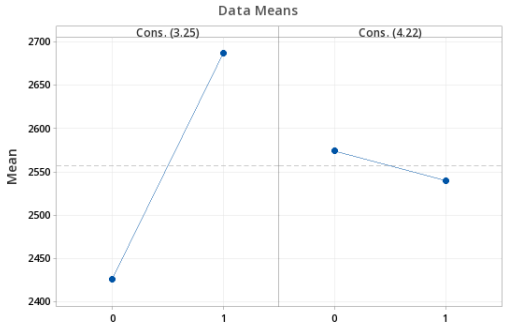

Figure 4.7 : The main effects plot for constraints (3.25) and (4.24).

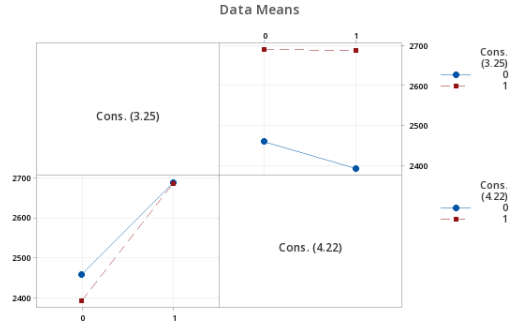

Figure 4.8 : The interactions plot for constraints (3.25) and (4.24).

At the 5% significance level, we observe that there is no significant impact for adding or removing constraints (3.23), (3.27), and (4.25), because the p-values for the related tests are equal to 0.996, 0.779, and 0.743. We do not include those three constraints in the model for the sake of having less number of constraints.

Based on the full factorial experiment and our statistical analysis, we present the improved formulation for the problem of minimising the makespan on the single-machine CTSP as model S5 in the following.

Model S5

$$z = \min\ C_{\max} \tag{4.26}$$

subject to

(3.21), (3.22), (3.24), (3.26), (3.28)-(3.29), (4.24).

4.4 The relax-and-solve algorithm

In this section, we propose a relax-and-solve (R&S) matheuristic algorithm for solving problem $1|(a_j, L_j, b_j)|C_{\max}$. The R&S has been shown to deliver quality solutions for challenging scheduling problems (Ahmadian et al., 2020; Ahmadian et al., 2021). The operation of the R&S algorithm is similar to that of the fix-and-optimize algorithm of Helber and Sahling (2010). The fix-and-optimize algorithm is utilized to solve many optimization problems, including variations of the timetabling problem (Dorneles et al., 2014), the shortest path problem (Carvalho et al., 2021) and inventory routing problem (Friske et al., 2022).

The general idea of the R&S includes destructing the execution order of a small number of consecutive tasks of jobs, and constructing a new order for executing the tasks. Starting from an initial feasible sequence of tasks, the destruction is performed through relaxing certain constraints or variables in mathematical program of the problem, while the construction is performed by solving a new instance of the mathematical program. The earlier work by Salehipour (2020) refers to those operations as "relax" and "solve".

Our proposed R&S algorithm for problem $1|(a_j, L_j, b_j)|C_{\max}$ (see Crefch4.alg.rs) has certain distinguishing characteristics. First, the proposed R&S uses a pre-processing procedure before the relax and solve operations, that spends t_0 seconds to improve an initial sequence. Second, it benefits from the exact delays between the tasks of jobs, which results in not all $2n!$ sequences of the tasks being feasible. Therefore, both in the local search algorithm and through solving model S5, only feasible sequences are explored. Third, unlike the original R&S, we relax two subsets of tasks, where each relax iteration is followed by a solve operation for t_r seconds, and the whole relax and solve operation is iterated for $t > t_r$ seconds. The details of the components of Algorithm 5 is as follows.

Algorithm 5: The R&S algorithm for problem $1|(a_j, L_j, b_j)|C_{\max}$.

Input: The initial sequence $\sigma_0 = (1, \ldots, 2n)$ with makespan C_{σ_0}.
Output: A sequence σ with makespan C_σ.
Pre-processing:
$\sigma_1, C_{\sigma_1} \leftarrow$ Run Algorithm 4;
$\sigma_2, C_{\sigma_2} \leftarrow$ Given σ_1, solve model S5 by an exact solver;
$\sigma \leftarrow$ The best sequence between σ_1 and σ_2;
while *the stopping condition is not met* **do**
    Select $\sigma', \sigma'' \subset \sigma$ to relax;
    $\sigma_{tmp} \leftarrow$ Given σ with relaxed σ', σ'', solve model S5 by an exact solver;
    $\sigma \leftarrow$ Update the best sequence;
end
return σ *with the makespan value of* C_σ*;*

4.4.1 Solution representation

A feasible solution is represented by a sequence of the tasks in positions 1 to $2n$, where $H_{(i)}$ represents the task in position i in the sequence as shown in Figure 4.9.

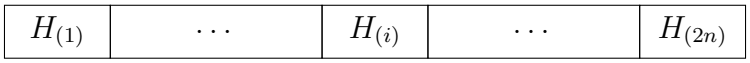

Figure 4.9 : The solution representation in the proposed R&S algorithm.

4.4.2 The initial sequence

The initial sequence $\sigma_0 = (1, \ldots, 2n)$ is constructed in the same way to that of Algorithm 4, i.e., by appending the jobs one by one from 1 to n. Each pair of tasks $(H_{(2j-1)}, H_{(2j)}) \in \sigma_0$ represents the two tasks of job j.

4.4.3 Pre-processing

The pre-processing aims to improve the initial sequence σ_0 in two steps. In the first step, σ_0 is passed into the local search (see Algorithm 4), where interleaving and nesting moves are applied to σ_0. We let σ_1 denote the sequence obtained by the local search with the makespan value of C_{σ_1}.

In the second step, we further improve σ_1 by using an optimisation solver. For that, model S5 is solved with a time-limit of t_0 seconds. We warm-start the optimisation process by providing σ_1 as a starting point. Let σ_2 denote the resulting sequence with C_{σ_2} as its value of makespan.

4.4.4 Relax and solve operations

The main operations of the R&S algorithm are the relax and solve where a series of "relaxed" problems are solved. Given sequence σ from the pre-processing, a relaxed problem is generated by destructing the execution order of two subsets of tasks. Let $\sigma', \sigma'' \subset \sigma$ denote those two subsets. In model S5, that is equivalent to considering binary variables $x_{hh'}, h, h' \in \sigma' \cup \sigma''$, to be decided by model S5, for representing those tasks, and setting value of the remaining binary variables, i.e., $x_{hh'}, h, h' \notin \sigma' \cup \sigma''$ according to the relative position of h and h' in σ. Therefore, we treat those variables as parameters. That implies that the non-relaxed tasks are not allowed to change their position in σ. By solving model S5, the binary variables for the relaxed tasks will be decided upon, resulting therefore in a re-constructed sequence of executing the tasks.

Note that we keep the size of subsets σ' and σ'' small, so that a relaxed problem includes only a small number of binary variables, and as such the relaxed problem may be solved in a reasonable amount of time. Each relaxed problem is solved by the solver for t_r seconds, and the whole relax and solve procedure is run for a total time of t seconds. We denote the best available sequence as σ with the makespan value of C_σ.

4.5 Computational experiments

Extensive computational experiments are performed to evaluate the performance of the new mathematical model (model S5), and that of the two matheuristic solution methods, i.e., the binary-search algorithm of Section 4.2, and the R&S algorithm of Section 4.4. We test the model and the algorithms on the 240 instances denoted as "general set" in Section 3.1.2.

We limit the run time of the binary-search algorithm and the stand-alone Gurobi to 3,600 seconds. Regarding the value of parameters of the R&S algorithm, we set the time-limit t_0 to 100 seconds, and we set the time-limit t such that the total time elapsed for each instance is 3,600 seconds. Because the first step in the pre-processing (see Section 4.4.3) is almost instantaneously performed for any instance, the value of t is therefore almost equal to 3,500 seconds. We set the value of time-limit t_r to 20 seconds. In addition to those three time-limits, we need to set size of the relaxing subsets σ' and σ'' in the R&S algorithm. We choose an identical value for size of σ' and σ'', which is proportionate to the number of jobs in the instance. For small to medium-sized instances with $n \leq 25$, we set the size of σ' and σ'' to be equal to 5, and for large instances with $n > 25$, we set it to 10. We choose a larger value for larger instances because we observe, during a preliminary test, that having larger chunks in larger instances leads to higher quality solutions, compared to having smaller chunks. We, however, do not choose values greater than 10 because Gurobi cannot solve the test instances with 15 jobs and more within the time limit.

We utilise Gurobi version 8.0.0 (Gurobi Optimization, 2018), as the solver within the binary search and R&S algorithms. We also solve the same 240 instances with the solver Gurobi, through optimising model S5. We implement the models and algorithms in the programming language Python version 3.6. We perform all computational experiments on the same PC mentioned in Section 3.3, and we utilise four processors.

We solve the 240 benchmark instances by the following five methods: (1) model S3 by the solver Gurobi, (2) model S5 by the solver Gurobi, (3) the binary search algorithm incorporated with model S3, (4) the binary search algorithm incorporated with model S5, and (5) the R&S algorithm. For both binary search algorithms we set the bound $bb = \frac{LB+3UB}{4}$ (i.e., it is equal to the three fourth of the interval). We also use the "call-back" functionality of the Gurobi solver to stop the solver as soon as a feasible solution is found (this can be seen as a computation time reduction strategy). We also alter the parameter "MIPfocus" of Gurobi, focusing therefore on obtaining a feasible solution, than e.g., on obtaining an optimal solution.

Table 4.12 reports outcomes of the first two solution methods, i.e., optimising models S3 and S5 by the solver Gurobi, which we denote by MIP_S3 and MIP_S5, respectively. From the table, we observe that model S5 obtains higher quality solutions than model S3,

meaning that it has a superior performance to model S3. For example, model S5 obtains the best solutions for additional 30 instances. The gap of solutions obtained by model S5 is as low as 3.3%, which is in the order of five times smaller than that of model S3. While model S5 obtains a smaller number of feasible solutions than model S3, it delivers an optimal solution for more instances than model S3. We believe that the criteria feas and opt are less important in the context of models S3 and S5 because both problems are only capable of generating competitive solutions for instances with up to 25 jobs.

Table 4.12 : Computational results for methods MIP_S5 and MIP_S3.

n	Job category	Gap		Best		Feas		Opt	
		MIP_S5	MIP_S3	MIP_S5	MIP_S3	MIP_S5	MIP_S3	MIP_S5	MIP_S3
5	S	0.0	0.0	10	10	10	10	10	10
	M	0.0	0.0	10	10	10	10	10	10
	L	0.0	0.0	10	10	10	10	10	10
10	S	0.0	0.0	10	10	10	10	10	10
	M	0.0	0.0	10	10	10	10	10	10
	L	0.0	0.0	10	10	10	10	10	10
15	S	0.3	0.2	6	6	10	10	2	1
	M	0.3	0.4	4	5	10	10	0	0
	L	0.2	0.4	8	4	10	10	2	1
20	S	0.2	0.8	4	2	10	10	0	0
	M	0.1	1.0	7	1	10	10	0	0
	L	0.2	1.1	7	1	10	10	0	0
25	S	0.8	4.0	3	0	10	10	0	0
	M	0.7	3.4	4	0	10	10	0	0
	L	0.7	4.5	6	0	10	10	0	0
40	S	9.9	52.0	0	0	6	9	0	0
	M	10.3	43.5	0	0	9	10	0	0
	L	7.3	49.9	0	0	7	10	0	0
50	S	17.2	43.7	0	0	2	4	0	0
	M	10.8	86.8	0	0	2	6	0	0
	L	9.8	55.9	0	0	1	8	0	0
100	S	-	-	0	0	0	0	0	0
	M	-	-	0	0	0	0	0	0
	L	-	-	0	0	0	0	0	0
Total/average		3.3	16.6	109	79	177	197	64	62

Next, we compare the performance of binary search algorithms. We denote by BS_S3 and BS_S5 the binary search algorithms incorporated with models S3 and S5. We show the outcomes of BS_S3 and BS_S5 in Table 4.13. Recall that the only difference between BS_S3 and BS_S5 is the implemented mathematical model. The results of Table 4.13 further indicate the superiority of model S5 to model S3 because, for example, the gap of 3.9% for BS_S5 is in the order of almost four times less than that of BS_S3, which is 16.6%. Also, BS_S5 obtains 11 more best solutions.

Table 4.13 : Computational results for methods BS_S5 and BS_S3.

n	Job category	Gap		Best		Feas		Opt	
		BS_S5	BS_S3	BS_S5	BS_S3	BS_S5	BS_S3	BS_S5	BS_S3
5	S	0.0	0.0	10	10	10	10	10	10
	M	0.0	0.0	10	10	10	10	10	10
	L	0.0	0.0	10	10	10	10	10	10
10	S	0.0	0.0	10	10	10	10	10	10
	M	0.0	0.0	10	10	10	10	10	10
	L	0.0	0.0	10	10	10	10	10	10
15	S	0.9	1.8	2	0	10	10	0	0
	M	1.6	1.9	0	0	10	10	0	0
	L	1.1	2.0	0	0	10	10	0	0
20	S	0.9	1.6	3	1	10	10	0	0
	M	1.5	1.9	1	0	10	10	0	0
	L	0.8	1.3	2	1	10	10	0	0
25	S	1.5	2.4	0	0	10	10	0	0
	M	1.1	1.9	3	0	10	10	0	0
	L	1.4	3.2	2	0	10	10	0	0
40	S	5.2	14.4	0	0	10	10	0	0
	M	5.6	17.8	0	0	10	10	0	0
	L	5.8	18.4	0	0	10	10	0	0
50	S	5.8	31.3	0	0	10	10	0	0
	M	5.6	32.2	0	0	10	10	0	0
	L	6.3	35.1	0	0	10	10	0	0
100	S	14.6	58.4	0	0	10	10	0	0
	M	15.0	66.7	0	0	10	10	0	0
	L	19.5	67.4	0	0	10	10	0	0
Total/average		3.9	15.0	73	62	240	240	60	60

Tables 4.12 and 4.13 show that our proposed formulation S5 is more effective, than the existing formulation S3, in obtaining better solutions for $1|(a_j, L_j, b_j)|C_{\max}$.

Table 4.14 presents the computational results obtained by the R&S method. Recall that we utilise model S5 in the R&S matheuristic (see Section 4.4). The gap of the R&S method is as low as 0.6%, and it delivers the best known solution for 164 instances. In addition, the R&S method obtains the best solution for all instances with 40 or more jobs. The results indicate that the R&S method performs the best among the five tested methods.

To further illustrate the quality of solutions obtained by the R&S method, we report "gap from LB (in %)" for the solutions reported by model S5, and the BS_S5 and R&S matheuristics in Table 4.15. That criterion is calculated as $\frac{z-LB}{LB} \times 100$, where LB is the best lower bound obtained for the instance through the procedure explained in Section 4.2.1. As the table shows, the gap from LB for the solutions obtained by model S5 and BS_S5 increases as the number of jobs increase. On the other hand, for the R&S method, the values of gap are almost stable over instances with different number of jobs.

Table 4.14 : Computational results for R&S.

n	Job category	Gap	Best	Feas	Opt
5	S	0.0	10	10	10
	M	0.0	10	10	10
	L	0.0	10	10	10
10	S	0.0	10	10	10
	M	0.0	10	10	10
	L	0.0	10	10	10
15	S	1.7	0	10	0
	M	2.1	1	10	0
	L	1.8	0	10	0
20	S	2.0	0	10	0
	M	2.3	1	10	0
	L	2.2	0	10	0
25	S	0.5	7	10	0
	M	0.7	3	10	0
	L	0.9	2	10	0
40	S	0.0	10	10	0
	M	0.0	10	10	0
	L	0.0	10	10	0
50	S	0.0	10	10	0
	M	0.0	10	10	0
	L	0.0	10	10	0
100	S	0.0	10	10	0
	M	0.0	10	10	0
	L	0.0	10	10	0
Total/average		0.6	164	240	60

That suggests the stability of the R&S method for solving larger instances, and confirms effectiveness of the R&S for solving various instances.

We also investigate the effectiveness of both steps of the pre-processing procedure that we proposed for the R&S matheuristic. We solve each instance by the R&S method, and record the value of makespan for the initial sequence, the sequences obtained in each step of the pre-processing, and for the final sequence. We let C_{σ_0}, C_{σ_1}, C_{σ_2} and C_{σ} denote those values of the makespan. For each instance, we calculate percentage of improvement gained in each step, out of total improvement of $TI = C_{\sigma_0} - C_{\sigma}$. Those can be calculated as $\frac{C_{\sigma_0} - C_{\sigma_1}}{TI} \times 100$, $\frac{C_{\sigma_1} - C_{\sigma_2}}{TI} \times 100$, and $\frac{C_{\sigma_2} - C_{\sigma}}{TI} \times 100$. The average improvements for different job categories are shown in Table 4.16. We do not report the percentage of improvement for instances with 5 and 10 jobs because all of those instances are optimally solved almost instantly in the second step of the pre-processing.

According to Table 4.16, the contribution of the constructive heuristic (for generating initial sequences) to the total improvement is significant, and varies from 60.9% to 66.8%. For the instances with $n = 15, 20, 25$, the majority of the remaining improvements is obtained in the second step of the pre-processing. The main reason for that includes the

Table 4.15 : Gap from the lower bound for MIP_S5, BS_S5, and R&S.

n	Job category	MIP_S5	BS_S5	R&S
5	S	10.2	10.2	10.2
	M	10.2	10.2	10.2
	L	10.1	10.1	10.1
10	S	5.0	5.0	5.0
	M	5.6	5.6	5.6
	L	6.0	6.0	6.0
15	S	4.0	3.4	4.9
	M	5.5	4.3	6.1
	L	5.2	4.3	6.0
20	S	5.9	5.2	7.2
	M	7.1	5.6	7.9
	L	6.8	6.1	8.2
25	S	7.4	6.6	6.3
	M	7.8	7.4	7.5
	L	8.1	7.4	7.6
40	S	10.5	15.8	5.0
	M	11.1	16.0	5.2
	L	11.8	13.5	5.7
50	S	11.1	23.1	5.0
	M	11.4	16.5	5.4
	L	12.8	15.8	6.1
100	S	22.4	-	6.8
	M	24.3	-	8.1
	L	29.1	-	8.0
Total/average		10.40	9.44	6.84

size of model S5 generated for those instances, which is small enough, allowing therefore the solver Gurobi to improve the solution to a great extent within 100 seconds. As the size of instances increases to $n = 40, 50, 100$, the majority of the improvements is due to the main body of the R&S algorithm. This further suggests the effectiveness of the proposed R&S algorithm for large instances.

Table 4.16 : Improvement (in % of total) gained in the three steps of the R&S algorithm.

n	Job category	Step 1 of pre-processing	Step 2 of pre-processing	Main body of R&S
15	S	61.2	38.6	0.1
	M	60.9	39.0	0.0
	L	63.7	36.0	0.2
20	S	64.2	34.7	1.2
	M	63.8	35.9	0.4
	L	64.6	34.3	1.1
25	S	64.6	27.4	7.9
	M	63.8	32.3	3.9
	L	63.7	31.8	4.5
40	S	64.2	6.9	28.9
	M	64.3	5.9	29.8
	L	65.1	4.7	30.1
50	S	64.6	4.8	30.6
	M	65.1	4.0	30.9
	L	64.8	4.8	30.4
100	S	66.8	1.4	31.8
	M	66.0	1.1	32.9
	L	66.3	3.6	30.1

Chapter 5

Coupled tasks on parallel machines

In this chapter, we introduce the problem of scheduling a set of coupled-task jobs on parallel identical machines with the objective function of minimising the makespan. In the parallel-machine variant of the coupled task scheduling problem (CTSP), we seek a job allocation to machines as well as a job schedule on each machine such that the makespan is minimised. To the best of our knowledge, this is the first attempt to address the CTSP with parallel identical machines and the makespan criterion. The motivation for this setting is that in the healthcare applications of the CTSP (see Section 2.2), there are typically a number of identical pieces of equipment with the same functionality. This is the major motivation for us to study the CTSP with parallel identical machines. The problem is denoted by $Pm|(a_j, L_j, b_j)|C_{\max}$. As a first step in studying the problem, we provide a full overview of the computational complexity for $Pm|(a_j, L_j, b_j)|C_{\max}$. In Section 5.1 we formally define the problem of scheduling the coupled-tasks on parallel identical machines. We then prove that the majority of these problems are (strongly) NP-hard. An important result of our work discussed in Section 5.2 includes showing that the existence of a $(2-\varepsilon)$-approximation algorithm for the problem implies $P = NP$. The latter result improves a recently proposed bound for the open-shop counterpart as well. In Section 5.3, we present polynomially solvable cases of the problem and develop efficient algorithms for minimising the makespan. The results presented in this chapter are submitted for possible publication as:

- Khatami, M., Oron, D., and Salehipour, A. (2021a). "Scheduling coupled tasks on parallel identical machines". *Submitted to Annals of Operations Research*.

5.1 NP-hardness proof

Given the set $J = \{1, \ldots, n\}$ of n coupled-task jobs and the set $M = \{1, \ldots, m\}$ of parallel identical machines, problem $Pm|(a_j, L_j, b_j)|C_{\max}$ consists of scheduling the coupled-task jobs on the machines such that the makespan is minimised. Each task can be performed on any machine and there is an exact delay period between the two tasks of each job. The triplet (a_j, L_j, b_j) represents a job $j \in J$, where a_j, L_j and b_j are positive integer parameters. Each machine can perform at most one task at a time and preemption of tasks is not allowed, i.e., once the operation of a task is started it must be completed with no interruption. The tasks of other jobs, however, can be processed during the delay period.

As discussed in Section 2.1.1, the computational complexity of the single machine CTSP under the objective of minimising the makespan was extensively studied in Orman and Potts (1997). In general, they showed that $1|(a_j, L_j, b_j)|C_{\max}$ is strongly NP-hard. They also showed that $1|(a_j = L_j = b_j)|C_{\max}$ is strongly NP-hard, implying therefore that problems $1|(a_j = b_j, L_j)|C_{\max}$, $1|(a_j, L_j = b_j)|C_{\max}$ and $1|(a_j = L_j, b_j)|C_{\max}$ are also strongly NP-hard. From those results it follows that even if any two parameters of a_j, L_j and b_j, out of three, are identical for all jobs, the problem remains strongly NP-hard. Hence, we conclude that problems $1|(a_j, L, b)|C_{\max}$, $1|(a, L_j, b)|C_{\max}$, $1|(a, L, b_j)|C_{\max}$, $1|(a_j, L_j, b)|C_{\max}$, $1|(a, L_j, b_j)|C_{\max}$ and $1|(a_j, L, b_j)|C_{\max}$ are strongly NP-hard as well. Even the problem $1|(p, L_j, p)|C_{\max}$ was shown to be strongly NP-hard, where p is a positive constant (the initial and completion tasks of all jobs are identical). From these results, we conclude that the associated problems in the parallel identical machine setting are also strongly NP-hard, more precisely:

Theorem 5. *The following parallel-machine CTSPs are strongly NP-hard:*

- $Pm|(a_j = L_j = b_j)|C_{\max}$*, resulting in the strong NP-hardness of cases* $Pm|(a_j = b_j, L_j)|C_{\max}$*,* $Pm|(a_j, L_j = b_j)|C_{\max}$ *and* $Pm|(a_j = L_j, b_j)|C_{\max}$*;*
- $Pm|(a_j, L, b)|C_{\max}$*, and also* $Pm|(a, L_j, b)|C_{\max}$ *and* $Pm|(a, L, b_j)|C_{\max}$*, resulting in the strong NP-hardness of* $Pm|(a_j, L_j, b)|C_{\max}$*,* $Pm|(a, L_j, b_j)|C_{\max}$*,* $Pm|(a_j, L, b_j)|C_{\max}$*; and,*
- $Pm|(p, L_j, p)|C_{\max}$*.*

Orman and Potts (1997), proposed an optimal $O(n)$-time algorithm for $1|(p, p, b_j)|C_{\max}$, i.e., when the delays are equal to the initial tasks across all jobs. From here, it follows that their algorithm is optimal for $1|(a_j, p, p)|C_{\max}$ as well. We observe that the parallel-machine counterparts of both problems are not easily solved. We show in the following that $Pm|(p, p, b_j)|C_{\max}$ is NP-hard even for the two-machine setting, as is $Pm|(a_j, p, p)|C_{\max}$. We use the equal cardinality partition (ECP) problem for the NP-hardness proof.

The equal cardinality partition problem (ECP): Given a set $W = \{w_1, \ldots, w_q\}$ of q elements where each element i has an integer size w_i and $\sum_{j \in W} w_j = 2B$, does there exist two disjoint subsets W_1 and W_2 with $|W_1| = |W_2| = \frac{q}{2}$, such that $\sum_{j \in W_1} w_j = \sum_{j \in W_2} w_j = B$?

Given an instance of ECP, we construct an instance of the parallel identical machines CTSP with two machines, i.e., $P2|(p, p, b_j)|C_{\max}$, as following.

The two-machine CTSP (CTP1): We construct an instance of the decision version of the two-machine CTSP with the set of jobs $J = \{1, \ldots, 2q\}$, where we distinguish the following two subsets of jobs:

- $a_j = L_j = p, b_j = w_j, w_j \in W, 1 \leq j \leq q$,
- $a_j = L_j = b_j = p, q+1 \leq j \leq 2q$,

where $p > \frac{B}{3}$. We denote jobs 1 to q as "partition jobs" and jobs $q+1$ to $2q$ as "identical jobs".

Lemma 4. *If ECP has a solution, so does the constructed instance of CTP1 such that* $C^*_{max} \leq \frac{3qp}{2} + B$.

Proof. Assume that ECP has a solution. We construct a feasible schedule for CTP1 as follows. There are $\frac{q}{2}$ interleaving pairs of jobs on each machine. The first job of each interleaving pair is one of the identical jobs, and the second job of each interleaving pair on the first (second) machine is from the set W_1 (W_2). The processing time of each interleaving pair is therefore equal to $3p + w_j$ (see Figure 5.1), where w_j is the processing time of the completion task of the second job of the interleaving pair. The total processing times on the first and second machines are equal to $C^*_1 = \sum_{j \in W_1}(3p + w_j) = \frac{3qp}{2} + B$ and $C^*_2 = \sum_{j \in W_2}(3p + w_j) = \frac{3qp}{2} + B$.

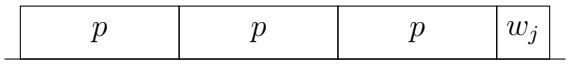

Figure 5.1 : An interleaving pair in the constructed schedule for CTP1 instance.

□

Lemma 5. *If CTP1 has a solution, ECP should have a solution as well.*

Proof. First, observe that there is no idle time in the schedule of Figure 5.1. Hence, any optimal solution for CTP1 should have no idle time as well. It is easy to observe that this can only be achieved if there are q interleaving pairs with no idle time between them. Also, we note that an interleaving pair has no idle time only if one of the identical jobs is the first job of the pair, i.e., all identical jobs should be scheduled as the first jobs of the interleaving pairs. Second, there should be an equal number of interleaving pairs on both machines, i.e., $\frac{q}{2}$, since otherwise the additional interleaving pair(s) will have a processing time of at least $3p$, and hence, $C_1 > \frac{3qp}{2} + 3p > \frac{3qp}{2} + B > C^*_1$. That completes the proof. □

Based on Lemmas 4 and 5, we have:

Theorem 6. *Problem* $P2|(p,p,b_j)|C_{\max}$ *is at least* NP*-hard in the ordinary sense.*

The NP-harness result for the two-machine problem can be easily generalised to the strong NP-hardness of the m-machine problem. For that purpose, we use the 3-partition problem (3-PP), as following:

3-PP: Given a set $W = \{w_1, \ldots, w_{3m}\}$ of $3m$ elements where each element i has an integer size w_i and, and a positive integer B, where $\frac{B}{4} < w_j < \frac{B}{2}$, for $j = 1, \ldots, 3m$, and $\sum_{j=1}^{3m} w_j = mB$, does there exist m disjoint subsets $W_1, \ldots, W_m$ such that $\sum_{j \in W_i} w_j = B$ and $|W_i| = 3$, for $i = 1, \ldots, m$ (there are exactly three elements in each subset W_i whose sum is exactly B)?
Given an instance of the 3-PP, we construct an instance of $Pm|(p,p,b_j)|C_{\max}$ as following.

The m-machine CTSP (CTP2): We construct an instance of the decision version of the m-machine CTSP with a set $J = \{1, \ldots, 6m\}$ of jobs, where we distinguish the following two subsets of jobs:

- $a_j = L_j = p, b_j = w_j, w_j \in W, 1 \leq j \leq 3m$;
- $a_j = L_j = b_j = p, 3m + 1 \leq j \leq 6m$;

where $p > \frac{B}{3}$. Similar to the two-machine variant we denote jobs 1 to $3m$ as "partition jobs" and jobs $3m + 1$ to $6m$ as "identical jobs".

Lemma 6. *If 3-PP has a solution, so does the constructed instance of CTP2 such that a feasible schedule has* $C^*_{max} \leq 9p + B$.

Proof. Assume that 3-PP has a solution. We construct a feasible schedule for CTP2 as follows. There are 3 interleaving pairs of jobs on each machine, where the interleaving pairs are constructed similar to the two-machine variant (see Figure 5.1). We note that the second job of each interleaving pair on machine $i \in M$ is from the set W_i. As a result, the total processing time on machine $i \in M$ is equal to $C^*_i = \sum_{j \in W_i}(3p + w_j)$. Because we assumed that 3-PP has a solution, we have therefore $C^*_i = 9p + B, i = 1, \ldots, m$, as desired. □

Lemma 7. *If CTP2 has a solution, so does 3-PP.*

Proof. The proof is similar to the two-machine case: In order to ensure that there is no idle time in the schedule, there should be $3m$ interleaving pairs such that there are exactly 3 interleaving pairs on each machine $i \in M$. □

Based on Lemmas 6 and 7 we have Theorem 7:

Theorem 7. *Problem* $Pm|(p,p,b_j)|C_{\max}$ *is strongly* NP*-hard.*

5.2 Approximation results

In this section we demonstrate that the existence of a $(2-\varepsilon)$-approximation algorithm for $P2|(p,p,b_j)|C_{\max}$ implies $P = NP$. We utilize the well-known partition problem for the proof.

The partition problem (PP): Given a set $W = \{w_1, \ldots, w_q\}$ of q elements where each element i has an integer size w_i and where $\sum_{j=1}^{q} w_j = 2B$, the question is whether there exist two disjoint subsets W_1 and W_2 such that $\sum_{w_j \in W_1} w_j = \sum_{w_j \in W_2} w_j = B$?

Given an instance of the partition problem, we construct an instance of the problem $P2|(a_j, L_j, b_j)|C_{\max}$ as following.

The two-machine CTSP (CTP3): Given a set $J = \{1, \ldots, q+3\}$ of jobs, we set the tasks and delays of the jobs as:

- $a_j = b_j = w_j, L_j = Y + B - w_j, w_j \in W, 1 \leq j \leq q$,
- $a_{q+1} = L_{q+1} = B, b_{q+1} = Y$,
- $a_{q+2} = Y, L_{q+2} = b_{q+2} = B$,
- $a_{q+3} = b_{q+3} = B, L_{q+3} = Y + 2B$,

where $Y > 2B$.

Lemma 8. *If PP has a solution, there exists a feasible schedule for CTP3 with* $C^*_{max} \leq Y + 4B$.

Proof. Let us assume that PP has a solution. We construct a feasible schedule for CTP3 as depicted in Figure 5.2. Assume that the starting time of the schedule is time 0. Let the initial (completion) task of job $q+1$, i.e., a_{q+1} (b_{q+1}) be processed on machine M_1 (M_2), where a_{q+1} starts at time 0. Similarly, let the initial (completion) task of job $q+2$, i.e., a_{q+2} (b_{q+2}) be performed on machine M_1 (M_2), where a_{n+2} starts at time $2B$. The initial task of job $q+3$ starts at time 0 on M_2, and its completion task finishes at time $Y + 4B$ on machine M_1. It remains to schedule jobs 1 to q. We assume that jobs having $a_j = b_j = w_j \in W_1$, denoted as W_1-jobs, are processed during the two idle times on M_1, and jobs having $a_j = b_j = w_j \in W_2$, denoted as W_2-jobs, are performed during the two idle times on M_2. We note that the initial and completion tasks of W_1-jobs (W_2-jobs) are processed on M_1 (M_2) in the same order. Therefore, the starting time of the initial task of job $j \in W_1$ is $B + \sum_{k=1}^{j-1} w_k$. As a result, the starting time of its completion task will be the time $(B + \sum_{k=1}^{j-1} w_k + w_j) + (Y + B - w_j) = Y + 2B + \sum_{k=1}^{j-1} w_k$. It is easy to verify that all those jobs will be scheduled with no overlap in their tasks.

□

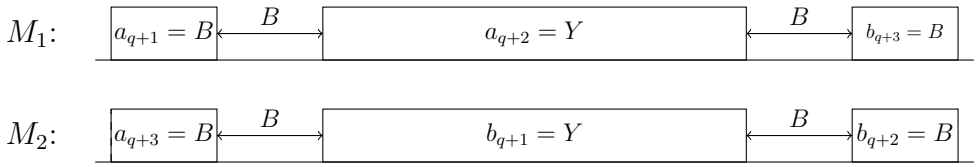

Figure 5.2 : The constructed schedule for CTP3 instance.

Lemma 9. *If PP has no solution, then neither does CTP3.*

Proof. We can consider the following two cases where PP has no solution: (1) $\sum_{w_j \in W_1} w_j > B$ and (2) $\sum_{w_j \in W_1} w_j < B$. In either of the cases, there will be some jobs that their tasks cannot be finished during the idle times of the machines, and hence, either they should finish before the constructed schedule starts, or they should start after the constructed schedule finishes. □

Theorem 8 follows from Lemmas 8 and 9:

Theorem 8. *The existence of a* $(2-\varepsilon)$*-approximation algorithm for the problem* $P2|(a_j, L_j, b_j)|C_{\max}$ *implies* $P = NP$.

Proof. Let $C^*_{\max}$ denote the optimal makespan and $C^\sigma_{\max}$ be that of an arbitrary schedule σ. As we note in Lemma 7, if PP has no solution, the schedule will consist of some jobs from W_1 or W_2 that are completely processed before or after the constructed schedule. This means that $C^\sigma_{\max} > Y + 4B$. As such, $C^\sigma_{\max}$ will be at least equal to $Y + 4B + Y + B = 2Y + 5B$. Another formation of the schedule is to process jobs $q+1$ and $q+2$ on the same machine. In that case $C^\sigma_{\max}$ is at least equal to $2Y + 2B$. This indicates that $\frac{C^\sigma_{\max}}{C^*_{\max}}$ tends to 2 when Y is sufficiently large in comparison to B:

$$\frac{C^\sigma_{\max}}{C^*_{\max}} = \frac{2Y + 2B}{Y + 4B} \tag{5.1}$$

□

The result obtained in Theorem 8 can be further extended to other machine settings:

Theorem 9. *The existence of a* $(2-\varepsilon)$*-approximation algorithm for the problem* $O2|(a_j, L_j, b_j)|C_{\max}$ *implies* $P = NP$.

Proof. Note that problem $P2|(a_j, L_j, b_j)|C_{\max}$ can be regarded as $O2|(a_j, L_j, b_j)|C_{\max}$, i.e., the two-machine open-shop CTSP, where the initial task of all jobs must be processed on M_1, and their completion tasks must be processed on M_2. The schedule depicted in Figure 5.3 shows that, where the difference to the schedule of Figure 5.2 includes the completion task of job $q+3$ is processed before its initial task, i.e., b_{q+3} starts at time 0 on M_2, and a_{q+3} starts at time $Y + 3b$ on M_1. Also, the initial (completion) tasks of W_1

(W_2)-jobs are processed in the interval $[B, 2B]$ on M_1 (M_2), and the completion (initial) tasks of W_1 (W_2)-jobs are processed in the interval $[Y + 2B, Y + 3B]$ on M_2 (M_1). The rest of the proof will be similar to Theorem 7. □

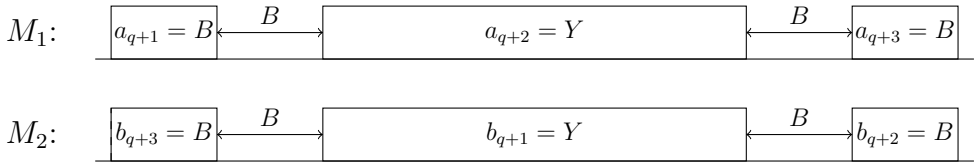

Figure 5.3 : The constructed schedule for problem $O2|(a_j, L_j, b_j)|C_{\max}$.

Theorem 9 improves the in-approximability results proposed by Ageev (2018) for $O2|(a_j, L_j, b_j)|C_{\max}$. Also note that Ageev (2018) showed that problem $O2|(a_j, L_j, b_j)|C_{\max}$ does not admit approximation algorithms with a ratio better than $(1.5-\varepsilon)$ unless $P = NP$, even for the case of $a_j = b_j, \forall j \in J$. We provide an optimal scheduling policy for two variants of $Pm|(a, L, b)|C_{\max}$ and $Pm|(p, L, p)|C_{\max}$ in the following.

5.3 Optimal schedule for $Pm|(a, L, b)|C_{\max}$ and $Pm|(p, L, p)|C_{\max}$

In this section, we generate an optimal schedule for problems $Pm|(a, L, b)|C_{\max}$ and $Pm|(p, L, p)|C_{\max}$. Recall that the single-machine variant of both these problems can be solved easily (see Baptiste (2010) and Orman and Potts (1997)). This implies that it only suffices to optimally allocate the jobs to m parallel identical machines as then the problem is reduced to m single-machine problems.

First, observe that problem $Pm|(p, L, p)|C_{\max}$ is obtained from $Pm|(a, L, b)|C_{\max}$ by letting $a = b = p$. As a result, we obtain an optimal allocation policy for problem $Pm|(a, L, b)|C_{\max}$, and then we extend our results to problem $Pm|(p, L, p)|C_{\max}$. We show that partitioning the jobs equally among the machines, i.e., the "equal allocation" policy, leads to an optimal schedule. We note that whenever the equal allocation policy is not possible, e.g., when the number of jobs is odd and the number of machines is even, then assigning the jobs as equally as possible to the machines results in an optimal schedule. As an example, given 3 jobs and 2 machines, we assign two jobs to machine 1, and one job to machine 2.
Next, we show the optimality of the equal allocation policy for $Pm|(a, L, b)|C_{\max}$.

5.3.1 Problem $Pm|(a, L, b)|C_{\max}$

We prove the optimality of the equal allocation policy for two identical machines, and then extend it to m identical machines. We assume $a \geq b$ and $L \geq \max\{a, b\}$, since otherwise the problem is trivial. Note that the proof also applies to $b \geq a$ due to the reverse property of the CTSP. Let s denote the starting time of a block of k jobs. We

define a block of jobs as the set of at most $k \leq \lceil \frac{L}{a} \rceil$ jobs. It follows from $a \geq b$ and $L \geq \max\{a, b\}$ that the first (initial) tasks of the jobs in the block are processed without idle time between the tasks. Hence, the k initial tasks will be processed in time intervals $[s, s+a], [s+a, s+2a], \ldots, [s+(k-1)a, s+ka]$. The second (completion) tasks of the jobs in the block are processed in the time intervals $[s+L+a, s+L+a+b], [s+L+2a, s+L+2a+b], \ldots, [s+L+ka, s+L+ka+b]$. As a result, the makespan of the block is equal to:

$$C_{\max}(k) = s + L + ka + b. \tag{5.2}$$

In the following we show that the makespan of the block is reduced if we equally allocate the jobs of the block to be executed on two machines, each with $\frac{k}{2}$ jobs (instead of executing all of the jobs of the block on only one machine).

Lemma 10. *Under two parallel identical machines,* $C_{\max}(k) > C_{\max}(\frac{k}{2})$.

Proof. Without loss of generality assume that there is only one block of k jobs. Let s be the starting time of the schedule. It follows from Equation (5.2) that $C_{\max}(k) = s+L+ka+b$ and $C_{\max}(\frac{k}{2}) = s + L + \frac{k}{2}a + b$.
Therefore, $C_{\max}(k) - C_{\max}(\frac{k}{2}) = (s+L+ka+b) - (s+L+\frac{k}{2}a+b) = \frac{k}{2}a > 0$. This implies that $C_{\max}(k) > C_{\max}(\frac{k}{2})$ for any $k \in \mathbb{R}$. Therefore, it is always optimal to equally (or as equally as possible) partition the jobs between the machines. □

Now, we show that Lemma (10) can be easily extended to m parallel identical machines:

Lemma 11. *Under m parallel identical machines,* $C_{\max}(k) > C_{\max}(\frac{k}{m})$.

Proof. Again assume that there is only one block of k jobs, and let s be the starting time of the schedule. It follows that $C_{\max}(k) - C_{\max}(\frac{k}{m}) = (s+L+ka+b) - (s+L+\frac{k}{m}a+b) = \frac{(m-1)k}{m}a > 0$, for any $k \in \mathbb{R}, m > 1$. □

Given a three-job instance with $a = 3, L = 4, b = 2$, and 2 machines, the optimal makespan is equal to 12, as shown in Figure 5.4, in which we assign two jobs to machine 1, and one job to machine 2. We note that not following the equal allocation policy may result in all three jobs being assigned to either machine 1 or machine 2. In either case, the makespan is equal to 21.

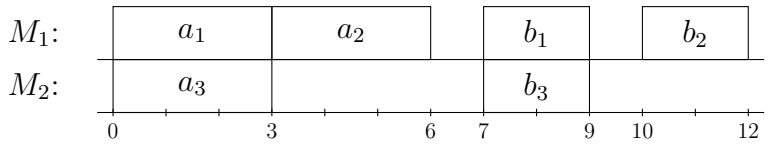

Figure 5.4 : The optimal schedule for a three-job instance of $P2|(a, L, b)|C_{\max}$.

Next, we extend our results to problem $Pm|(p, L, p)|C_{\max}$.

5.3.2 Problem $Pm|(p, L, p)|C_{\max}$

Assume $L \geq p$ since otherwise the problem is trivial. From the results for $Pm|(a, L, b)|C_{\max}$, it is easy to show that in $Pm|(p, L, p)|C_{\max}$ a block includes at most $k \leq \lceil \frac{L}{p} \rceil$ jobs and the makespan of the block is equal to

$$C_{\max}(k) = s + L + p(k + 1). \tag{5.3}$$

The makespan of the block is reduced if we allocate the jobs of the block to be executed on m identical machines, each with $\frac{k}{m}$ jobs since $C_{\max}(k) - C_{\max}(\frac{k}{m}) = (s + L + p(k + 1)) - (s + L + p\frac{k}{m+1}) = \frac{(m-1)k}{m}p > 0$, for any $k \in \mathbb{R}, m > 1$.

For example, consider the optimal makespan for the three-job instance with $a = 3, L = 4, b = 3$, which is equal to 13, if the equal allocation policy is applied, see Figure 5.5. Note that one may obtain a schedule with the makespan of 23 if the equal allocation policy is not followed.

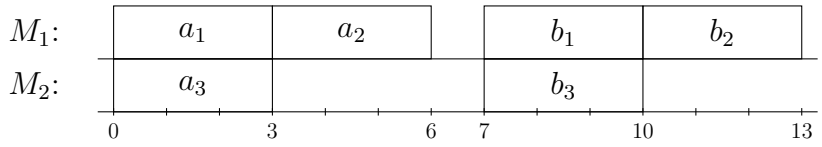

Figure 5.5 : The optimal schedule for the three-job instance of $P2|(p, L, p)|C_{\max}$.

Following these propositions, an optimal schedule for problems $Pm|(a, L, b)|C_{\max}$ and $Pm|(p, L, p)|C_{\max}$ is obtained through (i) partitioning the jobs equally among the machines; and, (ii) creating $\lceil \frac{\eta_i}{\lceil L/a \rceil} \rceil$ job blocks for machine i, where η_i is the number of jobs allocated to machine i: *(ii-1)* for $Pm|(a, L, b)|C_{\max}$, solving m single-machine problems, i.e., $1|(a, L, b)|C_{\max}$ each with the algorithm of Baptiste (2010), and *(ii-2)* for problem $Pm|(p, L, p)|C_{\max}$, blocks $1, 2, \ldots, r-1$ contain $k_{\max} = \lceil L/a \rceil$ jobs each and are all interleaved and, block r contains $\eta_i - rk_{\max}$ jobs that are interleaved. The solution procedure is polynomial in the problem input size.

Algorithm 6: Optimal procedure for $Pm|(a, L, b)|C_{\max}$ and $Pm|(p, L, p)|C_{\max}$.

Input: a, L, b, p, m.
Output: An optimal schedule.
Partition the jobs equally among the machines;
for $1 \leq i \leq m$ **do**
    Create $\lceil \frac{\eta_i}{\lceil L/a \rceil} \rceil$ job blocks;
    Solve m single-machine problems for $Pm|(a, L, b)|C_{\max}$, and for problem $Pm|(p, L, p)|C_{\max}$, form blocks $1, 2, \ldots, r-1$ with $k_{\max} = \lceil L/a \rceil$ jobs each and interleave them all, and block r with $\eta_i - rk_{\max}$ jobs that are interleaved;
end

Chapter 6

Coupled tasks on flow-shops

This chapter explores the coupled task scheduling problem (CTSP) in the flow-shop environment, and particularly, the ordered flow-shop environment. For that purpose, we first study the ordered flow-shop problem and propose benchmarks and solution algorithms for the problem in Section 6.1 and Section 6.2. Then, in Section 6.3, we explore new results for the CTSP in the classic flow-shop environment, and in the ordered flow-shop environment. The results presented in Section 6.1 and Section 6.2 are published in:

- Khatami, M., Salehipour, A., and Hwang, F. J. (2019). "Makespan minimization for the m-machine ordered flow shop scheduling problem". *Computers and Operations Research* 111, 400–414.
- Khatami, M. and Salehipour, A. (2020). "A relax-and-solve algorithm for the ordered flow-shop scheduling problem". *IEEE IEEM 2020*. Singapore.

In addition, the results presented in Section 6.3 are submitted for possible publication as:

- Khatami, M., Salehipour, A., and Cheng, T. C. E. (2021b). "Flow-shop scheduling with exact delays to minimize makespan". *Submitted to Computers & Industrial Engineering*.

6.1 Ordered flow-shops

The ordered flow-shop scheduling problem, first introduced by Smith (1968), is a subcategory of the classical flow-shop scheduling problem, where there are structured properties for processing times. In the classical flow-shop problem, the processing times of jobs are usually assumed to be independent of each other, as well as independent of the machines. The jobs' processing times in an industrial environment, however, may be related to the physical characteristics of the jobs and/or machines (Smith et al., 1975). For example, the processing times of jobs may be considered to be correlated because the machine with old design would process the jobs slower than that with the up-to-date technology. In the ordered flow-shop scheduling problem, the structured properties of jobs and machines can be described by the following two conditions, which apply to all jobs and machines: (1) job-ordered condition, that is, if a the processing time of a job is smaller than that of another job on some machine, then it should be the case on all

machines, and (2) machine-ordered condition, that is, if the processing time of a job on a machine is smaller than that on another machine, then it should be the case for all jobs. We will first study the ordered flow-shops, propose two efficient heuristics, and one fast Iterated Local Search algorithm. We also propose three sets of benchmarks consisting of 600 instances for the problem.

6.1.1 Problem definition and formulation

Given a set of jobs $J = \{1, \ldots, n\}$ and a set of different machines $M = \{1, \ldots, m\}$, we denote the processing time of job j on machine r by $p_{r,j}$, where $r \in M$ and $j \in J$. The ordered flow-shop scheduling problem is characterised by the following two conditions:

1. if for any two jobs $j, k \in J, p_{r,j} < p_{r,k}, r \in M$, then $p_{q,j} \leq p_{q,k}, \forall q \in M$; and,
2. if for any two machines $r, q \in M, p_{r,k} < p_{q,k}, k \in J$, then $p_{r,j} \leq p_{q,j}, \forall j \in J$.

We assume that only permutation schedules are allowed. Also, all jobs are assumed to be available at time zero, preemption of jobs is not allowed, and each machine can process at most one job at a time. The ordered flow-shop scheduling problem aims to obtain the best order of performing jobs on machines (a permutation) with respect to some criteria. We consider the objective of minimising the makespan. An example of the ordered flow-shop, which is borrowed from Panwalkar and Woollam (1980), is shown in Table 6.1.

Table 6.1 : An instance of a 5-job 6-machine ordered flow-shop scheduling problem (Panwalkar and Woollam, 1980).

	Machine					
Job	1	2	3	4	5	6
1	2	3	15	3	8	8
2	3	4	21	7	13	15
3	9	12	30	15	19	21
4	13	14	34	19	24	26
5	15	16	37	24	28	35

It is known that a pyramidal-shaped sequence is optimal for the ordered flow-shop scheduling problem (Smith et al., 1976). A pyramidal-shaped sequence consists of two sub-sequences, in which the jobs in the first sub-sequence are ordered by the shortest processing time (SPT) dispatching rule, and the jobs in the second sub-sequence are sequenced in the longest processing time (LPT) order. Those two sub-sequences will be hereafter called SPT sub-sequence and LPT sub-sequence, respectively.

Among different MILP formulations available for the flow-shop scheduling problem we implement the "starting time-based" formulation proposed by Wilson (1989), which

according to Tseng et al. (2004), is one of the best performing models for solving the permutation flow-shop scheduling problem. In this formulation, which is represented by Model PF1 in the following, a binary decision variable $z_{j,i}$ takes a value of 1 if job j is assigned to position $i \in J$ in the sequence, and 0 otherwise. Also, the non-negative decision variable $s_{r,i}$ indicates the starting time of the job in position i on machine r. Based on those notations, the Wilson's model for the permutation flow-shop scheduling problem can be presented by Model PF1:

Model PF1

$$z = \min C_{\max} = \min(s_{m,n} + \sum_{j=1}^{n} p_{m,j} z_{j,n}) \tag{6.1}$$

subject to

$$\sum_{j=1}^{n} z_{j,i} = 1, \quad 1 \le i \le n, \tag{6.2}$$

$$\sum_{i=1} z_{j,i} = 1, \quad 1 \le j \le n, \tag{6.3}$$

$$s_{1,1} = 0, \tag{6.4}$$

$$s_{1,i} + \sum_{j=1}^{n} p_{1,j} z_{j,i} = s_{1,i+1}, \quad 1 \le i \le n-1, \tag{6.5}$$

$$s_{r,1} + \sum_{j=1} p_{r,j} z_{j,1} = s_{r+1,1}, \quad 1 \le r \le m-1, \tag{6.6}$$

$$s_{r,i} + \sum_{j=1} p_{r,j} z_{j,i} \le s_{r+1,i}, \quad 1 \le r \le m-1, \quad 2 \le i \le n, \tag{6.7}$$

$$s_{r,i} + \sum_{j=1} p_{r,j} z_{j,i} \le s_{r,i+1}, \quad 2 \le r \le m, \quad 1 \le i \le n-1, \tag{6.8}$$

$$z_{j,i} \in \{0,1\}, \quad 1 \le j \le n, \quad 1 \le i \le n, \tag{6.9}$$

$$s_{r,i} \ge 0, \quad 1 \le r \le m, \quad 1 \le i \le n. \tag{6.10}$$

The objective function (eq. (6.1)) minimises the makespan. Constraints (6.2) and (6.3) are of the assignment problem, and ensure that exactly one job is assigned to each position and exactly one position is assigned to each job. Constraints (6.4) set the starting time of the schedule to zero. Constraints (6.5) and (6.6) ensure that there is no idle time on the first machine, and the first job in the sequence is processed on all m machines without delay. Constraints (6.7) indicate that the starting time of each job on machine $r+1$ is no earlier than its completion time on machine r. Constraints (6.8) ensure that the job in position $i+1$ does not start on machine r until the job in position i has completed its processing on that machine. Finally, constraints (6.9) and (6.10) state decision variables are binary and non-negative.

6.1.2 Proposed solution methods

Thanks to the pyramidal-shaped property, the size of the set of all candidate feasible solutions in an ordered flow-shop problem significantly reduces from $n!$ to 2^{n-1}. Hence, it is sensible to utilise the pyramidal-shaped property in the development of solution methods. In this study, we utilise the pyramidal-shaped property in developing problem-specific heuristics and meta-heuristics. The heuristic algorithms include a pyramidal variant of the Nawaz-Enscore-Ham (NEH) algorithm (Nawaz et al., 1983) and a procedure named "Pair-Insert". We also develop an iterated local search (ILS) meta-heuristic.

The Pyramidal-NEH algorithm

The well-known NEH algorithm is one of the most effective heuristics for solving the permutation flow-shop scheduling problem. Ruiz and Maroto (2005) showed this through a comprehensive evaluation of available heuristics for the permutation flow-shop scheduling problem. One may refer to Dong et al. (2008) and Kalczynski and Kamburowski (2008) for more details on the applications of the NEH algorithm and its variants.

The NEH algorithm consists of three steps. In Step 1, a sequence π_0 is generated by sorting the n jobs in non-increasing order of the total processing times of m operations. In Step 2, the first two jobs from π_0 are selected, and a sequence π for the two jobs with the minimum makespan is generated. A complete sequence is then constructed in Step 3, where the remaining jobs of π_0 are inserted in the sequence π one by one, such that a job is inserted into a position, among all possible positions, which minimises the makespan of the yielded partial sequence. Note that there are i possible positions for inserting job $3 \leq i \leq n$ into the partial sequence. Taillard (1990) improved the computational complexity of the NEH algorithm from $O(n^3m)$ to $O(n^2m)$. With regard to tie-breaking rules, Fernandez-Viagas and Framinan (2014) showed that their idle time-based tie-breaking mechanism outperforms all other methods.

We propose a modification to Step 3 of the original NEH algorithm to improve its efficiency for solving the ordered flow-shop scheduling problem, based on the pyramidal property. Our proposed solution procedure is thus named the Pyramidal-NEH algorithm and is shown in Algorithm 7.

Note that in case of ties, Algorithm 7 chooses the sequence π'. The Pyramidal-NEH algorithm restricts the allocation of an unassigned job only to the first or the last position of the partially built sequence, maintaining therefore the pyramidal-shaped property of the final sequence. In the Pyramidal-NEH algorithm $2(n-1)$ partial sequences are investigated in order to build a complete solution, whereas $\frac{n\times(n+1)}{2} - 1$ partial sequences need be considered in the original NEH algorithm. The time complexities of Steps 1 and 2 of Algorithm 7 are $O(n \log n)$ and $O(m)$, identical to those of the NEH algorithm. Step 3, however, performs $2mn$ operations when calculating the makespan of the partial se-

Algorithm 7: The Pyramidal-NEH algorithm.

Input: The processing times of n jobs on all the m machines.
Output: A job sequence π.
Step 1: Generate a sequence π_0 by sorting the n jobs in the non-increasing order of the total processing times of m operations;
Step 2: Select from π_0 the first two jobs, and generate a sequence π of the two jobs with the minimum makespan;
Step 3 (Complete sequence construction):
for $3 \leq i \leq n$ **do**
    Generate π' and π'' by letting π be prefixed and suffixed respectively with the ith job of π_0;
    Compare the makespans of π' and π'', and set π as the one with smaller makespan;
end
return *the constructed sequence* π

quence at each iteration. Therefore, the time complexity of the Pyramidal-NEH algorithm is $O(n^2m)$.

The Pair-Insert algorithm

Another heuristic algorithm that we propose for the ordered flow-shop scheduling problem is named the "Pair-Insert" algorithm, that utilises the pyramidal-shaped property as well as the general idea of the NEH algorithm, as summarized in Algorithm 8. Performing the same first two steps as Algorithm 7, the Pair-Insert algorithm constructs a partial sequence in Step 3 by adding unassigned jobs in pairs. The pyramidal-shaped property of the problem leaves only four possible choices for concatenating a pair of unassigned jobs with a partial sequence. Let π_0 be the sequence obtained in Step 1 and π be a partial sequence constructed with the first $i-1$ jobs of π_0. Then the four choices of concatenation are: (1) $(i+1, i, \pi)$, i.e., prefix both jobs $i+1$ and i to π; (2) $(\pi, i, i+1)$, i.e., suffix both jobs $i+1$ and i to π; (3) $(i, \pi, i+1)$, i.e., prefix job i and suffix job $i+1$ to π; and, (4) $(i+1, \pi, i)$, i.e., prefix job $i+1$ and suffix job i to π. Among those four permutations, the one with the least makespan is selected as the partial sequence constructed with the first $i+1$ jobs of π_0.

Note that in case of ties, the algorithm chooses the sequence with the smallest index among the four choices. The Pair-Insert algorithm also runs in $O(n^2m)$ and the total number of partial sequences to be evaluated is equal to $2(n-1)$.

An Iterated Local Search algorithm

The Iterated Local Search (ILS) algorithm has been widely applied to solve the flow-shop scheduling problems, and has delivered high quality solutions. Examples include the studies of Stützle (1998), Marmion et al. (2011) and Lin et al. (2013). We propose

Algorithm 8: The Pair-Insert algorithm.

Input: The processing times of n jobs on all the m machines.
Output: A job sequence π.
Step 1: Generate a sequence π_0 by sorting the n jobs in the non-increasing order of the total processing times of m operations;
Step 2: Select from π_0 the first two jobs, and generate a sequence π of the two jobs with the minimum makespan;
Step 3 (Complete sequence construction):
Set $i = 3$;
while $i \leq n$ **do**
    Generate four partial sequences by concatenating π with the ith and $(i+1)$th jobs of π_0 in accordance with (1): $(i+1, i, \pi)$, (2): $(\pi, i, i+1)$, (3): $(i, \pi, i+1)$, and (4): $(i+1, \pi, i)$;
    Compare the makespans of the four permutations, and set π as the one with the smallest makespan;
    Set $i = i + 2$;
end
return *the constructed sequence* π

an ILS algorithm for the ordered flow-shop scheduling problem, that is as follows. Upon generating an initial solution, a local search is applied to obtain an improved solution. Then, a perturbation is iteratively applied to the current solution, followed by the local search, which is applied to the perturbed solution. The process continues until the stopping criterion is met (Talbi, 2009).

Our proposed ILS algorithm incorporates a post-processing procedure in order to further improve the best obtained solution. For this reason, the post-processing is initiated when the main algorithm completes. The post-processing includes a local search algorithm, which utilises a different neighborhood structure from that used in the main algorithm. The motivation behind the post-processing is that exploring the search space with a different neighborhood structure that does not utilise the pyramidal-shaped property may lead to exploring some yet unexplored solutions, and we may therefore obtain improved solutions. Recall that although we know that one of the pyramidal-shaped sequences must be optimal, a non-pyramidal sequence may also be optimal. Algorithm 9 summarises our ILS algorithm, where Phase 1 represents the main algorithm and Phase 2 represents the post-processing procedure.

Generating initial solutions Both random and greedy approaches have been used to generate initial solutions for the flow-shop scheduling problem (Ponnambalam et al., 2001; Osman and Potts, 1989). Since the ILS algorithm includes applying random perturbations to the solution, we utilise the Pair-Insert heuristic (Algorithm 8), which is a deterministic algorithm, to build an initial solution for the ILS algorithm. This implies that the initial

Algorithm 9: The ILS algorithm for the ordered flow-shop problem.

Input: The processing times of n jobs on m machines, $d_0, d_1, T_0, F, S, time\_limit$.
Output: A job sequence.
Phase 1:
Pair-Insert algorithm(); % Algorithm 8
Local search algorithm(); % Algorithm 10
$d = d_0$, $flag[F] = 0$, $T = T_0$;
while $elapsed\_time \leq time\_limit$ **do**
  for $1 \leq l \leq d$ **do**
    Perturbation();
    Local search algorithm(); % Algorithm 10
  end
  if *"no new solution" found* **then**
    $flag[F] = flag[F] + 1$;
    if $flag[F] \geq F$ **then**
      $d = d_1$;
    end
  else
    $flag[F] = 0$, $d = d_0$;
  end
  $T = \alpha \times T$;
end

Phase 2:
while $elapsed\_time \leq time\_limit$ **do**
  Swap two randomly selected jobs in the current solution;
  if *superior solution found* **then**
    Update the best found solution;
  end
end
return *the best found solution*

solution will be a pyramidal-shaped sequence.

Perturbation The main procedure in the ILS algorithm includes two operations of perturbation and local search. Through those operations, the ILS iteratively obtains improved solutions.

Each iteration of perturbation includes removing some jobs from their current positions in the sequence, and inserting them at some other positions. While jobs are randomly selected to be removed, their insertion positions are determined such that the pyramidal-shaped property of the solution is maintained. Precisely, if job j is removed from SPT sub-sequence, then it is inserted into LPT sub-sequence, and between two adjacent jobs l and k, where $\sum_{r=1}^{m} p_{r,l} \geq \sum_{r=1}^{m} p_{r,j} \geq \sum_{r=1}^{m} p_{r,k}$. Likewise, if job j is removed from LPT sub-sequence, it is then inserted into SPT sub-sequence, and that between two adjacent jobs l and k, where $\sum_{r=1}^{m} p_{r,l} \leq \sum_{r=1}^{m} p_{r,j} \leq \sum_{r=1}^{m} p_{r,k}$.

In order to control the diversification and intensification aspects of the ILS algorithm, the number of such removals and insertions (i.e., d) is adaptively changed during the course of algorithm's operation. For this reason, the number of removals and insertions is increased to d_1, from its default value of d_0, whenever certain number of consecutive iterations, denoted as F, are performed with no improvements in the solution. On the other hand, if an improved solution is obtained, the number of removals and insertions is reduced to the default value of d_0. The sequence generated by the perturbation operation is always accepted.

Local search The local search method applied in the proposed ILS algorithm utilises the "insert" operator to build new neighboring solutions, and operates as follows. A job is removed from its position in the sequence, and is inserted into another position such that the pyramidal-shaped property is maintained. This operation, which is performed for all jobs in the sequence, starts from the job in the first position in the sequence, and proceeds to the job in the last position. The process is continued until either a "new solution" is obtained, or all neighboring solutions are explored without obtaining such a new solution. A neighbor solution is defined as a "new solution", if either it has a better objective function value, or it has a worse objective function value, however, meeting the "acceptance criterion". Note that if a new solution is obtained, then it is accepted, i.e., the "first improvement strategy" is applied, and the process is re-started from the beginning of the sequence. This process is continued until S iterations are performed. Algorithm 10 summarises the proposed local search.

Note that in the typical iterated local search algorithm the local search goes for a local optimum (Stützle and Ruiz, 2018). Though, delivering a local optimum is not guaranteed in our algorithm since the local search is applied for a certain number of iterations, i.e., parameter S.

Acceptance and termination criteria We implement an acceptance criterion that accepts both superior and inferior solutions. Precisely, an improved solution is always accepted. A worse solution can be accepted if the local search cannot deliver a new solution (see Local search) and also the degradation amount is not greater than a certain threshold (similar to the threshold accepting algorithm). The threshold parameter is initialised as T_0 and decremented according to the geometric decrement rule of $T = \alpha \times T$, where $0 < \alpha < 1$. The algorithm continues until a certain time limit is elapsed.

Post-processing The Phase 2 of the ILS algorithm includes a post-processing procedure, which is applied to the best delivered solution of Phase 1. The Phase 2 applies a "swap" local search and aims to deliver new high quality solutions. For that, the neigh-

Algorithm 10: Local search procedure of the ILS algorithm.

Input: Current solution.
Output: A new solution or the current solution.
$flag[P] = 1$, $flag[S] = 0$;
while $flag[P] = 1$ ***and*** $flag[S] \leq S$ **do**
    $i = 1$, $flag[C] = 0$;
    while $flag[C] = 0$ **do**
        Remove the job in the ith position and insert it into another position in the sequence, maintaining the pyramidal property;
        if *a "new solution" (superior solution or inferior one passing the acceptance criterion) obtained or all neighbor solutions are generated with "no new solution"* **then**
            $flag[C] = 1$;
        end
        $i = i + 1$;
    end
    if *a "new solution" is found* **then**
        Update the current solution;
        if *the "new solution" is a superior solution* **then**
            Update the best found solution;
        end
    else if *"no new solution" found* **then**
        $flag[P] = 0$;
    end
    $flag[S] = flag[S] + 1$;
end
return *the solution*

boring solutions are built by using the swap operator, applied to randomly selected jobs, regardless of maintaining the pyramidal-shaped property. Discovering non-pyramidal sequences is a way to let the algorithm search for better solutions in unexplored spaces. Phase 2 is run for the same time-limit as Phase 1.

6.1.3 Computational experiments

In this section we report the computational results of the developed heuristic algorithms on three sets of randomly generated benchmark instances. First, we explain the instance generation procedure, and discuss tuning the parameters of the algorithms. Then, we run the algorithms on the generated instances and report the computational results.

The proposed algorithms were coded in the computational package MATLAB version 9.4 (MATLAB, 2018). We perform all computational experiments on the same PC mentioned in Section 3.3. For comparison purposes we solve the instances with the available solution method for the permutation flow-shop problem. The top performing methods for the permutation flow-shop problem include the Iterated Greedy Algorithms (IGA) of Ruiz

and Stützle (2007) and of Fernandez-Viagas and Framinan (2014). Recently, Benavides and Ritt (2018) proposed a new IGA (denoted as IGA_{BR} throughout this study) that reports superior results for the permutation flow-shop problem. The recently published IGA_{BR} is very effective in solving both the permutation and the non-permutation flow-shop scheduling problems. The IGA_{BR} algorithm was coded in the programming language C++, and the code is available to the public. We use the same code. In addition, we use the IBM ILOG CPLEX version 12.8.0 (CPLEX, 2017) to solve Model PF1 to optimality (where possible). Because the proposed ILS and IGA_{BR} start with an initial solution, we use the same initial solution to warm start the solver CPLEX. This ensures a fair comparison. We also force the solver CPLEX to utilise only one processor (thread), and to use a time limit as a stopping criterion. For the remaining parameters of the CPLEX we use the default values.

Instance generation and evaluation

We generate ordered instances, as benchmarks, for the problem, based on the available instances for the permutation flow-shop problem, i.e., based on the benchmarks of Taillard (1993) and Vallada et al. (2015).

Taillard's instances were chosen from a large set of randomly generated instances, aiming to generate a set of the most difficult instances. The Taillard's instances range from 20 to 500 jobs, and 5 to 20 machines, and include 12 combinations of number of jobs and number of machines denoted as (n, m). Those 12 combinations include (20, 5), (20, 10), (20, 20), (50, 5), (50, 10), (50, 20), (100, 5), (100, 10), (100, 20), (200, 10), (200, 20), and (500, 20). Each combination consists of 10 instances. Therefore, a total number of 120 instances were produced. For more recent studies of Taillard's instances we refer the interested reader to Khatami and Zegordi (2017), Khorasanian and Moslehi (2017), and Liu et al. (2017b).

Vallada et al. (2015) performed extensive experiments and selected 240 "small" instances and 240 "large" instances, from a pool of 72,000 instances. The number of jobs in the Vallada et al.'s small instances is chosen from the set $\{10, 20, 30, 40, 50, 60\}$, and the number of machines is chosen from the set $\{5, 10, 15, 20\}$. Hence, there are $6 \times 4 = 24$ combinations of n and m. Each combination consists of 10 instances. This results in 240 instances. For the large instances, the number of jobs is selected from the set $\{100, 200, 300, 400, 500, 600, 700, 800\}$ and the number of machines is selected from the set $\{20, 40, 60\}$. Therefore, there exist $8 \times 3 = 24$ combinations of n and m, where each combination consists of 10 instances, and therefore, a total number of 240 large instances are produced. We refer the interested reader to Fernandez-Viagas et al. (2017) and Dubois-Lacoste et al. (2017) for two recent studies of Vallada et al.'s instances.

We note that Taillard's and Vallada et al.'s instances are for the permutation flow-

shop scheduling problem. In order to generate challenging benchmark instances for the ordered flow-shop scheduling problem, the following three steps are performed on all the 120 Taillard's instances and on all the 480 Vallada et al.'s instances.

- Step 1: Sort, for each machine, the operation processing times of all the n jobs in a non-decreasing order;
- Step 2: Sort, for each job, the operation processing times on all the m machines in a non-decreasing order;
- Step 3: Permute the order of machines.

Steps 1 and 2 generate the instances in which the processing times of jobs are in non-decreasing order, i.e., every job has its smallest processing time on the first machine, the second smallest processing time on the second machine and so on. Those instances can be easily solved by the SPT rule. Watson et al. (2002) also showed that the structured problems are relatively easy to solve. Thus, we perform Step 3 for permuting the order of machines, aiming at producing non-trivial instances. We let `T`, `S` and `L` denote our three sets of instances derived from the Taillard's and the Vallada et al.'s small and large instances.

According to Vallada et al. (2015), the difficulty of an instance can be measured as the percentage of the deviation from its lower bound (LB) solution, i.e., $\delta = 100 \times (UB - LB)/LB$, where the LB is obtained by utilising the method of Ladhari and Haouari (2005) (eq. (6.11)), and UB is the objective function's upper bound calculated by using the Pair-Insert algorithm.

$$LB = \max_{1 \leq r < q \leq m} [\min_{j \in J} R_{r,j} + C^{r,q}_{\max}(J) + \min_{j \in J} Q_{q,j}], \tag{6.11}$$

In Equation (6.11), $C^{r,q}_{\max}(J)$ denotes the optimal makespan of a two-machine permutation flow-shop problem with time lags, defined on the job set J and machine pair (r, q). In addition, $R_{r,j}$ and $Q_{q,j}$ are the summation of the processing times of job j on some subset of machines. For more details on the implementation of this lower bound, see "LB5" in Ladhari and Haouari (2005). Note that this tight lower bound is calculated based on the bottleneck machine, which is suitable for our problem, particularly, because the number of bottleneck machines in the ordered flow-shop scheduling problem is small.

For the 5-machine instances we can easily evaluate all 120 possible permutations of machines, and choose the one with the largest value of δ. For instances with 10 or more machines, however, we evaluate 1000 random permutations of machines instead, because evaluating all possible permutations is very expensive, if not impractical. Indeed, evaluating 1000 permutations ensures, to some extent, that the so generated instances are

very difficult to solve. In summary, a total number of 520,800 instances were evaluated (93,600 instances of `T`, 187,200 instances of `S` and 240,000 instances of `L`), out of which 600 instances were produced as very difficult and challenging instances for the ordered flow-shop scheduling problem (120 for `T`, 240 for `S`, and 240 for `L`). For the illustration purposes, we show the name of an instance in the format `B-n-m-x`, where `B` is the benchmark name that can be `T`, `S` or `L`, `n` and `m` denote the number of jobs and the number of machines, and `x` denotes the instance number, ranging from 1 to 10. For example, an instance named `T-20-5-1` is the instance in benchmark `T`, that includes 20 jobs and 5 machines and is assigned the index number 1.

Figure 6.1 highlights the necessity of producing instances with large values of δ. For comparison purposes, the box-plot includes a set of "arbitrary" instances, which were generated by randomly permuting machines based on instances of benchmark `T`. According to the box-plot, the δ metric may reveal how easily such arbitrary instances might be solved. The overall δ for the benchmark `T` is 3.33, while it is 0.64 for the arbitrary instances. In addition, our preliminary tests showed that the Pair-Insert algorithm obtains a proven optimal solution for 37 (out of 120) arbitrary instances, even for the largest size (a proven optimal solution is either reported by the solver CPLEX or is extracted when $UB = LB$). This means that at least about 31% of arbitrary instances were optimally solved with the Pair-Insert algorithm. In addition, we realise that the minimum of δ for every category of arbitrary instances is 0. Note that a value of 0 for δ implies that there was at least one instance solved to optimality by the Pair-Insert algorithm. We also used statistical tests to validate the obtained results. Because the data do not follow a normal distribution, we applied the Wilcoxon signed-rank test for the paired data with a 95% confidence level. We observed a p-value of 0 for the test, confirming therefore the statistical significance of selecting instances with large values of δ. Overall, the results indicate the impact of generating instances with large deviations (associated with large values of δ) to build challenging benchmark instances for the ordered flow-shop scheduling problem.

We note that all 600 benchmark instances, which we generate in this study, are available online in the Mendeley data repository at http://dx.doi.org/10.17632/cd2rv7hyyj.1. Future studies for the ordered flow-shop scheduling problem could benefit from these benchmark sets in evaluating their methods.

Evaluation of the proposed heuristic algorithms

We first evaluate the proposed heuristic algorithms, i.e., the Pyramidal-NEH algorithm (denoted as NEH_{Pyr}) and the Pair-Insert algorithm (denoted as PI), by solving the three sets of benchmark instances and comparing the outcomes with those obtained by the NEH algorithm with Taillard acceleration (denoted as NEH_{Tai}), and also with the

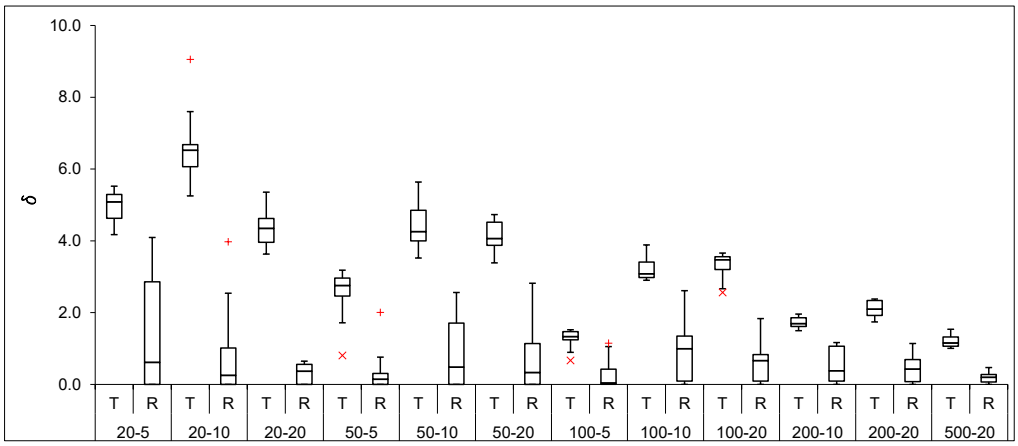

Figure 6.1 : Comparison between the instances of benchmark `T` and the arbitrary instances (denoted as `R` in the figure).

NEH algorithm with Taillard acceleration and tie-breaking rule of Fernandez-Viagas and Framinan (2014) (denoted as NEH_{FF}).

We use three criteria of "number of best obtained solutions (N_{Best})", "average relative percentage deviation ($ARPD$)", and "average relative percentage time ($ARPT$)" to evaluate the performance of those four heuristic algorithms.

The quality of the results is shown by the relative percentage deviation (RPD), which is calculated as $\frac{z-z^*}{z^*} \times 100$, where z is the objective function value, i.e., the makespan delivered by an algorithm, and z^* is the best known objective function value for the instance. The metric $ARPD$ is the average of RPDs over groups of instances. The metric RPT denotes the relative percentage computation time of an algorithm, and is calculated as $\frac{CPU-ACT}{ACT} \times 100$, where CPU is the computation time of the algorithm for an instance, and ACT is the average computation time for all algorithms on the same instance. The $ARPT'$ is the average of RPTs over groups of instances, and $ARPT$ is then calculated as $ARPT = ARPT' + 1$. For more details on the metrics $ARPD$ and $ARPT$ we refer the interested reader to Fernandez-Viagas et al. (2017).

Table 6.2 presents the values of the three metrics N_{Best}, $ARPD$ and $ARPT$ for the four heuristics of NEH_{Pyr}, PI, NEH_{Tai}, and NEH_{FF} over the three sets of benchmark instances. The highlighted numbers denote the outperforming values. The results show that the PI is the best algorithm in terms of N_{Best}, and the NEH_{Pyr} is the second best. The PI generates the best solution for almost 73% of the instances, and in the order of over 4 times more than the NEH_{Tai} algorithm. The quality of the PI algorithm is further confirmed based on the metric $ARPD$, where it delivers solutions with an $ARPD$ value of 0.81, where the $ARPD$ of NEH_{FF} (the second best algorithm) is 1.08. The quality of solutions of NEH_{Pyr} is similar to that of NEH_{Tai}, and slightly worse than that of NEH_{FF}. Finally, the NEH_{Pyr} is the fastest and the PI is the slowest algorithms based on the metric

ARPT.

Table 6.2 : Summary of metrics N_{Best}, *ARPD* and *ARPT* for the four heuristic algorithms on the three benchmark sets.

Metric	Benchmark	NEH_{Tai}	NEH_{FF}	NEH_{Pyr}	PI
N_{Best}	T	20	16	20	**89**
	S	42	39	42	**211**
	L	44	50	55	**137**
	Sum	106	105	117	**437**
ARPD	T	1.30	1.22	1.30	**0.91**
	S	1.56	1.51	1.56	**1.10**
	L	0.61	0.52	0.61	**0.42**
	Average	1.16	1.08	1.16	**0.81**
ARPT	T	0.54	0.76	**0.43**	2.28
	S	0.49	0.66	**0.38**	2.47
	L	0.86	1.10	**0.65**	1.39
	Average	0.63	0.84	**0.49**	2.05

The superiority of the PI algorithm can be further investigated in Figure 6.2, where the *RPD*s of the heuristic algorithms are compared based on the three benchmark sets. In addition, the Wilcoxon signed-rank test for the paired data of the *RPD* values of the heuristic algorithms with a 95% confidence level, which is detailed in Table 6.3, confirms the quality of the PI algorithm.

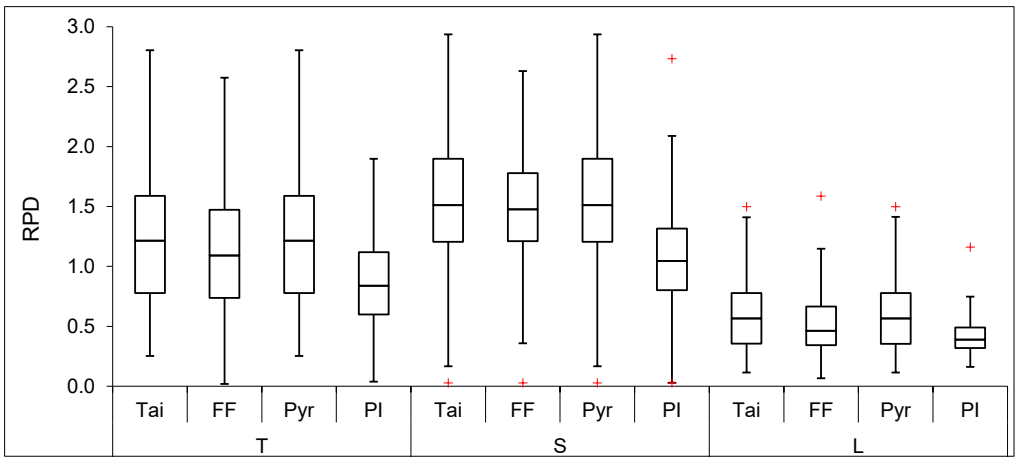

Figure 6.2 : *RPD*s of the heuristic algorithms on three benchmark sets T, S and L.

We should note that although the PI heuristic is not as fast as the other three heuristics, the computation time is not a concern because the largest instances (i.e., L-800-60-x) is solved within a second by any of the four heuristic algorithms. To conclude, the PI can be considered as the best performing heuristic for generating initial solutions for the ordered flow-shop scheduling problem, and therefore, we utilise the PI heuristic to generate initial solutions for the ILS algorithm.

Table 6.3 : Wilcoxon signed-rank test for RPDs of the heuristic algorithms.

Benchmark	Comparison	Wilcoxon signed-rank test	
		Test statistic	p-value
T	PI vs. NEH_{Tai}	455	0.000
	PI vs. NEH_{FF}	466	0.000
	PI vs. NEH_{Pyr}	457	0.000
S	PI vs. NEH_{Tai}	970	0.000
	PI vs. NEH_{FF}	726	0.000
	PI vs. NEH_{Pyr}	970	0.000
L	PI vs. NEH_{Tai}	2621	0.000
	PI vs. NEH_{FF}	4100	0.000
	PI vs. NEH_{Pyr}	2667	0.000

Parameter tuning for the ILS algorithm

A full factorial design of experiments (Montgomery, 2017) is applied to tune the values of the parameters. Those parameters include T_0 (the initial threshold value), α (the threshold decrement parameter), the upper and lower levels of d (the number of modifications in the perturbation), F (the number of iterations with no improvement), and S (the maximum number of times that the local search is performed at each iteration). To set the initial threshold value for each instance, a coefficient of its lower bound LB is used. Four candidate values of 0.05, 0.1, 0.15 and 0.2 are considered for the coefficient. For the threshold decrement parameter, we consider $(1/T_0)^{(1/t)}$, where t is selected from $\{400, 600, 800, 1000\}$. Also, recall that the number of modifications in the perturbation is changed adaptively as the algorithm proceeds. It starts with a default value of d_0, and if F consecutive iterations are performed with no improvement, it is increased to d_1. We consider two scenarios for the values of the pair (d_0, d_1), that are: $(\lfloor \log(n/2) \rceil, \lfloor \log 10n \rceil)$ and $(\lfloor 2 \log n \rceil, \lfloor 3 \log n \rceil)$. We also consider four candidate values of 5, 10, 15 and 20 for F. For the intensification parameter S, that is the maximum number of times that the local search is performed at each iteration, we consider four values of 6, 8, 10 and 12. Those candidate values lead to a total number of $4 \times 4 \times 2 \times 4 \times 4 = 512$ combinations. To tune the algorithm, 35 instances with different sizes are solved with those 512 parameter combinations. Hence, a total number of 17,920 instances are solved and RPD of each problem is used to represent its quality. Based on the obtained results of the means of the analysis of variance (ANOVA) technique, d and S statistically play a significant role in the performance of the algorithm. The obtained optimal value of those parameters are presented in Table 6.4.

Evaluation of the ILS algorithm

We test the proposed ILS algorithm on the three sets of benchmark instances of Section 6.1.3. For comparison purposes, we solve those instances with IGA_{BR}, and also

Table 6.4 : Value of the parameters used in the ILS algorithm.

Parameter	Definition	Value
T_0	Initial threshold value	$0.1 \times LB$
α	Threshold decrement factor	$(1/T_0)^{(1/600)}$
(d_0, d_1)	Default and increased number of removals and insertions	$(\lfloor \log (n/2) \rceil, \lfloor \log 10n \rceil)$
F	Number of iterations with no improvement	15
S	The maximum number of times the local search is performed at each iteration	10

with the solver CPLEX (of solving Model PF1) with a warm start. The solution of the PI heuristic is utilised to warm start the CPLEX.

We use the three criteria of N_{Best}, $ARPD$ and $ARPT$ to evaluate the performance of ILS and IGA$_{BR}$ algorithms, and the solver CPLEX. We use the same time limit (milliseconds) for ILS and IGA$_{BR}$ algorithms, which is calculated as $\tau \times n \times m$, where τ is set to 15 (hence, each phase of ILS algorithm is set to run for $\frac{\tau \times n \times m}{2}$ milliseconds). We solve each instance for three times by the ILS and IGA$_{BR}$ algorithms, and report the average of the three runs. A computation time limit of 3,600 seconds (60 minutes) is applied to the CPLEX, and the CPLEX is set to utilise one processor. Other than those, we use the default values for the remaining parameters of the CPLEX. We note that an optimal solution to Model PF1 is not guaranteed by setting a time limit for the CPLEX.

Table 6.5 presents the results of the three methods on the instances of benchmark `T`, where the best values across the methods are highlighted. It is easily realised that the ILS algorithm is performing very well on the larger instances, where it is the superior algorithm for the 80 large instances (`T-50-10-x` to `T-500-20-x`).

Table 6.5 : Detailed comparison of the methods on the benchmark `T`.

Problem	ILS			IGA$_{BR}$			CPLEX		
size (n-m)	N_{Best}	$ARPD$	$ARPT$	N_{Best}	$ARPD$	$ARPT$	N_{Best}	$ARPD$	$ARPT$
20-5	5	0.06	**0.86**	**10**	**0.00**	**0.86**	**10**	**0.00**	1.28
20-10	3	0.03	**0.64**	**10**	**0.00**	**0.64**	**10**	**0.00**	1.72
20-20	5	0.01	**0.65**	**10**	**0.00**	**0.65**	**10**	**0.00**	1.71
50-5	4	**0.03**	**0.00**	3	0.04	**0.00**	**5**	0.05	2.99
50-10	**8**	**0.02**	**0.01**	0	0.09	**0.01**	3	0.10	2.99
50-20	**9**	**0.02**	**0.01**	0	0.09	**0.01**	1	0.06	2.98
100-5	**9**	**0.01**	**0.01**	1	0.08	**0.01**	0	0.18	2.99
100-10	**10**	**0.01**	**0.01**	0	0.17	**0.01**	0	0.32	2.98
100-20	**10**	**0.01**	**0.02**	0	0.16	**0.02**	0	0.32	2.95
200-10	**10**	**0.01**	**0.02**	0	0.13	**0.02**	0	0.54	2.95
200-20	**10**	**0.01**	**0.05**	0	0.16	**0.05**	0	0.52	2.90
500-20	**10**	**0.00**	**0.12**	0	0.10	**0.12**	0	0.36	2.77
Total	**93**	**0.02**	**0.20**	34	0.08	**0.20**	39	0.20	2.60

Table 6.6 presents the results of the methods on the instances of benchmark S. The best values across the methods are highlighted. It is clear that the ILS algorithm is the superior method for the larger instances, i.e., for the last 80 ones (S-50-5-x to S-60-20-x). The average computation time for the solver CPLEX is more than 1800 seconds, while this is less than 7 seconds for ILS and IGA$_{BR}$ algorithms.

Table 6.6 : Detailed comparison of the methods on the benchmark S.

Problem	ILS			IGA$_{BR}$			CPLEX		
size (n-m)	N_{Best}	$ARPD$	$ARPT$	N_{Best}	$ARPD$	$ARPT$	N_{Best}	$ARPD$	$ARPT$
10-5	**10**	**0.00**	1.42	**10**	**0.00**	1.42	**10**	**0.00**	**0.15**
10-10	**10**	**0.00**	1.43	**10**	**0.00**	1.43	**10**	**0.00**	**0.14**
10-15	**10**	**0.00**	1.44	**10**	**0.00**	1.44	**10**	**0.00**	**0.13**
10-20	**10**	**0.00**	1.45	**10**	**0.00**	1.45	**10**	**0.00**	**0.11**
20-5	4	0.04	**0.57**	**10**	**0.00**	**0.57**	**10**	**0.00**	1.87
20-10	4	0.04	**0.64**	**10**	**0.00**	**0.64**	**10**	**0.00**	1.72
20-15	5	0.02	**0.60**	**10**	**0.00**	**0.60**	**10**	**0.00**	1.81
20-20	4	0.02	**0.74**	**10**	**0.00**	**0.74**	**10**	**0.00**	1.53
30-5	3	0.06	**0.02**	**8**	0.01	**0.02**	**8**	**0.01**	2.97
30-10	2	0.08	**0.08**	5	0.01	**0.08**	**9**	**0.00**	2.84
30-15	2	0.03	**0.06**	5	0.01	**0.06**	**9**	**0.00**	2.88
30-20	4	0.02	**0.16**	4	0.01	**0.16**	**9**	**0.01**	2.68
40-5	2	0.05	**0.00**	3	0.03	**0.00**	**8**	**0.01**	2.99
40-10	2	0.05	**0.03**	1	0.05	**0.03**	**9**	**0.02**	2.95
40-15	4	0.02	**0.02**	0	0.04	**0.02**	**8**	**0.01**	2.97
40-20	2	0.01	**0.01**	0	0.05	**0.01**	**9**	**0.01**	2.97
50-5	**5**	**0.02**	**0.00**	2	0.05	**0.00**	4	0.03	2.99
50-10	**7**	**0.02**	**0.01**	0	0.10	**0.01**	4	0.08	2.99
50-15	4	**0.01**	**0.01**	0	0.09	**0.01**	**6**	0.02	2.98
50-20	3	**0.02**	**0.01**	1	0.07	**0.01**	**7**	0.03	2.97
60-5	**9**	**0.01**	**0.05**	3	0.04	**0.05**	2	0.10	2.89
60-10	4	**0.02**	**0.01**	0	0.12	**0.01**	**7**	0.03	2.99
60-15	5	**0.02**	**0.01**	0	0.11	**0.01**	**6**	0.06	2.98
60-20	**9**	**0.02**	**0.01**	0	0.10	**0.01**	1	0.08	2.97
Total	124	0.02	**0.37**	112	0.03	**0.37**	**186**	**0.02**	2.27

The results of the three methods on the instances of benchmark L, which are reported in Table 6.7, show the good performance of the ILS algorithm. We note that the ILS algorithm outperforms both IGA$_{BR}$ and CPLEX, in terms of solution time and quality.

Table 6.8 provides a summary of the computational results of the three methods. Based on the metric N_{Best}, the ILS is the best performing algorithm among all, because it delivers the best solution for more than 75% of the instances. Also, quality of solutions of the ILS algorithm is very good according to the metric $ARPD$. More precisely, the ILS algorithm generates superior solutions than the IGA$_{BR}$ algorithm, and that in the order of almost 7 times. The quality of the ILS algorithm can be further observed in Figure 6.3, where the RPDs of the methods are compared across the three benchmark sets. Since

Table 6.7 : Detailed comparison of the methods on the benchmark L.

Problem	ILS			IGA_{BR}			CPLEX		
size (n-m)	N_{Best}	$ARPD$	$ARPT$	N_{Best}	$ARPD$	$ARPT$	N_{Best}	$ARPD$	$ARPT$
100-20	**10**	**0.01**	**0.02**	0	0.16	**0.02**	0	0.35	2.95
100-40	**10**	**0.01**	**0.05**	0	0.13	**0.05**	0	0.47	2.90
100-60	**10**	**0.00**	**0.07**	0	0.11	**0.07**	0	0.42	2.86
200-20	**10**	**0.00**	**0.05**	0	0.16	**0.05**	0	0.59	2.90
200-40	**10**	**0.01**	**0.09**	0	0.14	**0.09**	0	0.58	2.81
200-60	**10**	**0.00**	**0.14**	0	0.12	**0.14**	0	0.47	2.73
300-20	**10**	**0.00**	**0.07**	0	0.13	**0.07**	0	0.57	2.86
300-40	**10**	**0.01**	**0.14**	0	0.12	**0.14**	0	0.46	2.73
300-60	**10**	**0.01**	**0.20**	0	0.10	**0.20**	0	0.39	2.61
400-20	**10**	**0.00**	**0.09**	0	0.11	**0.09**	0	0.43	2.81
400-40	**10**	**0.01**	**0.18**	0	0.09	**0.18**	0	0.37	2.65
400-60	**10**	**0.00**	**0.25**	0	0.09	**0.25**	0	0.37	2.50
500-20	**10**	**0.01**	**0.12**	0	0.09	**0.12**	0	0.45	2.77
500-40	**10**	**0.00**	**0.21**	0	0.10	**0.21**	0	0.32	2.57
500-60	**10**	**0.01**	**0.30**	0	0.08	**0.30**	0	0.34	2.40
600-20	**10**	**0.00**	**0.14**	0	0.07	**0.14**	0	0.35	2.73
600-40	**10**	**0.01**	**0.25**	0	0.11	**0.25**	0	0.39	2.50
600-60	**10**	**0.01**	**0.35**	0	0.10	**0.35**	0	0.35	2.31
700-20	**8**	**0.00**	**0.16**	2	0.06	**0.16**	0	0.33	2.69
700-40	**10**	**0.00**	**0.28**	0	0.10	**0.28**	0	0.29	2.43
700-60	**10**	**0.01**	**0.39**	0	0.08	**0.39**	0	0.36	2.22
800-20	**7**	**0.01**	**0.18**	3	0.05	**0.18**	0	0.27	2.65
800-40	**9**	**0.01**	**0.32**	1	0.06	**0.32**	0	0.29	2.37
800-60	**10**	**0.00**	**0.43**	0	0.08	**0.43**	0	0.31	2.14
Total	**234**	**0.01**	**0.19**	6	0.10	**0.19**	0	0.40	2.63

ILS and IGA_{BR} algorithms use the same time limit, the $ARPT$ measure is similar for both, and far better than that of the solver CPLEX. The Wilcoxon signed-rank test for paired data with 95% confidence level is carried out on the RPDs of the three algorithms (see Table 6.9), and confirms the quality of the ILS algorithm.

Table 6.8 : Metrics N_{Best}, $ARPD$ and $ARPT$ for ILS, IGA$_{BR}$, and CPLEX.

Metric	Benchmark	ILS	IGA$_{BR}$	CPLEX
N_{Best}	T	**93**	34	39
	S	124	112	**186**
	L	**234**	6	0
	Sum	**451**	152	225
$ARPD$	T	**0.02**	0.08	0.20
	S	0.02	0.03	**0.02**
	L	**0.01**	0.10	0.40
	Average	**0.02**	0.07	0.21
$ARPT$	T	**0.20**	**0.20**	2.60
	S	**0.37**	**0.37**	2.27
	L	**0.19**	**0.19**	2.63
	Average	**0.25**	**0.25**	2.50

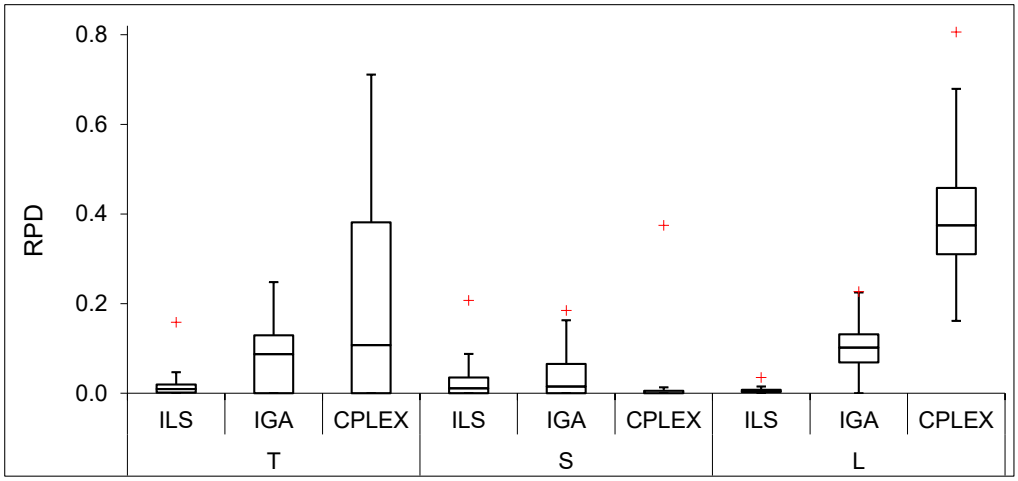

Figure 6.3 : RPDs of the ILS, IGA$_{BR}$ and the solver CPLEX.

Table 6.9 : Wilcoxon signed-rank test for RPDs of ILS, IGA$_{BR}$ and CPLEX.

Benchmark	Comparison	Wilcoxon signed-rank test	
		Test statistic	p-value
T	ILS vs. IGA$_{BR}$	606	0.000
	ILS vs. CPLEX	518	0.000
S	CPLEX vs. ILS	4993	0.005
	ILS vs. IGA$_{BR}$	4485	0.001
L	ILS vs. IGA$_{BR}$	67	0.000
	ILS vs. CPLEX	0	0.000

6.2 A relax-and-solve algorithm

We propose a relax-and-solve algorithm for the ordered flow-shop scheduling problem. For that, we implement the model proposed by Wagner (1959) (denoted by Model PF2 in the following), where binary decision variables z_{ji} take the value of 1 if job j is assigned to position $i \in J$ in the sequence, and 0 otherwise. Also, the non-negative decision variables x_{ri} and y_{ri} are utilised to represent the idle time of machine r before starting the job in position i, and the idle time of the job in position i after finishing its process on machine r, respectively.

Model PF2

$$z = \min C_{\max} = \min(\sum_{j \in J} p_{mj} + \sum_{i \in J} x_{mi}) \tag{6.12}$$

subject to

$$\sum_{j \in J} z_{ji} = 1, \quad i \in J, \tag{6.13}$$

$$\sum_{i \in J} z_{ji} = 1, \quad j \in J, \tag{6.14}$$

$$\sum_{j \in J} p_{rj} z_{j1} + x_{r1} + y_{r1} = x_{r+1,1}, \quad r \in M \setminus \{m\}, \tag{6.15}$$

$$\sum_{j \in J} p_{rj} z_{j,i+1} + x_{r,i+1} + y_{r,i+1} = \sum_{j \in J} p_{r+1,j} z_{ji} + x_{r+1,i+1} + y_{ri}, \quad i \in J \setminus \{n\}, r \in M \setminus \{m\}, \tag{6.16}$$

$$z_{ji} \in \{0,1\}, \quad i,j \in J, \tag{6.17}$$

$$x_{ri} \geq 0, \quad y_{ri} \geq 0, \quad r \in M, i \in J. \tag{6.18}$$

The objective function (eq. (6.12)) minimises the makespan. The assignment constraints (6.13) and (6.14) ensure that each position is filled with only one job and exactly one position is assigned to each job. The job-adjacency and the machine-linkage constraints presented in (6.15) and (6.16) ensure that the process of job in position i cannot be started on machine $r+1$ until its process on machine r is finished, and the process of job in position $i+1$ cannot be started on machine r, until the process of the job in position i on that same machine is finished. Constraints (6.17) and (6.18) ensure $z_{ji} \in \{0,1\}$, $x_{ri} \geq 0$ and $y_{ri} \geq 0$.

We note that in our preliminary experiments, model PF2 performed better than model PF1 for the proposed method of this study.

6.2.1 The proposed relax-and-solve method

We propose an efficient relax-and-solve (R&S) heuristic algorithm for the problem, as summarized in Algorithm 11. We utilise the pyramidal-shaped property in the develop-

ment of the neighborhoods for the proposed R&S algorithm. In the following, we discuss the components of the proposed R&S heuristic.

Algorithm 11: The R&S heuristic algorithm.

```
1 Input: The initial sequence π of performing the operations on the machines,
    parameter K.
2 k := 1;
3 while the stopping condition is not met and k ≤ K do
4 |   Apply neighborhood k (N_k) to relax π' ⊂ π;
5 |   Solve the problem by using an optimisation solver;
6 end
7 return The best obtained schedule (the solution);
```

Solution representation

As the problem under study is considered with the permutation assumption, any sequence of executing the jobs on the machine is then feasible. Therefore, we present a feasible solution by a sequence of jobs in positions 1 to n, i.e., a permutation, where job $J_{(i)}$ represents the job in position i in the sequence; see Figure 6.4.

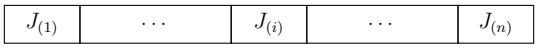

Figure 6.4 : The solution representation in the proposed R&S algorithm.

Initial solution

The pair-insert heuristic algorithm (see Algorithm 8) is utilised to generate the initial solution for the R&S algorithm.

6.2.2 Neighborhoods

The main body of the proposed R&S algorithm applies the relax operation by utilising a set of neighborhoods. We develop three neighborhoods, each of which applies a unique relaxation method that results in two relaxed subsets. Let π denote the complete sequence. From π, two subsets $\pi'_1, \pi'_2 \subset \pi$ are selected to be relaxed, and that by letting variables $z_{ji} \in \{0,1\}, \forall i \in \pi'_1 \cup \pi'_2$ (see Model PF2), i.e., letting Model PF2 decide on the values of z_{ji}. For the remaining variables, i.e., $z_{ji}, \forall i \notin \pi'_1 \cup \pi'_2$, their values are kept as they appear in π, i.e., those variables are treated as parameters. The solve operation consists of solving Model PF2 by using an optimisation solver.

Selection of the relaxed subsets π'_1 and π'_2 results in new "smaller" optimisation problems that have fewer variables and constraints than Model PF2 associated with the original

problem. In addition, changing the relaxed subsets results in different optimisation problems. An optimisation problem generated in a neighborhood is hereafter called a "sub-problem" of that neighborhood. For example, given $\pi = (J_{(1)}, \ldots, J_{(i')}, J_{(i'+1)}, \ldots, J_{(i'')}, J_{(i''+1)}, \ldots, J_{(n)})$, we generate a sub-problem where the decision variables related to $J_{(1)}$ to $J_{(i')}$ and $J_{(i''+1)}$ to $J_{(n)}$ (i.e., z_{j1} to $z_{ji'}$ and $z_{j,i''+1}$ to z_{jn} in Model PF2) are optimised. We treat the decision variables related to $J_{(i'+1)}$ to $J_{(i'')}$, i.e., $z_{j,i'+1}$ to $z_{ji''}$ as parameters.

Due to the pyramidal-shaped property, two equally-sized subsets π'_1 and π'_2 are relaxed in each neighborhood. We denote the neighborhoods by N_1, N_2 and N_3, and the size of the subsets by n_1, n_2 and n_3, associated with N_1, N_2 and N_3, where, e.g., n_1 denotes the number of jobs that are relaxed in π'_1 (and also in π'_2) in N_1. A certain number of sub-problems is solved in each neighborhood, while ensuring that any part of the sequence is subject to optimisation at least once within each neighborhood. The number of sub-problems that are solved in each neighborhood N_1, N_2 and N_3 is equal to $\lfloor \frac{n}{n_1} \rfloor$, $\lfloor \frac{n}{n_2} \rfloor$ and $\lfloor \frac{n}{n_3} \rfloor$. The mechanism of selecting π'_1 and π'_2 for each neighborhood is as follows. The order of applying the neighborhoods will be discussed in Section 6.2.3.

Neighborhood N_1 The pyramidal-shaped property of the problem is considered in the mechanism of selecting π'_1 and π'_2 in N_1. Let $S_1 = \{1, \ldots, \lfloor \frac{n}{n_1} \rfloor\}$ be the set of all sub-problems generated by N_1. Then, π'_1 and π'_2 in the sub-problem $s \in S_1$ are the sth and the $(\lfloor \frac{n}{n_1} \rfloor - s + 1)$th parts of the sequence, respectively. We note that each part of the sequence contains n_1 jobs. As an example, for $s = 1$, π'_1 and π'_2 are the first and the last n_1 jobs, respectively, that means π'_1 includes jobs 1 to n_1 and π'_2 includes jobs $n - n_1 + 1$ to n, as shown in Figure 6.5. The reason behind selecting N_1 as the first neighborhood in the algorithm is that the initial solution of the algorithm, that is obtained by the pair-insert heuristic, is a pyramidal-shaped sequence.

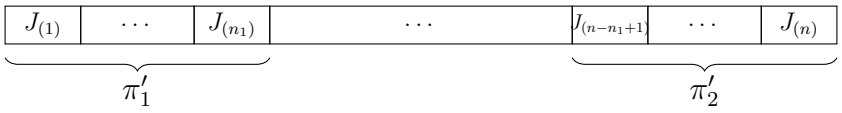

Figure 6.5 : Selection of π'_1 and π'_2 in the first sub-problem of N_1.

Neighborhood N_2 The mechanism of selecting π'_1 and π'_2 in N_2 is similar to the rolling horizon method. Let $S_2 = \{1, \ldots, \lfloor \frac{n}{n_2} \rfloor\}$ be the set of all sub-problems generated by N_2. Then, π'_1 and π'_2 in the sub-problem $s \in S_2$ are the sth and the $(s+1)$th part of the sequence, respectively. Similar to N_1, each part of the sequence contains n_2 jobs. For example, for $s = 1$, π'_1 and π'_2 are the first and the second n_2 jobs, respectively, implying π'_1 includes jobs 1 to n_2 and π'_2 includes jobs $n_2 + 1$ to $2n_2$ (see Figure 6.6).

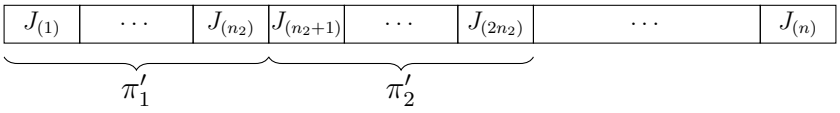

Figure 6.6 : Selection of π'_1 and π'_2 in the first sub-problem of N_2.

Neighborhood N_3 The mechanism of selecting π'_1 and π'_2 in N_3 is a disjoint version of the mechanism used in N_2. Precisely, let $S_3 = \{1, \ldots, \lfloor \frac{n}{n_3} \rfloor\}$ be the set of all sub-problems generated by N_3. Then, π'_1 and π'_2 in the sub-problem $s \in S_3$ are the sth and the $(s+2)$th part of the sequence, respectively. As in N_1 and N_2, each part of the sequence contains n_3 jobs. As an example, for $s = 1$, π'_1 and π'_2 are the first and the third n_3 jobs, respectively, implying π'_1 includes jobs 1 to n_3 and π'_2 includes jobs $2n_3 + 1$ to $3n_3$ (see Figure 6.7).

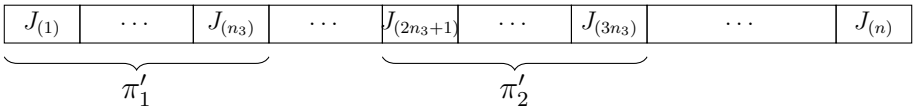

Figure 6.7 : Selection of π'_1 and π'_2 in the first sub-problem of N_3.

Stopping criterion

Two stopping criteria are considered for the algorithm. First, a time-limit criterion that works as follows is applied. Before searching each neighborhood, if the total elapsed computation time exceeds the time limit T, the algorithm stops. Otherwise, the neighborhood is completely explored. The second criterion terminates the algorithm if the algorithm goes back to a visited neighborhood with no improvement in the objective function. That is, if the best objective function is obtained in neighborhood k, and there has been no improvement with other neighborhoods, the algorithm then stops. The reason for applying this criterion is that if no improvement is gained in the neighborhoods other than k, the algorithm returns to the same neighborhood where it reached the best obtained solution. Although, because the sub-problems are heuristically solved by setting the time-limit t for the solver, finding an improving solution in the same neighborhood is unlikely.

6.2.3 Computational experiments

In this section we report the computational results of the proposed R&S algorithm on benchmark `T`. Recall that R&S includes two time limits. The first time limit, denoted by t, is for the solver Gurobi, i.e., Gurobi is allowed to spend t seconds on solving each sub-problem. We set $t = 20$ because it leads to good quality solutions in a reasonable amount of time. We observe that smaller or greater values of t led to poor quality solutions or

unnecessarily long run time for the algorithm. The second time limit, which we denote by T, is the total time limit of R&S. Because of the different instance sizes (number of jobs and machines) we consider parameter $T = 2 \times n \times T_m$, i.e., as a function of the number of jobs and machines, where $T_m = 0.8, 1.0, 1.2$ for $m = 5, 10, 20$ is a machine-dependent coefficient. We set the relaxation size of sub-problems, i.e., parameters n_1, n_2 and n_3 to 15, 20 and 15 for N_1, N_2 and N_3. We apply the neighborhoods in order N_1, N_2, N_3 and that for two times. We perform all computational experiments on the same PC mentioned in Section 3.3.

Table 6.10 summarises the performance of three solution methods of R&S, ILS and CPLEX (CPLEX, 2017). We compare the methods across three criteria of best, gap (in %), and time (sec) (see Section 3.3).

Table 6.10 : Summary of the outcomes of different solution methods.

	R&S	ILS	CPLEX
Best	51	85	39
Gap (%)	0.046	0.012	0.197
Time (seconds)	280.66	27.44	2694.25

According to Table 6.10, the performance of R&S is superior to solving Model PF2 with CPLEX, because R&S obtains a larger number of best solutions, and that with a smaller gap. Both R&S and ILS methods have small values of gap, which are less than 0.05%. While the proposed R&S method does not outperform ILS, it has a promising performance because R&S obtains the best solution in 51 of the instances. That corresponds to about 42% of instances. This very good performance along with the straightforward implementation of the proposed R&S method are among the main benefits of the R&S algorithm. With respect to the computational time, the R&S time is almost 280 seconds on average, that is small enough for real-world applications.

The instance size-wise analyses reported in Table 6.11, which shows that the performance of R&S is reliable among different sizes of the instances, and even in large instances its gap from the best solution is less than 0.09%.

Table 6.11 : Summary of the outcomes over different instance sizes.

	R&S		ILS		CPLEX	
Size	Best	Gap %	Best	Gap %	Best	Gap %
Small	38	0.031	33	0.022	39	0.029
Medium	13	0.056	42	0.002	0	0.365
Large	0	0.087	10	0.000	0	0.359

6.3 Flow-shops with coupled tasks

The flow-shop scheduling CTSP is a generalisation of no-wait flow-shop scheduling in which an exact delay exists between the consecutive tasks of each job. The flow-shop CTSP to minimise the makespan has been mostly studied for the two-machine case. Leung et al. (2007) showed that the two-machine case can be solved in $O(n \log n)$ time if all the delays are equal. However, the problem is not trivial if arbitrary delays are considered. For example, Yu et al. (2004) showed that even if only two distinct values are considered for the delays, then the problem is strongly NP-hard. This result holds even for the case with unit execution time (UET) tasks. In this chapter, we first formulate the problem as the traveling salesman problem in Section 6.3.1. In Section 6.3.2, we find the optimal makespan for the two-machine case with distinct delays and show that the problem becomes strongly NP-hard when there are more than two machines. We devote Section 6.3.3 to studying the case with ordered delays, showing that an optimal permutation schedule possesses the pyramidal-shaped property, and providing an $O(n^2)$ dynamic program to solve the general problem and an $O(n \log n)$ procedure to solve the case where the largest processing times occur on the first or the last machine. To the best of our knowledge, our research is the first attempt that studies flow-shop CTSP on an arbitrary number of machines with both distinct and ordered delays.

6.3.1 Properties of the problem

We now discuss several properties of the coupled task flow-shop scheduling problem. First, we note that the coupled task flow-shop reduces to the no-wait flow-shop if $L_{rj} = 0, \forall j \in J, r \in M \setminus \{m\}$. Second, it is easy to see that only permutation schedules, in which the processing orders of the jobs on all the m machines are the same, are feasible for the no-wait flow-shop scheduling problem, whereas in the coupled task flow-shop, non-permutation schedules can also be feasible. Therefore, an optimal schedule for the coupled task flow-shop scheduling problem may not necessarily be a permutation schedule.

Table 6.12 : Data for problem $I_{2\times2}$.

Job	p_{1j}	p_{2j}	L_{1j}
1	1	2	3
2	2	3	6

Consider an instance with two machines and two jobs denote by problem $I_{2\times2}$. Table 6.12 shows the data for problem $I_{2\times2}$. We show in Figure 6.8 the optimal makespan for problem $I_{2\times2}$, under both permutation and non-permutation schedules. It is evident that the non-permutation schedule (Figure 6.8b) yields a makespan of 11, which is better than that under the permutation schedule, i.e., 12. We have the following lemma:

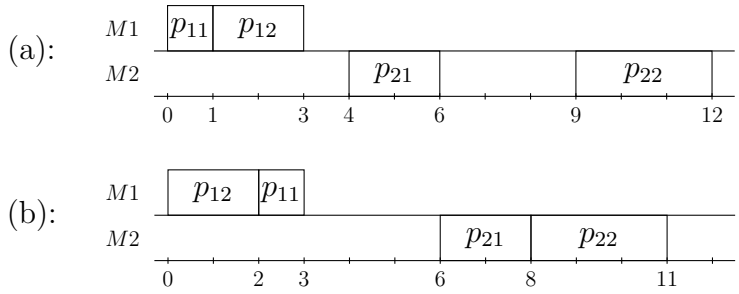

Figure 6.8 : The optimal permutation (a) and non-permutation (b) schedules for problem $I_{2\times 2}$.

Lemma 12. *An optimal schedule for the coupled task flow-shop scheduling problem is not necessarily a permutation schedule.*

Proof. See the optimal schedule for problem $I_{2\times 2}$ depicted in Figure 6.8. □

In the rest of the section we only consider permutation schedules for the coupled task flow-shop problem because the problem with non-permutation schedules is strongly NP-hard even for the case with two machines, two distinct delays, and UET tasks (Yu et al., 2004). Next, we formulate the coupled task flow-shop scheduling problem as the traveling salesman problem (TSP), and develop some properties of the problem based on the TSP formulation in Sections 6.3.2 and 6.3.3.

Baker and Trietsch (2013) showed that the n-job no-wait flow-shop scheduling problem can be formulated as an $(n+1)$-city TSP, where each city corresponds to a job and the intercity distances correspond to the delays between the jobs. In addition, one dummy city is added, from which the distances to all the other cities are zero and to which the distance from city j is the sum of the processing times of job j. We formulate the coupled task flow-shop scheduling problem as a TSP because the problem is a generalisation of the no-wait flow-shop scheduling problem.

Let D_{jk} be the delay in starting job k, measured from the starting time of job j, which is calculated by

$$D_{jk} = p_{1j} + \max\Big\{0, (p_{2j} - p_{1k} + L_{1j} - L_{1k}), (p_{2j} + p_{3j} - p_{1k} - p_{2k} + L_{1j} + L_{2j} - L_{1k} - L_{2k}), \ldots, (\sum_{r=2}^{m} p_{rj} - \sum_{r=1}^{m-1} p_{rk} + \sum_{r=1}^{m-1} L_{rj} - \sum_{r=1}^{m-1} L_{rk})\Big\}, \forall j,k \in J. \tag{6.19}$$

In Equation (6.19), the delay in starting job k after job j in the sequence is composed of two terms. The first term is the processing time of job j on the first machine since the delay is calculated from the start time of job j. The second term calculates the maximum

delay incurred on the start time of job j on machines 2 to m. In addition, let T_j be the distance from city j to the dummy city, i.e.,

$$T_j = \sum_{r=1}^{m} p_{rj} + \sum_{r=1}^{m-1} L_{rj}. \tag{6.20}$$

Then, we show in Table 6.13 the distance matrix of the corresponding TSP. From the distance matrix shown in Table 6.13, we see that a tour for the TSP corresponds to a sequence of processing the jobs on the machines. The total cost of the tour is equivalent to the makespan of the sequence.

Table 6.13 : The distance matrix of the TSP for the coupled task flow-shop problem.

-	D_{12}	D_{13}	...	D_{1n}	T_1
D_{21}	-	D_{23}	...	D_{2n}	T_2
.	.				.
.		.			.
.			.		.
D_{n1}	D_{n2}	D_{n3}	...	-	T_n
0	0	0	...	0	-

We can use a "reduced" distance matrix to simplify the TSP formulation of the coupled task flow-shop scheduling problem. To this end, we define D'_{jk} and T'_j via Equations (6.21) and (6.22) as follows:

$$D'_{jk} = D_{jk} - p_{1j}, \forall j, k \in J, \tag{6.21}$$

$$T'_j = T_j - p_{1j}, \forall j \in J. \tag{6.22}$$

Replacing D_{jk} with $D'_{jk}, \forall j, k \in J$, and T_j with $T'_j, \forall j \in J$ in Table 6.13, we obtain the reduced distance matrix. It is noted that the TSP tour remains the same under the reduced distance matrix. The optimal cost (makespan), however, is different by a constant value, which is equal to $\sum_{j=1}^{n} p_{1j}$. Next, we present our results for the coupled task flow-shop problem with distinct delays.

6.3.2 Distinct delays

Recall that the coupled task flow-shop scheduling problem with non-permutation schedules is strongly NP-hard even for the two-machine case. However, the two-machine case with permutation schedules and identical delays is polynomially solvable (Gilmore and Gomory, 1964; Leung et al., 2007). We generalise this result to the case with distinct delays and show that it is polynomially solvable.

We first transform the case with distinct delays to the equivalent no-wait case, i.e., with zero delays. We let P and P′ denote the cases with distinct and zero delays, respectively. We set the processing time of each task in P′ as the processing time of the task in P plus the amount of delay of the task. The incurred delays between the jobs, i.e., D_{jk}, are equal in both cases. First, observe that minimising the makespan for P′ is equivalent to minimising the makespan for P. Second, it is clear that minimising the makespan is equivalent to minimising the total idle time on either machine or, equivalently, minimising the total idle time on both machines (Emmons and Vairaktarakis, 2012). We let I_1 and I_2 denote the total idle times on machines 1 and 2, respectively. Third, note that irrespective of the processing times of P′, the values of idle times in both cases of P and P′ are equal.

Consider again problem $I_{2\times 2}$. In the optimal schedule depicted in Figure 6.8a, $I_1 = 9$ and $I_2 = 4 + 3 = 7$. It follows that $p'_{11} = p_{11} + L_{11} = 1 + 3 = 4, p'_{21} = p_{21} + L_{11} = 2 + 3 = 5, p'_{12} = p_{12} + L_{12} = 2 + 6 = 8$, and $p'_{22} = p_{22} + L_{12} = 3 + 6 = 9$, where p'_{rj} is the processing time of job j on machine r in case P′. We show in Figure 6.9 the schedule for P′, which is equivalent to the optimal schedule for case P.

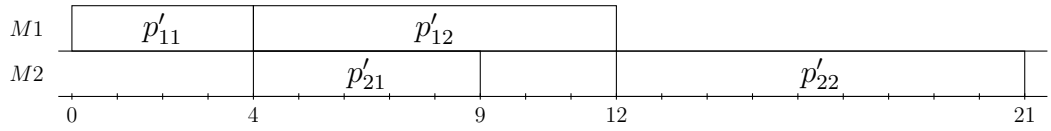

Figure 6.9 : The equivalent no-wait schedule to the optimal schedule for case P.

Gilmore and Gomory (1964) provided an $O(n \log n)$-time algorithm that finds the optimal makespan for case P′. Therefore, it suffices to transform case P to case P′, as stated in the following lemma.

Lemma 13. *The two-machine CTSP with distinct delays and permutation schedules can be transformed into an equivalent two-machine no-wait flow-shop scheduling problem.*

Proof. Let $p'_{rj} = p_{rj} + L_{1j}, \forall j \in J, r = 1, 2$. Substituting them into Equation (6.19) yields $D_{jk} = p_{1j} + \max\{0, (p'_{2j} - p'_{1k})\}, \forall j, k \in J$, which is indeed the definition of the no-wait flow-shop problem given in Baker and Trietsch (2013). As a result, the case with distinct delays and processing times p_{rj} is equivalent to the no-wait case with processing times p'_{rj}. □

Lemma 13 results in Theorem 10.

Theorem 10. *The optimal makespan for the two-machine coupled task flow-shop scheduling problem with distinct delays and permutation schedules can be obtained in $O(n \log n)$ time.*

Proof. The result is clear because the transformation is performed in constant time (see Lemma 13) and the two-machine no-wait flow-shop problem can be solved in $O(n \log n)$ time by the algorithm of Gilmore and Gomory (1964). □

We observe that Theorem 10 does not hold when there are more than two machines (see the definition in Equation (6.19)). Indeed, the problem is not trivial for $m > 2$:

Theorem 11. *The coupled task flow-shop scheduling problem with distinct delays and permutation schedules is strongly NP-hard for $m > 2$.*

Proof. We immediately deduce the result because the coupled task flow-shop problem is a generalisation of the no-wait flow-shop problem, which is strongly NP-hard for $m > 2$ as shown in Röck (1984b). □

In summary, the coupled task flow-shop scheduling problem with arbitrary processing times and distinct delays is polynomially solvable if there are two machines and permutation schedules are considered. If there are more than two machines or if non-permutation schedules are considered, the problem is strongly NP-hard. Next, we introduce an additional assumption for the processing times and delays, which leads to finding an optimal solution for any number of machines in polynomial time.

6.3.3 Ordered delays

In this section we study the coupled task flow-shop problem with the additional assumption of ordered processing times and delays. The following two conditions are satisfied for the ordered processing times assumption:

(i) for any two jobs $j, k \in J$, if $p_{rj} < p_{rk}, r \in M$, then $p_{qj} \leq p_{qk}, \forall q \in M$; and,

(ii) for any two machines $r, q \in M$, if $p_{rk} < p_{qk}, k \in J$, then $p_{rj} \leq p_{qj}, \forall j \in J$,

where (i) is called the job-ordered condition and (ii) is called the machine-ordered condition. Following the job-ordered condition, we can rank the jobs by their processing times as follows: We call job j smaller than job k, which is denoted by $j < k$, if the processing time of job k on each machine is at least as large as that of job j on the same machine. In case there are two identical jobs, we rank them by their job index.

We can show that the problem with ordered processing times and distinct delays is strongly NP-hard even for the case with two machines. The proof follows from the case with UET tasks, which satisfies the assumption of ordered processing times and is strongly NP-hard (Yu et al., 2004). If delays are identical, however, the problem is polynomially solvable (Gilmore and Gomory, 1964; Leung et al., 2007). So we consider the case with ordered delays: If job j is smaller than job k, i.e., $j < k$, then the delay of job k after completion on any machine $r < m$ is at least as large as the delay of job j after completion on the same machine. We formally state this condition as follows:

(iii) for any two jobs $j, k \in J$, if $j < k$, then $L_{rj} \leq L_{rk}, \forall r \in M \setminus \{m\}$.

Therefore, in addition to conditions (i) and (ii), the new condition (iii) ensures that the delays are ordered as well. We call this case coupled task ordered flow-shop scheduling. Observe from (iii) that the case with identical delays is a special case of ordered delays.

We present the reduced distance matrix of the TSP associated with the coupled task ordered flow-shop scheduling problem via an example, which is denoted by problem $I_{3\times3}$. Problem $I_{3\times3}$ consists of three jobs and three machines labelled as M1, M2, and M3. Table 6.14 shows the data of problem $I_{3\times3}$.

Table 6.14 : The data for problem $I_{3\times3}$.

Job	p_{1j}	p_{2j}	p_{3j}	L_{1j}	L_{2j}
1	2	1	2	1	2
2	5	3	3	3	2
3	6	4	5	4	3

As shown in Table 6.14, the jobs are ordered as $1 < 2 < 3$ and the machines are ordered as $M1 > M3 > M2$. The delays are also ordered. Table 6.15 shows the reduced distance matrix of the TSP associated with problem $I_{3\times3}$, where the values appearing in the optimal tour are highlighted. The minimum makespan is equal to 27 and the optimal job sequence is $(3, 2, 1)$, which are depicted in Figure 6.10 and are obtained by solving the TSP. We note that an instance of the TSP needs to be solved to find the optimal sequence.

Table 6.15 : The reduced distance matrix of TSP for problem $I_{3\times3}$.

-	0	0	**6**
5	-	0	11
10	**3**	-	16
0	0	**0**	-

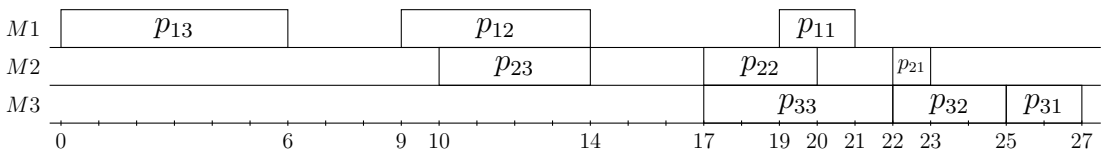

Figure 6.10 : The optimal schedule for problem $I_{3\times3}$.

Next, we derive an important property of an optimal sequence for the coupled task ordered flow-shop scheduling problem, and then we find an optimal schedule for the problem.

The pyramidal property

We show that the pyramidal-shaped property holds for an optimal sequence for the coupled task ordered flow-shop scheduling problem. The pyramidal-shaped property implies that the jobs can be partitioned into two disjoint sets, where the jobs in the first set are sequenced in the SPT order and those in the second set follow the LPT order. Without loss of generality, we assume that the largest job is in the first set. We start by showing the following lemma.

Lemma 14. *For any four jobs $j, k \in J, j < k$ and $i, l \in J$, and $i < l$, $D'_{ki} - D'_{kl} \geq D'_{ji} - D'_{jl}$.*

Proof. We represent the elements of the reduced distance matrix as follows:

$$D'_{jk} = \max\{0, \zeta_1, \ldots, \zeta_{m-1}\},$$

where $\zeta_1 = (p_{2j} - p_{1k} + L_{1j} - L_{1k})$, as defined in Equation (6.19), and so on for the rest of the elements. By substitution, we have

$$\begin{aligned} &\max\{0, A_1, A_2, \ldots, A_{m-1}\} - \max\{0, B_1, B_2, \ldots, B_{m-1}\} \geq \\ &\quad \max\{0, C_1, C_2, \ldots, C_{m-1}\} - \max\{0, E_1, E_2, \ldots, E_{m-1}\}, \end{aligned} \tag{6.23}$$

where A, B, C, and E represent the elements of $D'_{ki}, D'_{kl}, D'_{ji}$, and D'_{jl}, respectively. We call two elements "similar" if they have the same index, e.g., A_2 and B_2, and "non-similar" otherwise. It is evident that if all the four maximum terms are equal to zero, the lemma holds. The lemma also holds if in D'_{ki} (or in D'_{jl}), the maximum term is larger than zero, while the other three maximum terms are equal to zero. Therefore, we consider the case where in D'_{kl} the maximum term is larger than zero, while the other three maximum terms are equal to zero.

For that, suppose the second element is the largest, i.e., $D'_{kl} = B_1$. The assumption results in $0 - B_1 \geq 0 - 0$, meaning that the lemma holds if and only if $B_1 \leq 0$. Suppose $B_1 > 0$, which leads to $p_{2k} - p_{1l} + L_{1k} - L_{1l} > 0$. Because $i < l$, we have $p_{1i} < p_{1l}$ and $L_{1i} < L_{1l}$. Therefore,

$$\begin{aligned} &p_{2k} - p_{1l} + L_{1k} - L_{1l} > 0, \implies p_{2k} + L_{1k} > p_{1l} + L_{1l}, \\ &\implies p_{2k} + L_{1k} > p_{1i} + L_{1i}, \implies p_{2k} - p_{1i} + L_{1k} - L_{1i} > 0. \end{aligned} \tag{6.24}$$

Observe that $p_{2k} - p_{1i} + L_{1k} - L_{1i}$ is the second element, i.e., A_1 in the first maximum term. Based on our earlier assumption that the first maximum term is equal to zero, none of its elements including A_1 can be positive, so a contradiction. Similar calculations can

be performed for the other elements of D'_{kl} and D'_{ji}. Similarly, for the cases where two or three maximum terms are equal to zero, the lemma can be shown to hold.
Now, let us consider the case where all of the maximum terms in inequality (6.23) are larger than zero. We start with the case where similar elements are the largest in all the four maximum terms. For example, consider the case where the second elements are the largest in all the maximum terms. So we must show that $A_1 - B_1 \geq C_1 - E_1$. By expanding the terms, we obtain

$$\begin{aligned}(p_{2k} - p_{1i} + L_{1k} - L_{1i}) - (p_{2k} - p_{1l} + L_{1k} - L_{1l}) \geq \\ (p_{2j} - p_{1i} + L_{1j} - L_{1i}) - (p_{2j} - p_{1l} + L_{1j} - L_{1l}),\end{aligned} \tag{6.25}$$

where all the elements are cancelled out on both sides and the lemma always holds. This can also be shown for any other element. Now we consider the cases where the non-similar elements are the largest. We start with the case where the largest elements in exactly three of the terms are similar. For example, assume that B_1, C_1, and E_1 are the largest elements in their respective terms; however, A_2 is the largest element of D'_{ki}, implying that $A_2 \geq A_1$, so

$$\begin{aligned}(p_{2k} + p_{3k} - p_{1i} - p_{2i} + L_{1k} + L_{2k} - L_{1i} - L_{2i}) \geq (p_{2k} - p_{1i} + L_{1k} - L_{1i}) \\ \implies (p_{3k} - p_{2i} + L_{2k} - L_{2i}) \geq 0.\end{aligned} \tag{6.26}$$

It suffices to show that $A_2 - B_1 \geq C_1 - E_1$. Expanding the terms, we obtain

$$\begin{aligned}(p_{2k} + p_{3k} - p_{1i} - p_{2i} + L_{1k} + L_{2k} - L_{1i} - L_{2i}) - (p_{2k} - p_{1l} + L_{1k} - L_{1l}) \geq \\ (p_{2j} - p_{1i} + L_{1j} - L_{1i}) - (p_{2j} - p_{1l} + L_{1j} - L_{1l}) \\ \implies (p_{3k} - p_{2i} + L_{2k} - L_{2i}) \geq 0,\end{aligned} \tag{6.27}$$

which holds because it follows from our assumption in inequality (6.26). Considering any other largest term of A will lead to the same result. Similar calculations can be performed for D'_{jl}.
We also show that the lemma holds if A_1, C_1, and E_1 are the largest elements in their respective terms, and B_2 is the largest element in D'_{kl}. The latter leads to $(p_{3k} - p_{2l} + L_{2k} - L_{2l}) \geq 0$ following inequality (6.26). Expanding the terms, we obtain

$$\begin{aligned}(p_{2k} - p_{1i} + L_{1k} - L_{1i}) - (p_{2k} + p_{3k} - p_{1l} - p_{2l} + L_{1k} + L_{2k} - L_{1l} - L_{2l}) \geq \\ (p_{2j} - p_{1i} + L_{1j} - L_{1i}) - (p_{2j} - p_{1l} + L_{1j} - L_{1l}) \\ \implies -(p_{3k} - p_{2i} + L_{2k} - L_{2i}) \geq 0,\end{aligned} \tag{6.28}$$

which implies that either $(p_{3k} - p_{2i} + L_{2k} - L_{2i}) = 0$, where the lemma holds, or $(p_{3k} -$

$p_{2i} + L_{2k} - L_{2i}) < 0$, which is a contradiction to our assumption. Considering any other largest term of B will lead to the same result, and we can apply a similar argument for D'_{ji}. This completes the proof.
Noting that considering any other combination of the largest elements in the terms is similar to one of the aforementioned cases, we omit the proof for brevity. □

Lemma 14 leads to the following theorem.

Theorem 12. *An optimal sequence for the coupled task ordered flow-shop scheduling problem is in the pyramidal shape.*

Proof. We claim that any non-pyramidal sequence is dominated by a pyramidal one. Let π denote a pyramidal sequence for a set of n jobs, where job n is the largest job, i.e., it has the longest processing time. It is clear that $\pi = (a, \ldots, l, n, k, \ldots, b)$, where $a < \cdots < l < n$ and $n > k > \cdots > b$. Let $\pi = \pi_L \cup \pi_R$, where $\pi_L = (a, \ldots, l, n)$ and $\pi_R = (k, \ldots, b)$, i.e., we assume that job n is in π_L. Any shuffling of the job order of either π_L or π_R leads π to be a non-pyramidal sequence. Let π' present such a non-pyramidal sequence and π'_L correspond to the first part of the sequence, i.e., the sequence of the jobs up to and including job n and π'_R denotes the second part of the sequence, i.e., the sequence of the jobs after job n. We need to show that π is derivable from π' and that such a process does not increase the makespan, so the makespan of π' is never better than that of π.
In order to derive π from π', we follow the rule of Arora and Rana (1980). We select the next largest job in $\pi'_L \setminus \{n\}$ and move it to a position before a job just larger than it in the same sequence π'_L. We show that the move does not increase the makespan. Assume that, at some stage, the tour associated with π'_L is

$$S'_L = (n+1, 1, 2, \ldots, a, j, b, \ldots, c, j+1, \ldots, l, n),$$

where $n+1$ and n are the dummy city and the largest job, respectively. Also, j is the next largest job in $\pi'_L \setminus \{n\}$ and $j+1$ is the job larger than j. We denote the total distance for tour S'_L as z'_L:

$$z'_L = 0 + D'_{12} + \cdots + D'_{aj} + D'_{jb} + \cdots + D'_{c,j+1} + \cdots + D'_{ln}.$$

According to the rule, job j is moved to the position between jobs c and $j+1$. Let $z''_L = 0 + D'_{12} + \cdots + D'_{ab} + \cdots + D'_{cj} + D'_{j,j+1} + \cdots + D'_{ln}$ denote the makespan obtained as the result of the move. We need to show that $z'_L - z''_L = (D'_{aj} + D'_{jb} + D'_{c,j+1}) - (D'_{ab} + D'_{cj} + D'_{j,j+1}) \geq 0$. Either of the following two cases is possible: (1) $c > a$ or (2) $c < a$. We now show that in both cases, the makespan either improves or remains the same.

Case 1: Assume that $c > a$. Adding and subtracting D'_{cb} to and from the inequality $z'_L - z''_L = (D'_{aj} + D'_{jb} + D'_{c,j+1} + D'_{cb}) - (D'_{ab} + D'_{cj} + D'_{j,j+1} + D'_{cb}) \geq 0$ and re-arranging the terms yields

$$(D'_{cb} - D'_{cj}) + (D'_{aj} - D'_{ab}) + (D'_{jb} - D'_{j,j+1}) + (D'_{c,j+1} - D'_{cb}) \geq 0.$$

Given that $j + 1 > j, j > a, b, c$, and $c > a$ (as the underlying assumption of case 1), we have

$$D'_{cb} - D'_{cj} \geq D'_{ab} - D'_{aj}$$

and

$$D'_{jb} - D'_{j,j+1} \geq D'_{cb} - D'_{c,j+1},$$

which hold due to Lemma 14. Hence, $z'_L - z''_L \geq 0$.

Case 2: Assume that $c < a$. Again after adding and subtracting $D'_{a,j+1}$ to and from the inequality and re-arranging the terms, we have $(D'_{jb} - D'_{j,j+1}) + (D'_{a,j+1} - D'_{ab}) + (D'_{aj} - D'_{a,j+1}) + (D'_{c,j+1} - D'_{cj})$. It follows that

$$D'_{jb} - D'_{j,j+1} \geq D'_{ab} - D'_{a,j+1}$$

and

$$D'_{aj} - D'_{a,j+1} \geq D'_{cj} - D'_{c,j+1},$$

which hold due to Lemma 14, leading to $z'_L - z''_L \geq 0$. We note that similar calculations can be performed for π'_R, i.e., to the LPT part of the non-pyramidal sequence π'. As a result, any non-pyramidal sequence π' is dominated by the pyramidal sequence π. This completes the proof. □

Theorem 12 is the basis for a polynomial-time algorithm to find the minimum makespan for the coupled task ordered flow-shop scheduling that we propose in the next section.

The optimal makespan

We can find an optimal sequence for the coupled task ordered flow-shop scheduling problem in $O(n^2)$ time. In addition, we show that, under certain conditions, we can find an optimal sequence in $O(n \log n)$ time.

Theorem 13. *The minimum makespan for the coupled task ordered flow-shop scheduling problem can be found in $O(n^2)$ time.*

Proof. We prove the theorem by noting that the problem can be formulated as a TSP and that finding the shortest pyramidal tour in the TSP can be performed in $O(n^2)$ by

dynamic programming (Burkard et al., 1998; van der Veen and van Dal, 1991). The TSP tour $S = (n+1, a, \ldots, l, n, k, \ldots, b, n+1)$ associated with π is called a pyramidal tour if the distances between cities a and l are in ascending order, i.e., $a < \cdots < l$, and those between k and b are in descending order, i.e., $k > \cdots > b$. □

Next, we prove that if the first or the last machine processes the largest job, which we denote by $M_1^{\max}$ and $M_m^{\max}$, respectively, then an optimal sequence can be efficiently found in $O(n \log n)$ time.

For the proof, we first present a few properties of the problem. We sort the jobs in non-decreasing order of their processing times and re-label them accordingly.

Lemma 15. $D'_{kj} \geq D'_{k,k-1} + D'_{k-1,k-2} + \cdots + D'_{j+1,j}, \forall j, k \in J, j < k, M_1^{\max}$.

Proof. First, we consider $j = k-2$, which leads to $D'_{k,k-2} \geq D'_{k,k-1} + D'_{k-1,k-2}$. From Equations (6.19) and (6.21), we have $(p_{2k} - p_{1,k-2} + L_{1k} - L_{1,k-2}) \geq (p_{2k} - p_{1,k-1} + L_{1k} - L_{1,k-1}) + (p_{2,k-1} - p_{1,k-2} + L_{1,k-1} - L_{1,k-2}) \implies p_{2,k-1} - p_{1,k-1} \leq 0$, which is always true. It is not difficult to show that the lemma holds for other values of j. □

Lemma 16. $T'_2 - T'_1 \geq D'_{21}, M_1^{\max}$.

Proof. Using Equations (6.20) and (6.22) and expanding T'_2 and T'_1, we obtain

$$T'_2 - T'_1 = (\sum_{r=1}^{m} p_{r2} + \sum_{r=1}^{m-1} L_{r2} - p_{12}) - (\sum_{r=1}^{m} p_{r1} + \sum_{r=1}^{m-1} L_{r1} - p_{11}) =$$

$$= (\sum_{r=2}^{m} p_{r2} - \sum_{r=2}^{m} p_{r1} + \sum_{r=1}^{m-1} L_{r2} - \sum_{r=1}^{m-1} L_{r1}),$$

which is always non-negative since $\sum_{r=2}^{m} p_{r2} \geq \sum_{r=2}^{m} p_{r1}$ and $\sum_{r=1}^{m-1} L_{r2} \geq \sum_{r=1}^{m-1} L_{r1}$.

It is also clear that the right-hand side is non-negative and can be expanded as $\max\{0, (p_{22} - p_{11} + L_{12} - L_{11}), (p_{22} + p_{32} - p_{11} - p_{21} + L_{12} + L_{22} - L_{11} - L_{21}), \ldots, (\sum_{r=2}^{m} p_{r2} - \sum_{r=1}^{m-1} p_{r1} + \sum_{r=1}^{m-1} L_{r2} - \sum_{r=1}^{m-1} L_{r1})\}$.
We need to show that the left-hand-side is greater than or equal to every element inside the maximum term. We start with the term $(p_{22} - p_{11} + L_{12} - L_{11})$. As the jobs are sorted in non-decreasing order of their processing times, the left-hand-side is at least as large as that term: $(\sum_{r=2}^{m} p_{r2} - \sum_{r=2}^{m} p_{r1} + \sum_{r=1}^{m-1} L_{r2} - \sum_{r=1}^{m-1} L_{r1}) \geq (p_{22} - p_{11} + L_{12} - L_{11}) \implies (\sum_{r=3}^{m} p_{r2} - \sum_{r=3}^{m} p_{r1}) + (\sum_{r=2}^{m-1} L_{r2} - \sum_{r=2}^{m-1} L_{r1}) \geq 0$, which is always true.
We can also show that the left-hand-side is greater than or equal to the next element as well, i.e., $(\sum_{r=2}^{m} p_{r2} - \sum_{r=2}^{m} p_{r1} + \sum_{r=1}^{m-1} L_{r2} - \sum_{r=1}^{m-1} L_{r1}) \geq (p_{22} + p_{32} - p_{11} - p_{21} + L_{12} + L_{22} - L_{11} - L_{21})$. This can be written as $(\sum_{r=4}^{m} p_{r2} - \sum_{r=4}^{m} p_{r1}) + (\sum_{r=3}^{m-1} L_{r2} - \sum_{r=3}^{m-1} L_{r1}) \geq 0$, which is also always true.

Similarly, we can show that the left-hand-side is at least as large as the other terms in the maximum term of the right-hand side. This completes the proof. □

We note that Lemma 16 can be further generalised to $T'_j - T'_1 \geq D'_{j1}, 2 \leq j \leq n, M_1^{\max}$, but we do not give the proof here for brevity. Observe that Lemmas 15 and 16 hold for problem $I_{3\times3}$ and its reduced distance matrix, which is shown in Table 6.15. Lemmas 15 and 16 result in the following theorem.

Theorem 14. *The minimum makespan for the coupled task ordered flow-shop scheduling problem is obtained by the LPT sequence if* $M_1^{\max}$.

Proof. Due to the pyramidal-shaped property of the problem discussed in Theorem 12, it suffices to show that the largest job is the first job in an optimal sequence. Suppose the tour associated with the LPT sequence is

$$S = (n+1, n, n-1, \ldots, j, \ldots, 2, 1, n+1),$$

where the total distance for the tour S, denoted as z_S, is equal to

$$z_S = 0 + D'_{n,n-1} + \cdots + D'_{j+1,j} + D'_{j,j-1} + \cdots + D'_{21} + T'_1.$$

To prove that the LPT sequence is optimal, we show that any change in tour S will not improve the makespan. We first consider the case of constructing a new tour S' by inserting a job $j, \forall j \in J \setminus \{n\}$, before the largest job n. We need to show that the total distance of tour S' is at least as large as that of tour S. We consider the following two cases.

Case 1. $j > 1$: $z_{S'}$ is equal to

$$z_{S'} = 0 + D'_{jn} + D'_{n,n-1} + \cdots + D'_{j+1,j-1} + \cdots + D'_{21} + T'_1.$$

If we re-arrange the inequality $z_{S'} \geq z_S$, we obtain $D'_{jn} + D'_{j+1,j-1} \geq D'_{j+1,j} + D'_{j,j-1}$. From Lemma 15, we have $D'_{j+1,j-1} \geq D'_{j+1,j} + D'_{j,j-1}$, so the inequality holds.

Case 2. $j = 1$: $z_{S'}$ is equal to

$$z_{S'} = 0 + D'_{1n} + D'_{n,n-1} + \cdots + D'_{32} + T'_2.$$

If we re-arrange the inequality $z_{S'} \geq z_S$, we obtain $D'_{1n} + T'_2 \geq D'_{21} + T'_1$. From Lemma 16, we have $T'_2 - T'_1 \geq D'_{21}$, which means the inequality holds.

We can generalise this result by investigating the cases where more than one job are inserted before job n. Assuming there are two jobs before n, that are jobs $j, k, \forall j, k \in J \setminus \{n\}$, we construct the new tour S' with regard to three cases:

Case 1. $k > j > 1$: $z_{S'}$ is equal to

$$z_{S'} = 0 + D'_{jk} + D'_{kn} + D'_{n,n-1} + \cdots + D'_{j+1,j-1} + \cdots + D'_{j+1,j-1} + \cdots + D'_{21} + T'_1.$$

If we re-arrange the inequality $z_{S'} \geq z_S$, we obtain $D'_{jk} + D'_{kn} + D'_{k+1,k-1} + D'_{j+1,j-1} \geq D'_{j+1,j} + D'_{j,j-1} + D'_{k+1,k} + D'_{k,k-1}$. From Lemma 15, we have $D'_{j+1,j-1} \geq D'_{j+1,j} + D'_{j,j-1}$ and $D'_{k+1,k-1} \geq D'_{k+1,k} + D'_{k,k-1}$, so the inequality holds.

Case 2. $k > j = 1$: $z_{S'}$ is equal to

$$z_{S'} = 0 + D'_{1k} + D'_{kn} + D'_{n,n-1} + \cdots + D'_{k+1,k-1} + \cdots + D'_{32} + T'_2.$$

If we re-arrange the inequality $z_{S'} \geq z_S$, we obtain $D'_{1k} + D'_{kn} + D'_{k+1,k-1} + T'_2 \geq D'_{k+1,k} + D'_{k,k-1} + D'_{21} + T'_1$. From Lemma 16, we have $T'_2 - T'_1 \geq D'_{21}$, and from Lemma 15 we have $D'_{k+1,k-1} \geq D'_{k+1,k} + D'_{k,k-1}$ which means the inequality holds.

Case 3. $k = 2, j = 1$: $z_{S'}$ is equal to

$$z_{S'} = 0 + D'_{12} + D'_{2n} + D'_{n,n-1} + \cdots + D'_{43} + T'_3.$$

If we re-arrange the inequality $z_{S'} \geq z_S$, we obtain $D'_{12} + D'_{2n} + T'_3 \geq D'_{32} + D'_{21} + T'_1$. From the generalisation of Lemma 16, we have $T'_3 - T'_1 \geq D'_{31}$, and from Lemma 15 we have $D'_{31} \geq D'_{32} + D'_{21}$ which means the inequality holds.

Inserting more than two jobs will also result in the same arguments. This completes the proof. □

We note that the SPT and LPT sequences can be obtained in $O(n \log n)$ time. The following theorem shows that the SPT sequence minimises the makespan for the coupled task ordered flow-shop scheduling problem if the largest job is processed on the last machine.

Theorem 15. *The minimum makespan for the coupled task ordered flow-shop scheduling problem is obtained by the SPT sequence if* $M_m^{\max}$.

Proof. We claim that the proof similar to the those presented in Smith et al. (1975) and Panwalkar and Woollam (1979), and by noting that the case with $M_m^{\max}$ is the reverse of the case with $M_1^{\max}$, where the route for executing the jobs is reversed. From the reverse property of scheduling problems (McMahon and Burton, 1967), we see that the reverse of an optimal sequence of the case $M_1^{\max}$ obtained by the LPT rule is the SPT sequence, implying that the largest job is processed on the last machine. □

Consider the problem $I_{3\times3}$. Observe that the mirror image of the LPT sequence depicted in Figure 6.9 is the SPT sequence, which is optimal, and no job can start earlier.

To sum up, we showed that with arbitrary values for the delays, the two-machine case with permutation schedules can be transformed into the equivalent no-wait case. We demonstrated that the problem is polynomially solvable if there are only two machines, and is strongly NP-hard if there are more than two machines. We also introduced the additional assumption of ordered delays and proved that under the ordered delay assumption, an optimal sequence follows the pyramidal structure. We proposed a polynomial dynamic program to solve the case. We further showed that if the largest task of every job occurs on the first or the last machine, the LPT or SPT sequence is optimal, respectively.

Chapter 7

Concluding remarks

In this chapter, a summary of the obtained results and the limitations of this study are presented in Section 7.1, followed by future research directions presented in Section 7.2.

7.1 Limitations and Obtained results

This thesis focused on the coupled task scheduling problem (CTSP), with the main aims of exploring the computational complexity, and proposing solution methodologies for different variants of the problem. First, we conducted a comprehensive literature review of the available results, and presented a complete evaluation of the available mathematical models. We also proposed new publicly available benchmark data sets for several variants of the CTSP, to be a basis for future researches on this topic. We also discussed the real-world applications of the CTSP in detail. We then explored the computational complexity of several variants of the problem, and also proposed several algorithms to efficiently solve those variants. Our main contributions in this thesis can be summarized as following:

- Presenting a literature review of the coupled task problem (addressing aim [**i**]);
- Evaluating the available mathematical models for the single-machine and flow-shop coupled task problems (addressing aim [**i**]);
- Proposing new benchmark instances for different variants of the coupled task problem (addressing aim [**ii**]);
- Proposing a new mathematical formulation for the single-machine coupled task problem (addressing aim [**iv**]);
- Proposing two matheuristic algorithms for the single-machine coupled task problem (addressing aim [**iv**]);
- Proposing dynamic program and a heuristic algorithm for the single-machine coupled task problem with time-dependent processing times (addressing aim [**iv**]);
- Presenting NP-hardness proofs and polynomial algorithms for different variants of the parallel identical machines with coupled tasks (addressing aim [**iii**]);
- Proposing approximation bounds for two parallel identical machines with coupled tasks (addressing aim [**iii**]);

- Proposing approximation bounds for two-machine open-shop problem with coupled tasks (addressing aim [**iii**]);
- Correcting/Improving available mathematical models for the flow-shop coupled task problem (addressing aim [**iv**]);
- Proposing state-of-the-art algorithms for the ordered flow-shop problem (addressing aim [**iv**]);
- Proposing new complexity results for several variants of the flow-shop problem with coupled tasks (addressing aim [**iii**]).

Regarding the limitations of the study, aim [**v**] of the problem was to investigate how the coupled task problem can model real-world healthcare appointment scheduling problems. Pursuing that aim, we had the goal of formulating the chemotherapy appointment scheduling problem with real characteristics of the problem, e.g., the treatment time and delays for the utilised regimens, gained from an outpatient clinic. However, due to the long-term lock-down and restrictions imposed as the result of the COVID-19 pandemic, we did not fulfilled that aim.

7.2 Future research directions

Although we conducted research on several variants of the CTSP, there are still many open problems to be investigated. In the following we present three directions for future research:

- The computational complexity of the identical case of the single-machine coupled task problem, i.e., (a, L, b) is still open. As explained in Section 2.1.1, Ahr et al. (2004) and Baptiste (2010) explored this case and proposed algorithms with fixed inputs, however, it remains an open question whether the problem with arbitrary inputs is polynomially solvable (extending aim [**iii**] of this research).
- The single-machine problem under due-date related objectives is not yet explored. Resolving the computational complexity of those problems can be the initial step in that regard. Particularly, resolving the computational complexity of the single-machine problem with the objective of minimising the maximum lateness will pave the way for the other due-date related criteria, e.g., minimising the number of tardy jobs. Proposing solution algorithms for those problems will come as the next step in this direction (extending aims [**iii**] and [**iv**] of this research).
- Studying the parallel-machine CTSP is important as it can be utilised in formulating real-world healthcare appointment scheduling. This is due to the fact that many

healthcare clinics consists of a number of chairs/beds to service the patients, that can be considered as a parallel-machine setting. The variability in the characteristics of different clinics it warrants more research, specifically on formulating the problems and developing efficient solution methodologies (exploring aim [**v**] of this research).

References

Ageev, A. (2019). "Approximating the 2-machine flow shop problem with exact delays taking two values". *Journal of Global Optimization*.

Ageev, A. A. (2018). "Inapproximately lower bounds for open shop problems with exact delays". *Approximation and Online Algorithms*. Springer International Publishing AG, 45–55.

Ageev, A. A. and Baburin, A. E. (2007). "Approximation algorithms for UET scheduling problems with exact delays". *Operations Research Letters* 35(4), 533 –540.

Ageev, A. A. and Ivanov, M. (2016). "Approximating coupled-task scheduling problems with equal exact delays". *Discrete Optimization and Operations Research*. Springer International Publishing: Cham, 259–271.

Ageev, A. A. and Kononov, A. V. (2007). "Approximation algorithms for scheduling problems with exact delays". *Approximation and Online Algorithms*. Springer Berlin Heidelberg, 1–14.

Ahmadian, M. M., Salehipour, A., and Cheng, T. (2021). "A meta-heuristic to solve the just-in-time job-shop scheduling problem". *European Journal of Operational Research* 288(1), 14–29.

Ahmadian, M. M., Salehipour, A., and Kovalyov, M. (2020). "An Efficient Relax-and-Solve Heuristic for Open-Shop Scheduling Problem to Minimize Total Weighted Earliness-Tardiness". *Available at SSRN 3601396*.

Ahr, D., Békési, J., Galambos, G., Oswald, M., and Reinelt, G. (2004). "An exact algorithm for scheduling identical coupled tasks". *Mathematical Methods of Operations Research* 59(2), 193–203.

Allahverdi, A. (2016). "A survey of scheduling problems with no-wait in process". *European Journal of Operational Research* 255(3), 665 –686.

Amrouche, K. and Boudhar, M. (2016). "Two machines flow shop with reentrance and exact time lag". *RAIRO-Operations Research* 50(2), 223–232.

Amrouche, K., Boudhar, M., Bendraouche, M., and Yalaoui, F. (2017). "Chain-reentrant shop with an exact time lag: new results". *International Journal of Production Research* 55(1), 285–295.

Arabameri, S. and Salmasi, N. (2013). "Minimization of weighted earliness and tardiness for no-wait sequence-dependent setup times flowshop scheduling problem". *Computers & Industrial Engineering* 64(4), 902–916.

Arora, R. K. and Rana, S. P. (1980). "Scheduling in a semi-ordered flow-shop without intermediate queues". *AIIE Transactions* 12(3), 263–272.

Azadeh, A., Farahani, M. H., Torabzadeh, S, and Baghersad, M. (2014). "Scheduling prioritized patients in emergency department laboratories". *Computer Methods and Programs in Biomedicine* 117(2), 61–70.

Baker, K. R. and Trietsch, D. (2013). *Principles of sequencing and scheduling*. John Wiley & Sons.

Baptiste, P. (2010). "A note on scheduling identical coupled tasks in logarithmic time". *Discrete Applied Mathematics* 158(5), 583 –587.

Békési, J., Galambos, G., Jung, M. N., Oswald, M., and Reinelt, G. (2014). "A branch-and-bound algorithm for the coupled task problem". *Mathematical Methods of Operations Research* 80(1), 47–81.

Békési, J., Galambos, G., Oswald, M., and Reinelt, G. (2009). "Improved analysis of an algorithm for the coupled task problem with UET jobs". *Operations Research Letters* 37(2), 93 –96.

Benavides, A. J. and Ritt, M. (2018). "Fast heuristics for minimizing the makespan in non-permutation flow shops". *Computers & Operations Research* 100, 230 –243.

Bessy, S. and Giroudeau, R. (2019). "Parameterized complexity of a coupled-task scheduling problem". *Journal of Scheduling* 22(3), 305–313.

Blazewicz, J., Ecker, K., Kis, T., Potts, C. N., Tanas, M., and Whitehead, J. (2010). "Scheduling of coupled tasks with unit processing times". *Journal of Scheduling* 13(5), 453–461.

Blazewicz, J., Pawlak, G., Tanas, M., and Wojciechowicz, W. (2012). "New algorithms for coupled tasks scheduling - a survey". *RAIRO-Operations Research* 46(4), 335–353.

Brauner, N., Finke, G., Lehoux-Lebacque, V., Potts, C., and Whitehead, J. (2009). "Scheduling of coupled tasks and one-machine no-wait robotic cells". *Computers & Operations Research* 36(2), 301 –307.

Bürgy, R. and Gröflin, H. (2013). "Optimal job insertion in the no-wait job shop". *Journal of Combinatorial Optimization* 26(2), 345–371.

Bürgy, R. and Gröflin, H. (2017). "The no-wait job shop with regular objective: a method based on optimal job insertion". *Journal of Combinatorial Optimization* 33(3), 977–1010.

Burkard, R. E., Deineko, V. G., van Dal, R., van der Veen, J. A. A., and Woeginger, G. J. (1998). "Well-solvable special cases of the traveling salesman problem: a survey". *SIAM review* 40(3), 496–546.

Carlier, J. and Néron, E. (2003). "On linear lower bounds for the resource constrained project scheduling problem". *European Journal of Operational Research* 149(2), 314–324.

Carvalho, I. A., Noronha, T. F., Duhamel, C., Vieira, L. F., and Santos, V. F. d. (2021). "A fix-and-optimize heuristic for the minmax regret shortest path arborescence problem under interval uncertainty". *International Transactions in Operational Research*.

Chen, B. and Zhang, X. (2020). "Scheduling coupled tasks with exact delays for minimum total job completion time". *Journal of Scheduling*, 1 –13.

Chu, C. and Proth, J. (1996). "Single machine scheduling with chain: Structured precedence constraints and separation time windows". *IEEE Transactions on Robotics and Automation* 12(6), 835–844.

Condotta, A. and Shakhlevich, N. (2012). "Scheduling coupled-operation jobs with exact time-lags". *Discrete Applied Mathematics* 160(16), 2370 –2388.

Condotta, A. and Shakhlevich, N. (2014). "Scheduling patient appointments via multilevel template: A case study in chemotherapy". *Operations Research for Health Care* 3(3), 129 –144.

CPLEX, I. I. (2017). *version 12.8.0*. Armonk, New York, U.S.

Darties, B., Giroudeau, R., König, J.-C., and Simonin, G. (2016). "Some complexity and approximation results for coupled-tasks scheduling problem according to topology". *RAIRO-Operations Research* 50(4-5), 781–795.

Dell'Amico, M. (1996). "Shop problems with two machines and time lags". *Operations Research* 44(5), 777–787.

Dong, X., Huang, H., and Chen, P. (2008). "An improved NEH-based heuristic for the permutation flowshop problem". *Computers & Operations Research* 35(12), 3962 – 3968.

Dorneles, Á. P., Araújo, O. C. de, and Buriol, L. S. (2014). "A fix-and-optimize heuristic for the high school timetabling problem". *Computers & Operations Research* 52, 29–38.

Dubois-Lacoste, J., Pagnozzi, F., and Stützle, T. (2017). "An iterated greedy algorithm with optimization of partial solutions for the makespan permutation flowshop problem". *Computers & Operations Research* 81, 160 –166.

Ecker, K and Tanaś, M (2012). "Complexity of scheduling of coupled tasks with chains precedence constraints and any constant length of gap". *Journal of the Operational Research Society* 63(4), 524–529.

Elshafei, M., Sherali, H. D., and Smith, J. C. (2004). "Radar pulse interleaving for multi-target tracking". *Naval Research Logistics (NRL)* 51(1), 72–94.

Emmons, H. and Vairaktarakis, G. (2012). *Flow shop scheduling: Theoretical results, algorithms, and applications*. Vol. 182. Springer Science & Business Media.

Fernandez-Viagas, V. and Framinan, J. M. (2014). "On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem". *Computers & Operations Research* 45, 60 –67.

Fernandez-Viagas, V., Ruiz, R., and Framinan, J. M. (2017). "A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation". *European Journal of Operational Research* 257(3), 707 – 721.

Fondrevelle, J., Oulamara, A., Portmann, M.-C., and Allahverdi, A. (2009). "Permutation flow shops with exact time lags to minimise maximum lateness". *International Journal of Production Research* 47(23), 6759–6775.

França, P. M., Gendreau, M., Laporte, G., and Müller, F. M. (1995). "The m-traveling salesman problem with minmax objective". *Transportation Science* 29(3), 267–275.

Friske, M. W., Buriol, L. S., and Camponogara, E. (2022). "A relax-and-fix and fix-and-optimize algorithm for a Maritime Inventory Routing Problem". *Computers & Operations Research* 137, 105520.

Gawiejnowicz, S. (2008). *Time-dependent scheduling*. Springer Science & Business Media.

Giaro, K. (2001). "NP-hardness of compact scheduling in simplified open and flow shops". *European Journal of Operational Research* 130(1), 90 –98.

Gilmore, P. C. and Gomory, R. E. (1964). "Sequencing a one state-variable machine: A solvable case of the traveling salesman problem". *Operations Research* 12(5), 655–679.

Graham, R., Lawler, E., Lenstra, J., and Kan, A. R. (1979). "Optimization and approximation in deterministic sequencing and scheduling: A survey". *Annals of Discrete Mathematics* 5, 287 –326.

Grimes, D. and Hebrard, E. (2015). "Solving variants of the job shop scheduling problem through conflict-directed search". *INFORMS Journal on Computing* 27(2), 268–284.

Gröflin, H. and Klinkert, A. (2007). "Feasible insertions in job shop scheduling, short cycles and stable sets". *European Journal of Operational Research* 177(2), 763–785.

Gupta, J. N. D. and Gupta, S. K. (1988). "Single facility scheduling with nonlinear processing times". *Computers & Industrial Engineering* 14(4), 387 –393.

Gurobi Optimization, L. (2018). *Gurobi Optimizer Reference Manual*.

Hamdi, I. and Loukil, T. (2017). "The permutation flowshop scheduling problem with exact time lags to minimise the total earliness and tardiness". *International Journal of Operational Research* 28(1), 70–86.

Helber, S. and Sahling, F. (2010). "A fix-and-optimize approach for the multi-level capacitated lot sizing problem". *International Journal of Production Economics* 123(2), 247–256.

Huo, Y., Li, H., and Zhao, H. (2009). "Minimizing total completion time in two-machine flow shops with exact delays". *Computers & Operations Research* 36(6), 2018 –2030.

Hwang, F. J. and Lin, B. M. T. (2011). "Coupled-task scheduling on a single machine subject to a fixed-job-sequence". *Computers & Industrial Engineering* 60(4), 690 –698.

Johnson, S. M. (1954). "Optimal two- and three-stage production schedules with setup times included". *Naval Research Logistics Quarterly* 1(1), 61–68.

Kalczynski, P. J. and Kamburowski, J. (2008). "An improved NEH heuristic to minimize makespan in permutation flow shops". *Computers & Operations Research* 35(9), 3001 –3008.

Khatami, M., Oron, D., and Salehipour, A. (2021a). "Scheduling coupled tasks on parallel identical machines". *Submitted to Annals of Operations Research*.

Khatami, M. and Salehipour, A. (2019). "A simple heuristic for the coupled task scheduling problem". *MODSIM 2019*. Canberra, Australia.

Khatami, M. and Salehipour, A. (2020). "A relax-and-solve algorithm for the ordered flow-shop scheduling problem". *IEEE IEEM 2020*. Singapore.

Khatami, M. and Salehipour, A. (2021a). "A binary search algorithm for the general coupled task scheduling problem". *4OR* 19(4), 593–611.

Khatami, M. and Salehipour, A. (2021b). "Coupled task scheduling with time-dependent processing times". *Journal of Scheduling* 24, 223–236.

Khatami, M. and Salehipour, A. (2021c). "The coupled task scheduling problem: An improved mathematical program and a new solution algorithm". *Submitted to International Transactions in Operational Research*.

Khatami, M., Salehipour, A., and Cheng, T. C. E. (2020). "Coupled task scheduling with exact delays: Literature review and models". *European Journal of Operational Research* 282(1), 19 –39.

Khatami, M., Salehipour, A., and Cheng, T. C. E. (2021b). "Flow-shop scheduling with exact delays to minimize makespan". *Submitted to Computers & Industrial Engineering*.

Khatami, M., Salehipour, A., and Hwang, F. J. (2018). "Single-machine coupled task scheduling with time-dependent processing times". *ASOR 2018*. Melbourne, Australia.

Khatami, M., Salehipour, A., and Hwang, F. J. (2019). "Makespan minimization for the m-machine ordered flow shop scheduling problem". *Computers and Operations Research* 111, 400–414.

Khatami, M. and Zegordi, S. H. (2017). "Coordinative production and maintenance scheduling problem with flexible maintenance time intervals". *Journal of Intelligent Manufacturing* 28(4), 857–867.

Khorasanian, D. and Moslehi, G. (2017). "Two-machine flow shop scheduling problem with blocking, multi-task flexibility of the first machine, and preemption". *Computers & Operations Research* 79, 94 –108.

Ladhari, T. and Haouari, M. (2005). "A computational study of the permutation flow shop problem based on a tight lower bound". *Computers & Operations Research* 32(7), 1831 –1847.

Lehoux-Lebacque, V., Brauner, N., and Finke, G. (2015). "Identical coupled task scheduling: polynomial complexity of the cyclic case". *Journal of Scheduling* 18(6), 631–644.

Leung, J. Y.-T., Li, H., and Zhao, H. (2007). "Scheduling two-machine flow shops with exact delays". *International Journal of Foundations of Computer Science* 18(02), 341–359.

Li, H. and Zhao, H. (2007). "Scheduling coupled-tasks on a single machine". *IEEE Symposium on Computational Intelligence in Scheduling*, 137–142.

Lin, B. M. T., Lin, Y. Y., and Fang, K. T. (2013). "Two-machine flow shop scheduling of polyurethane foam production". *International Journal of Production Economics* 141(1), 286 –294.

Liu, M., Liu, X., Zheng, F., and Chu, F. (2017a). "Bi-objective optimization of a reentrant flow shop scheduling with exact time lag considering energy cost". *7th International Conference on Industrial Engineering and Systems Management (IESM 2017)*. Saarbrücken, Germany.

Liu, W., Jin, Y., and Price, M. (2017b). "A new improved NEH heuristic for permutation flowshop scheduling problems". *International Journal of Production Economics* 193, 21 –30.

Liu, Z., Lu, J., Liu, Z., Liao, G., Zhang, H. H., and Dong, J. (2019). "Patient scheduling in hemodialysis service". *Journal of Combinatorial Optimization* 37(1), 337–362.

Marinagi, C. C., Spyropoulos, C. D., Papatheodorou, C., and Kokkotos, S. (2000). "Continual planning and scheduling for managing patient tests in hospital laboratories". *Artificial Intelligence in Medicine* 20(2), 139–154.

Marmion, M.-E., Dhaenens, C., Jourdan, L., Liefooghe, A., and Verel, S. (2011). "NILS: a neutrality-based iterated local search and its application to flowshop scheduling". *European Conference on Evolutionary Computation in Combinatorial Optimization*, 191–202.

Martello, S., Pisinger, D., and Toth, P. (1999). "Dynamic programming and strong bounds for the 0-1 knapsack problem". *Management Science* 45(3), 414–424.

MATLAB (2018). *version 9.4.0 (R2018a)*. The MathWorks Inc.: Natick, Massachusetts, U.S.

McMahon, G. and Burton, P. (1967). "Flow-shop scheduling with the branch-and-bound method". *Operations Research* 15(3), 473–481.

McNaughton, R. (1959). "Scheduling with deadlines and loss functions". *Management Science* 6(1), 1–12.

Meziani, N., Boudhar, M., and Oulamara, A. (2018). "PSO and simulated annealing for the two-machine flowshop scheduling problem with coupled-operations". *European Journal of Industrial Engineering* 12(1), 43–66.

Meziani, N., Oulamara, A., and Boudhar, M. (2019). "Two-machine flowshop scheduling problem with coupled-operations". *Annals of Operations Research* 275(2), 511–530.

Minitab Statistical Software (2020). *State College, PA: Minitab, Inc.* Version 17.

Mitten, L. G. (1959). "Sequencing n jobs on two machines with arbitrary time lags". *Management Science* 5(3), 293–298.

Montgomery, D. C. (2017). *Design and analysis of experiments*. John wiley & Sons.

Mosheiov, G. (1994). "Scheduling jobs under simple linear deterioration". *Computers & Operations Research* 21(6), 653 –659.

Mosheiov, G., Oron, D., and Salehipour, A. (2021). "Coupled task scheduling with convex resource consumption functions". *Discrete Applied Mathematics* 293, 128–133.

Naeni, L. M. and Salehipour, A. (2019). "A new mathematical model for the traveling repairman problem". *2019 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, 1384–1387.

Nawaz, M., Enscore, E. E., and Ham, I. (1983). "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem". *Omega* 11(1), 91 –95.

Orman, A. J. and Potts, C. N. (1997). "On the complexity of coupled-task scheduling". *Discrete Applied Mathematics* 72(1), 141 –154.

Osman, I. H. and Potts, C. N. (1989). "Simulated annealing for permutation flow-shop scheduling". *Omega* 17(6), 551 –557.

Panwalkar, S. S. and Woollam, C. R. (1979). "Flow shop scheduling problems with no in-process waiting: A special case". *Journal of the Operational Research Society* 30(7), 661–664.

Panwalkar, S. S. and Woollam, C. R. (1980). "Ordered flow shop problems with no in-process waiting: Further results". *Journal of the Operational Research Society* 31(11), 1039–1043.

Pérez, E., Ntaimo, L., Malavé, C. O., Bailey, C., and McCormack, P. (2013). "Stochastic online appointment scheduling of multi-step sequential procedures in nuclear medicine". *Health care management science* 16(4), 281–299.

Pérez, E., Ntaimo, L., Wilhelm, W. E., Bailey, C., and McCormack, P. (2011). "Patient and resource scheduling of multi-step medical procedures in nuclear medicine". *IIE Transactions on Healthcare Systems Engineering* 1(3), 168–184.

Pinedo, M. (2012). *Scheduling*. Vol. 29. Springer.

Ponnambalam, S. G., Aravindan, P., and Chandrasekaran, S. (2001). "Constructive and improvement flow shop scheduling heuristics: An extensive evaluation". *Production Planning & Control* 12(4), 335 –344.

Potts, C. N. and Van Wassenhove, L. N. (1982). "A decomposition algorithm for the single machine total tardiness problem". *Operations Research Letters* 1, 177–181.

Potts, C. N. and Whitehead, J. D. (2007). "Heuristics for a coupled-operation scheduling problem". *Journal of the Operational Research Society* 58(10), 1375–1388.

Röck, H. (1984a). "Some new results in flow shop scheduling". *Zeitschrift für Operations Research* 28(1), 1–16.

Röck, H. (1984b). "The three-machine no-wait flow shop is NP-complete". *Journal of the ACM* 31(2), 336–345.

Ruiz, R. and Stützle, T. (2007). "Robust scheduling of a two machine flow shop with uncertain processing times". *European Journal of Operational Research* 177, 2033 – 2049.

Ruiz, R. and Maroto, C. (2005). "A comprehensive review and evaluation of permutation flowshop heuristics". *European Journal of Operational Research* 165(2), 479 –494.

Sahni, S. and Cho, Y. (1979). "Complexity of scheduling shops with no wait in process". *Mathematics of Operations Research* 4(4), 448–457.

Salehipour, A. (2020). "An algorithm for single- and multiple-runway aircraft landing problem". *Mathematics and Computers in Simulation* 175, 179–191.

Shapiro, R. D. (1980). "Scheduling coupled tasks". *Naval Research Logistics Quarterly* 27(3), 489–498.

Sherali, H. D. and Smith, J. C. (2005). "Interleaving two-phased jobs on a single machine". *Discrete Optimization* 2(4), 348 –361.

Simonin, G., Darties, B., Giroudeau, R., and König, J.-C. (2011a). "Isomorphic coupled-task scheduling problem with compatibility constraints on a single processor". *Journal of Scheduling* 14(5), 501–509.

Simonin, G., Giroudeau, R., and König, J.-C. (2011b). "Complexity and approximation for scheduling problem for a torpedo". *Computers & Industrial Engineering* 61(2), 352 –356.

Simonin, G. (2009). "L'impact de l'introduction du graphe de compatibilité dans les problèmes d'ordonnancement en présence de tâches-couplées". PhD thesis. Montpellier, France: Universite de Montpellier II.

Simonin, G., Giroudeau, R., and König, J.-C. (2013). "Approximating a coupled-task scheduling problem in the presence of compatibility graph and additional tasks". *International Journal of Planning and Scheduling* 1(4), 285–300.

Smith, M. L., Panwalkar, S. S., and Dudek, R. A. (1975). "Flowshop sequencing problem with ordered processing time matrices". *Management Science* 21(5), 544–549.

Smith, M. L., Panwalkar, S. S., and Dudek, R. A. (1976). "Flowshop sequencing problem with ordered processing time matrices: A general case". *Naval Research Logistics Quarterly* 23(3), 481–486.

Smith, M. L. (1968). "A critical analysis of flow-shop sequencing". PhD thesis. Texas Tech University.

Stützle, T. (1998). “Applying iterated local search to the permutation flow shop problem”. *FG Intellektik, TU Darmstadt, Darmstadt, Germany*.

Stützle, T. and Ruiz, R. (2018). “Iterated Local Search”. *Handbook of Heuristics*. Ed. by R. Martí, P. M. Pardalos, and M. G. C. Resende. Springer International Publishing: Cham, 579–605.

Taillard, E. (1990). “Some efficient heuristic methods for the flow shop sequencing problem”. *European Journal of Operational Research* 47(1), 65 –74.

Taillard, E. (1993). “Benchmarks for basic scheduling problems”. *European Journal of Operational Research* 64(2), 278 –285.

Talbi, E. G. (2009). *Metaheuristics: From design to implementation*. John Wiley & Sons.

Tseng, F. T., Stafford Jr., E. F., and Gupta, J. N. D. (2004). “An empirical analysis of integer programming formulations for the permutation flowshop”. *Omega* 32, 285 –293.

Vallada, E., Ruiz, R., and Framinan, J. M. (2015). “New hard benchmark for flowshop scheduling problems minimising makespan”. *European Journal of Operational Research* 240(3), 666 –677.

van der Veen, J. A. A. and van Dal, R. (1991). “Solvable cases of the no-wait flow-shop scheduling problem”. *Journal of the Operational Research Society* 42(11), 971–980.

Wagner, H. M. (1959). “An integer linear-programming model for machine scheduling”. *Naval Research Logistics Quarterly* 6, 131 – 140.

Watson, J.-P., Barbulescu, L., Whitley, L. D., and Howe, A. E. (2002). “Contrasting structured and random permutation flow-shop scheduling problems: Search-space topology and algorithm performance”. *INFORMS Journal on Computing* 14(2), 98–123.

Wilson, J. M. (1989). “Alternative formulations of a flow-shop scheduling problem”. *Journal of the Operational Research Society* 40(4), 395–399.

Yu, W., Hoogeveen, H., and Lenstra, J. K. (2004). “Minimizing makespan in a two-machine flow shop with delays and unit-time operations is NP-hard”. *Journal of Scheduling* 7(5), 333–348.

Zhang, X. and Van De Velde, S. (2010). “Polynomial-time approximation schemes for scheduling problems with time lags”. *Journal of Scheduling* 13(5), 553–559.