UNIVERSITY OF TECHNOLOGY SYDNEY
Faculty of Science

Efficient Solution Methods for Just-In-Time Machine and Shop Scheduling Problems

by

Mohammad Mahdi Ahmadian

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

Doctor of Philosophy

Sydney, Australia

March 2022

Certificate of Authorship/Originality

I, Mohammad Mahdi Ahmadian, declare that this thesis is submitted in fulfilment of the requirements for the award of Doctor of Philosophy, in the Faculty of Science at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

This document has not been submitted for qualifications at any other academic institution.

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

Production Note:
Signature removed prior to publication.

Date: March 29, 2022

ABSTRACT

Efficient Solution Methods for Just-In-Time Machine and Shop Scheduling Problems

by

Mohammad Mahdi Ahmadian

The classical machine (i.e. single and parallel machine) and shop scheduling (i.e. flow-shop, job-shop and open-shop) problems are concerned with performing a set of independent jobs on a given set of machines with or without precedence relations. This thesis explores variants of such problems, pertinent to the practice of Just-In-Time (JIT) manufacturing, where each job (operation) has a due date (or due window) and any deviation from it would incur either earliness or tardiness costs. Embracing JIT philosophy by companies (by discouraging late delivery and reducing warehousing and inventory costs), and their dire need for developing more realistic scheduling models have led to a growing body of research on earliness-tardiness minimization since the late 1970s. Yet, most studies have been devoted to single machine scheduling problems, and very little research has been conducted to address the multiple-machine or shop scheduling settings. Moreover, the current solution methodologies often fail to deliver quality solutions for these problems particularly as the size of instances grows. Therefore, this PhD thesis will contribute to developing efficient algorithms that are capable of obtaining high quality solutions for computationally challenging instances. In addition, we contribute to the existing approaches by integrating exact and heuristic algorithms to maximize the benefits associated with them.

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

Dedication

This thesis is dedicated to my mum Mehri Berangi who taught me to never give up.

Acknowledgements

I am extremely grateful to my supervisor, Dr. Amir Salehipour, for his invaluable advice and support during my PhD study. Without his guidance and insightful feedback this thesis would not have been possible.

I would like to extend my thanks to my co-supervisors Prof. Murray Elder and Dr. Leila Moslemi Naeni for their guidance throughout my PhD research studies and their constructive comments on earlier versions of this thesis.

Finally, I would like to thank my parents. Words cannot do justice to express my gratitude for the love and encouragement I have received from them.

Mohammad Mahdi Ahmadian

Sydney, Australia, 2022.

List of Publications

Journal Papers

J-1. **M. M. Ahmadian**, A. Salehipour and TCE. Cheng, "A meta-heuristic to solve the just-in-time job-shop scheduling problem," *European Journal of Operational Research*, 2020.

J-2. **M. M. Ahmadian** and A. Salehipour, "The just-in-time job-shop scheduling problem with distinct due-dates for operations," *Journal of Heuristics*, pp. 1-30, 2020.

J-3. **M. M. Ahmadian** and A. Salehipour, "Heuristics for flights arrival scheduling at airports." *International Transactions in Operational Research* 2020.

Conference Papers

C-1. **M. M. Ahmadian** and A. Salehipour, "A Matheuristic for Practical Flights Arrival and Departure Scheduling." *IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, pp. 1162-1166. IEEE, 2020.

Contents

List of Figures

Abbreviation

ACO - Ant Colony Optimization

B&B - Branch-and-Bound

CP - Constraint Programming

GA - Genetic Algorithm

JIT - Just-In-Time

LP - Linear Programming

MIP - Mixed Integer Programming

OM - Operations Management

PSO - Particle Swarm Optimization

R&S - Relax and Solve

SA - Simulated Annealing

TS - Tabu Search

VNS - Variable Neighborhood Search

Chapter 1

Introduction

1.1 Background

The relentless pursuit of organizations to increase market share has led to the meticulous study of internal processes often aiming at elimination of waste and enhancing productivity. In most cases such processes are identified by applying operations management (OM) practices and those adding no or little value are either eliminated or redesigned. Indeed this continual improvement of processes gained by OM allows companies to maintain competitive advantage. "Scheduling" as one of the major areas of OM plays a crucial role in fulfillment of such goals. It is concerned with determining and implementing intermediate- to short-term schedules that effectively utilize both personnel and facilities/resources while meeting customer demands (Heizer and Render, 2013). Sarin and Lefoka (1993) define manufacturing scheduling as "allocation of manufacturing resources to various jobs over time to best satisfy some criterion". Thanks to rigorous schedules, companies can have a better picture of tasks, recognize bottlenecks and handle them more efficiently to attain smoother production flow.

Just-In-Time (JIT) is a production and inventory control system aiming at reducing inventory costs by purchasing materials or manufacturing products only when they are needed. Developed by Toyota (Monden, 2011), JIT philosophy has significantly contributed to cost reduction and performance improvement in many manufacturing systems (Oliver, 1991). It attempts to address important issues including customer satisfaction. In addition to lower inventory, JIT systems may offer other benefits to companies such as reduced lot sizes, improved quality, enhanced motivation, and increased flexibility (McLachlin, 1997). These advantages have encouraged many companies to adopt JIT, including two-thirds of American manufacturers (Gao, 2018).

To adopt JIT, businesses need to look into a wide range of operations. For instance successful JIT production often requires effective JIT procurement. As a result the study of transportation operations between the suppliers and the manufacturer such as outbound deliveries and inbound shipments are of high importance (Stank and Crum, 1997). Moreover, although originally proposed for manufacturing systems, the applications of JIT go beyond production companies and being adapted for other sectors (Canel et al., 2000) including air traffic control (Beasley et al., 2000) or healthcare (Persona et al., 2008).

Implementing this philosophy has different implications on each area of OM. In scheduling research, JIT can be reflected by considering some criteria such as the (weighted) sum of earliness and tardiness. For instance, enhancing customer satisfaction can be partly achieved by reducing avoidable delays. Additionally, minimizing some function of earliness can contribute to reducing warehousing and inventory costs. The research on such models can be traced back to the late 1970s (Eilon and Chowdhury, 1977; Sidney, 1977). However, compared to other performance criteria (e.g. makespan minimization) very little research has been conducted into minimizing earliness and tardiness (Bürgy and Bülbül, 2018). Besides, solution methods already proposed often fail to deliver quality solutions for problems with such objective functions especially as the size of instances grows. As a result, in this PhD thesis, we will contribute to developing efficient algorithms for JIT machine and shop scheduling problems that are capable of obtaining high quality solutions for large instances. In particular we will investigate two JIT problems with wide application in transportation and manufacturing systems.

1.2 Research objectives

The major aim of this PhD thesis is to “develop efficient solution methods that are capable of obtaining high quality solutions for large sized instances of the machine and shop scheduling problems”. This aim can be further broken down into the following objectives:

- Developing a general and problem-independent framework to be used for a large class of problems;

- Building advanced optimization techniques with the capability of finding high quality solutions for computationally challenging instances of the problem which often cannot be efficiently solved using standard solvers within reasonable computation times;

- Equipping meta-heuristic algorithms with exact and heuristic methods in order to design efficient and more robust solution methods for the problem. Given most metaheuristics do not produce the same results every time they run, designing algorithmic frameworks with the ability to consistently converge to same (similar) quality solutions is of interest.

In general developing advanced optimization techniques and algorithms demand customized, highly efficient, and well established optimization methods. On the other hand, adding the JIT constraints to the problem further complicates the solution methods. Generally speaking, constructing a feasible schedule for these scheduling problems under the JIT environment is a highly complex process. The complexity can be both attributed to the sequencing (which takes care of the job processing order, and hence the sequence dependent setup times), and scheduling (which aims to optimize the start time of performing the jobs by minimizing earliness and tardiness penalties). Having said that, most available solvers can easily obtain good (if not optimal) solutions for small sized instances in a reasonable amount of time. Unfortunately by increasing the problem size, most solvers lose their efficiency and demand very long time to obtain reasonable solutions. As a result, many studies contributed to developing approximate sequencing and scheduling procedures. To this end, they often construct a sequence, and then attempt to schedule the given sequence. In spite of being time efficient, these approaches have two following disadvantages:

- To enhance a sequence, simple swap or remove-insert moves are applied. Although such moves can make significant improvements at the beginning of search procedure, since they mostly take place randomly, usually fail to guide the algorithm towards obtaining high quality, and preferably optimal, solutions;

- In a majority of cases, the constructed schedule is utterly simple and far from being optimal. This is because the schedule is developed for a given sequence, which may not be optimal, if the jobs are myopically positioned, i.e. without considering their subsequent effects on the rest of the sequence.

To overcome these issues, we will use available mixed integer programming models for the purpose of allocating and sequencing. But since these models include a huge number of binary variables, these variables are relaxed in the original model, and a relaxed version of model is then solved using available solvers. To enhance the sequences we not only utilize simple swapping or insertion moves but also make use of a new neighborhood structure in which the precedence constraints of a set of jobs are relaxed. This neighborhood leads to a list of jobs, which has two parts: the non-relaxed part, which includes the already allocated and sequenced jobs, and the relaxed part that includes jobs that are subject to both re-allocating and re-sequencing. The advantage of this method over the traditional manipulations is that here the moves are made with respect to other jobs, and their impacts on the rest of the sequence are well considered.

In short, this PhD thesis contributes to the research on the machine and shop scheduling problems in the following ways:

- Proposing a general framework which apart from its capability to compete with state-of-the-art methods is conceptually very simple and can be easily adapted for a large number of problem variants. We note that the state-of-the-art methods incorporate various components within the heuristics and meta-heuristics in order to surmount the difficulty arising in the sequence part. This leads to advanced solution techniques which are often very difficult to implement. We however, propose an algorithmic framework which is simple straightforward and can be simply adopted in real settings.;
- Harnessing the power of solvers for sequencing the smaller instances which has been mostly overlooked in the literature;
- Simplifying parameter tuning concerned with guiding the neighborhoods by delegating the sequencing decisions to the solver.

1.3 Problems addressed

In this thesis we focus on deterministic scheduling problems where all the information related to jobs (e.g. processing time, due date, release time) and machines (e.g. availability, breakdown) is known a priori. In the following we introduce scheduling settings discussed in this thesis.

- **Machine environment**: Suppose there is a finite set of jobs to be processed by only one machine. Machine environment is classified into two layouts, namely single machine and parallel machine. In a single machine configuration there is only one processing unit, whereas in parallel setting there are several machines with the same function (Chen et al., 1998a). Each machine can process one job at a time (aka resource constraint). A schedule is called feasible if there is no overlap in the processing of jobs on a given machine. Efforts to study JIT machine scheduling date back to as far as the late 1970s. The last decade has witnessed significant progress in developing exact methods for basic JIT machine scheduling problems (e.g. Tanaka and Fujikuma (2012)). Yet more realistic variants with sequence dependent setup times or time windows are still very challenging to solve optimally. For instance Tanaka and Araki (2013)'s exact algorithm for some instances with only 85 jobs may take more than 30 days to deliver a non-optimal solution even with 20 GB memory size. In this thesis we deal with two such problems that are Aircraft Landing Problem (ALP) and Aircraft Sequencing Problem (ASP). Both problems can be viewed as a single/parallel machine scheduling with sequence dependent setup times where the aircraft represents job and runway represents machine. The ALP is a classical scheduling problem concerned with inbound flights in which a fleet of aircraft must be sequenced and scheduled such that the total deviation from target arrival times is minimized. The ASP, also known as the runway scheduling problem, aims to improve runway utilization by optimally assigning aircraft to the runway and scheduling the departure and arrival operations of the runway such that the total (weighted) delay in landing and take-off operations is minimized.

- **Shop environment**: Differing from machine setting, in shop layout each job requires to visit several machines. The processing period for each visit is called operation. The order in which each job visits machines is called processing routes (aka precedence constraint). In flow-shop all jobs share the same processing route. On the other hand in job-shop each job has its own processing route. For open-shop the order is arbitrary. It is obvious that a schedule is feasible if satisfies both resource and precedence constraints. While adaptation of JIT manufacturing for shop layouts is almost as old as Just-in-Time philosophy itself, JIT shop scheduling research is still at an embryonic stage with respect to developing exact solution methodologies. Moreover in spite of all efforts to design efficient algorithms, due to stubborn nature of shop scheduling models even updating the upper bounds for some of the problems is still a challenge. In this context one may cite variants of the classical job-shop scheduling problem (JSS) such as Just-in-time job-shop scheduling (JIT-JSS). Introduced by Baptiste et al. (2008) in JIT-JSS each operation has a distinct due-date and any deviation of the operation completion time from its due-date incurs an earliness or tardiness penalty. We note that the upper bounds for benchmark instance of JIT-JSS have been last updated in 2014 (see Wang and Li (2014)). Even very recent studies like Bürgy and Bülbül (2018) simply overlook JIT-JSS. As a result in this thesis we attempt to design algorithms to tackle this computationally intractable variant.

1.4 Thesis organization

This thesis is organized as follows:

- *Chapter 2*: This chapter explores the literature on JIT in machine and shop scheduling settings. Major studies are highlighted and some applications of JIT systems are also investigated.

- *Chapter 3*: Two classical air traffic management problems that are related to scheduling aircraft on multiple runways are addressed in this chapter. The problems can be viewed as a single/parallel machine scheduling problem. An effective solution method based on a framework called "Relax-and-Solve" is proposed which is able to obtain high quality solutions for large instances in reasonable amounts of time.

- *Chapter 4*: This chapter is devoted to JIT-JSS. Two matheuristic algorithms are presented. To this end novel neighbourhood structures are proposed for the problem. It is shown that new neighbourhoods can explore the solution space more effectively than traditional manipulation techniques. The algorithms update the new best solutions for a large number of instances including large ones.

- *Chapter 5*: A brief summary of the thesis contents and its contributions are given in the final chapter. Recommendation for future works is given as well.

Chapter 2

Literature Survey

Part of the review presented in this chapter is based on following publication:

- M. M. Ahmadian and A. Salehipour, "Heuristics for flights arrival scheduling at airports." *International Transactions in Operational Research* (2020).

2.1 Introduction

Timely delivery of products together with reducing the holding costs are among highly valued goals delineated by firms. In make-to-order manufacturing systems, customers' satisfaction is of great importance (Fernandez-Viagas and Framinan, 2015). Indeed the late deliveries may result in contractual penalties, loss of customer goodwill or losing future bidding opportunities (Easton and Moodie, 1999). On the other hand early jobs can incur higher work-in-process or finished goods inventory levels which require more storage capacity. In this context long deviations from due dates can be interpreted as poor supply chain management. In scheduling theory such goals are often reflected by a number of performance criteria concerning earliness and tardiness of jobs.

In this chapter we review major studies to minimize earliness and tardiness. We first give a short definition of machine and shop scheduling problems (Section 2.2). Next we survey some of the papers addressing single and parallel machine scheduling problems and focus on one interesting application of such models in the context of air traffic control (Section 2.3). We then take a look at shop scheduling models (i.e. flow shop (Section 2.4), job shop (Section 2.5) and open shop (Section 2.6)) and see how JIT philosophy has been adopted for them. Finally we conclude the chapter with an overall assessment of the literature and suggest some directions for future research. It is worth mentioning part of the review given in Section 2.3 has been published in the following journal paper:

- M. M. Ahmadian and A. Salehipour, "Heuristics for flights arrival scheduling at airports." *International Transactions in Operational Research* (2020).

2.2 Problem statement

In machine scheduling layouts, there are n independent jobs ($N = \{1, \ldots, n\}$) to be processed either on single or a set $M = \{1, \ldots, m\}$ of the machines. Each job i is characterised by a processing time on machine j (i.e. p_{ij}). In shop scheduling systems (i.e. flow shop, job shop and open shop), the processing of job i is comprised of some tasks (called *operations*) each having a processing time p_{ij} on machine j. Moreover, the order in which operations of a job visit machines is called processing *route*. In a flow shop, all jobs share the same route and in a job shop each job has its own route. However, in an open shop setting the routing is immaterial.

In a JIT environment every job (or operation) has a due date (d), earliness (α) and tardiness (β) weights (costs), and any deviation of job/operation's completion time from its due date incurs an earliness or tardiness penalty. Specifically, completing job (or operation) before its due date leads to an earliness penalty (i.e. E), while completing it after the due date results in an tardiness penalty (i.e. T). The objective is to minimize the weighted sum of earliness-tardiness penalties.

According to Baker and Trietsch (2013) "a performance measure z is regular if:

(a) the scheduling objective is to minimize z, and

(b) z can increase only if at least one of the completion times in the schedule increases."

Presence of earliness costs renders most JIT objective functions to be non-regular. That is because the decrease of a completion time may result in increasing the associated earliness costs.

In the following we review major studies involving due dates and earliness-tardiness penalties for machine and shop scheduling problems. A due date refers to a single time that a job/operation should preferably be completed while a due window expresses the same notion within a time window. In addition to represent the problems, the standard three-field notation scheme is used (Graham et al., 1979).

2.3 Single and parallel machine scheduling

Research on single machine scheduling problem with the objective of minimizing (weighted) earliness-tardiness can be traced back to 70s (Eilon and Chowdhury, 1977; Sidney, 1977). For a good review of early works and results one can refer to Baker and Scudder (1990). Garey et al. (1988) showed that the problem, even with an identical weight for jobs is strongly NP-complete. Hall et al. (1991) proposed an efficient dynamic program for the common due date variant (i.e. $d_i = d$). Yeung et al. (2001) reported several pseudo-polynomial dynamic programs, as well as certain polynomial time algorithms for some special cases. Some studies have also developed approximation algorithms for the problem. Kovalyov and Kubiak (1999) were first who gave a fully polynomial approximation scheme (FPTAS) for $1|d_i = d, d \geq \sum_{i=1}^{n} p_i| \sum_i w_i(E_i + T_i)$ where $w_i = \alpha_i = \beta_i$. Later Erel and Ghosh (2008) and very recently Kellerer et al. (2018) obtained more efficient FPTASs based on reducing this problem to minimizing a half product function. Studies of Abdul-Razaq and Potts (1988); Azizoglu and Webster (1997); Li (1997); Chang (1999); Liaw (1999) have managed to deliver quality solutions for instances with up to 50 jobs by developing Branch-and-Bound (B&B). Valente and Alves (2005) proposed a decomposition based B&B in which the lower bounds are calculated by separating the problem into weighted earliness and weighted tardiness sub-problems and obtained optimal solutions for instances with up to 30 jobs. However, to produce high quality solutions for larger instances, many authors have resorted to heuristics and metaheuristics. For instance, Ow and Morton (1989) presented two dispatching rules, and a heuristic algorithm. Genetic Algorithm (GA) of Lee and Choi (1995), Evolutionary Strategy, Simulated Annealing (SA) and Threshold Accepting of Biskup and Feldmann (2005) under the restrictive common due window and distinct weighted earliness and tardiness penalties for the weighted case, Tabu search (TS) of Wan and Yen (2002), and hybrid GA with hill climbing and SA of M'Hallah (2007) for the unweighted variant are among the meta-heuristics proposed for the problem. In the last decade seminal works of Tanaka et al. (2009); Tanaka and Fujikuma (2012) made a breakthrough in solving some of the fundamental scheduling problems including minimizing (weighted) earliness-tardiness. The authors proposed dynamic programming algorithms for the general single-machine scheduling problem without/with machine idle time. They later released their algorithms as two powerful single-machine scheduling problem solvers called SiPS and SiPSi which can solve instances of problems such as $1|d_j|\sum_i(\alpha_i E_i + \beta_i T_i)$ with up to 300 jobs efficiently. They later extended their algorithm to include sequence dependent setup times i.e. $1|s_{ik}, d_i|\sum_i \beta_i T_i$ (Tanaka and Araki, 2013) which can only handle instances with up to 85 jobs.

In the parallel environments, Biskup and Cheng (1999) showed that minimizing the sum of earliness, tardiness and completion time penalties even on two identical machines is NP-hard. They investigated the polynomially solvable case, and designed a heuristic algorithm. Mason et al. (2009) proposed a mixed integer program and a heuristic for $P|d_i|\sum_i E_i + T_i$, and Kedad-Sidhoum et al. (2008) provided a time-indexed formulation of $P|d_i, r_i|\sum_i \alpha_i E_i + \beta_i T_i$ and obtained lower bounds based on the linear and Lagrangean relaxations of their formulation. Şen and Bülbül (2015) developed a preemptive relaxation for $R|d_i|\sum_i \alpha_i E_i + \beta_i T_i$ in which feasible non-preemptive schedules are constructed by job partition delivered by solution of the preemptive relaxation. Since their formulation for relaxation is computationally expensive, they used a Benders decomposition of it to handle the large instances. A number of studies have considered the problem in the presence of setup times. For instance, Balakrishnan et al. (1999) presented a mixed integer program and a Benders decomposition for the uniform machines with sequence dependent setup times (i.e. $Q|d_i, r_i, s_{ikj}|\sum_i \alpha_i E_i + \beta_i T_i$). Vallada and Ruiz (2012) studied a similar problem for unrelated parallel machines (i.e. $R|d_i, s_{ikj}|\sum_i \alpha_i E_i + \beta_i T_i$) for which they proposed a mathematical formulation. It is worthwhile to mention that lately an exact (Bulhoes et al., 2018) and also a heuristic (Kramer and Subramanian, 2017) framework have been proposed which can solve a large class of single and parallel machine problems including those with earliness-tardiness penalties. Considering a common due window Chen and Lee (2002) proposed a B&B algorithm and solved instances with up to 40 jobs. In some studies the size of the due window is also minimized alongside the costs associated with earliness and tardiness (see for example Janiak et al. (2007a) and Janiak et al. (2013)).

An application: air traffic control

It is widely believed that a relevant amount of delays experienced at the airports is often generated by an inefficient management of the runways capacity. Therefore, many studies identify airports as the bottlenecks of the air transportation system (Furini et al., 2015). The runway scheduling problem (RSP) aims to improve runway utilization by allocating the aircraft to runways and optimally determining the landing and take-off time of the aircraft so that a performance criterion is optimized, e.g., minimizing the total delays in landing and take-off operations. The RSP is known to have significant impact on other operations occurring at airports, for example, changes to the gate assignments (Zhang and Klabjan, 2017). The RSP has been referred to aircraft sequencing problem (ASP) and aircraft landing problem (ALP), though it seems there is no exact definition for ASP and ALP. For example, Furini et al. (2015) defines ASP the problem of optimally assigning an airport's runways to the arrival and departure operations (inbound and outbound traffic), as well as optimally scheduling those operations, whereas ALP is known to be dealing with only scheduling the arrival operations (inbound traffic; Beasley et al. (2000)). Following this, the ASP focuses on minimizing the delays in landing and take-off operations and does not penalize early landing and take-off operations, because it is unlikely that aircraft can depart before their estimated departure time. Samá et al. (2017)'s ASP model, which they name it aircraft scheduling problem, schedules the arrival and departure operations and takes into account the landing and take-off path in the terminal control area. The ALP's objective, however, considers both earliness and delay in the operations since aircraft can be landed before their estimated time of arrival. After all, due to safety restrictions, the inbound traffic has priority over the outbound and must therefore be managed more importantly and quickly than the outbound traffic.

The objective of the ALP is to obtain a sequence of aircraft landings together with their scheduled landing time such that early and late landings are minimized, and operational constraints are respected. The major operational constraints aim to maintain the minimum separation between every ordered pair of aircraft. Two most applied separation requirements are "radar separation", which is a lengthwise spacing of five nautical miles (some busy airports may consider less than this) and a vertical spacing of 1,000 feet, and "wake turbulence separation (wake vortex or separation time)", which is a time spacing and depends on the type of aircraft. Before operation of Airbus A380, the international civil aviation organization (ICAO) considered three categories of light (L), medium (M) and heavy (H) for wake turbulence. Airbus A380 added the fourth category of super (S). As discussed by Furini et al. (2015), other factors such as aircraft routes and weather conditions may impact separation between aircraft. Hence, the separation times may slightly vary among airports.

A number of different objectives and constraints may be considered for the ALP. One major additional operational constraint is the constrained position shifting (CPS), where each aircraft in the sequence may only deviate by a certain number of positions from its position in the first-come-first-serve (FCFS) order (in which aircraft are sequenced with respect to their arrival times to the airport). The CPS is used to model the "fairness policy" among airlines (Balakrishnan and Chandran, 2010). Some recent studies schedule the landing and take-off operations by considering the landing and take-off path in the terminal control area (see D'Ariano et al. (2015); Samá et al. (2017); Samá et al. (2018)). Aircraft must follow certain paths in the controlled airspace surrounding an airport in which they are guided by air traffic controllers. Those paths lead to additional operational constraints. The airport-dependent operational settings may also introduce additional constraints. Two important such constraints include temporary restrictions on utilizing some runways, e.g., due to wind direction, weather or congestion or simply due to airline's preference, and non-availability of some runways for certain arrivals. An example of the latter is Sydney Kingsford Smith airport in which only two of its three runways can accept Airbus A380.

It is a well-known practice that the air traffic controllers order the aircraft by using the FCFS rule, i.e., according to their arrival times to the airport. However, a considerable room for improvement is possible with the use of optimization techniques, mainly due to re-ordering the aircraft landings. This has been a motivation for development of various solution approaches for the ALP, ranging from exact methods to heuristics. For example, Bianco et al. (1999) modeled the ALP on one runway as the single machine scheduling problem with ready times and sequence-dependent setup times and the objective function of minimizing the completion times. They developed a dynamic programming algorithm. Ernst et al. (1999) proposed an exact simplex-based algorithm and a heuristic for the ALP, and obtained optimal solutions for instances with up to 44 aircraft. Beasley et al. (2000) presented a mixed-integer program (MIP) and solved instances with up to 50 aircraft to optimality. A branch-and-price algorithm was later proposed by Wen et al. (2005). They showed that the bounds, which they obtained by applying a column generation algorithm, are stronger than those obtained by the linear programming relaxation (of the MIP). Balakrishnan and Chandran (2010) proposed a dynamic programming algorithm for solving small instances of the ALP with the inclusion of CPS. By using a time discretization approach, Faye (2015) proposed exact and heuristic algorithms. His approach is based on an approximation of the separation times and discretization of the planning horizon. For many small and large instances his proposed methods deliver tighter lower bounds than those of Beasley et al. (2000). Ghoniem and Farhadi (2015) reformulated the ALP as a set partitioning problem and proposed a column generation algorithm. To solve the pricing problem, they used the solver CPLEX. Later, they proposed a branch-and-price algorithm to address the shortcomings in their earlier study. The pricing problem was now solved as the shortest path problem with time-windows and non-triangular separation times (Ghoniem et al., 2015). In spite of availability of exact algorithms for the ALP, practical and large instances still pose a computational challenge. Therefore, heuristics and meta-heuristics have also been proposed to solve the ALP.

The first efforts to use meta-heuristics can be traced back to the work of Abela et al. (1993), in which the genetic algorithm (GA) was used to solve instances with up to 20 aircraft. Hansen (2004) also used the GA for a simplified variant of the problem. Their test problems did not exceed 20 aircraft either. Pinol and Beasley (2006) adapted scatter search (SS) and bionomic algorithm (BA). By testing their algorithms on instances as large as 500 aircraft they obtained optimal solution for instances with up to 50 aircraft. Later, Yu et al. (2011) proposed a two-step algorithm, in which a feasible landing sequence is obtained in the first step and the landing times are determined by a local search procedure in the second step. Their investigation is limited to only one runway, for which superior solutions to those of Pinol and Beasley (2006) were reported. Considering the same instances, Salehipour et al. (2013), Vadlamani and Hosseini (2014), Sabar and Kendall (2015) and Girish (2016) have adapted simulated annealing (SA) and variable neighborhood search (VNS), adaptive large neighborhood search (ALNS), iterated local search (ILS) and particle swarm optimization (PSO), and obtained optimal solution for small instances, and good quality solutions for large instances. Nevertheless, only a few of those methods are able to obtain the best known solutions for large instances, due to the computational complexity of the ALP. In addition, because those algorithms include randomized components their performance may fluctuate and often deteriorates as size of the instances increases. Salehipour and Ahmadian (2017) and Salehipour et al. (2018) developed several algorithms including the variable neighborhood descent (VND) with novel relaxation neighborhoods and iterative greedy algorithms, and attempted to overcome several of the existing limitations. They reported promising solutions for the single-runway case, though they did not consider multiple runways. Certain special cases of the ALP have also been studied by Hansen (2004), Salehipour et al. (2009) and Ng et al. (2017), who investigated the arrivals and departures under uncertainty. We refer the interested reader to Bennell et al. (2011) for a comprehensive review of solution techniques for the ALP and to Ng et al. (2018) for an extensive overview and classification of meta-heuristic approaches for airside operations research, which provides details and research directions on this topic, as well as on related problems.

In practice, the schedule of the flights arrival (i.e., a feasible landing time of aircraft) typically covers a planning horizon of about one hour and is updated every few minutes. This is to ensure that the existing schedule is adjusted so that it accommodates new flights entering into the airport control area. About two to three minutes prior to landing, the schedule is frozen meaning that further changes to the landing schedule will not be considered because the aircraft is too close to the runway (Bennell et al., 2011). Following this, generating the landing schedules in a short amount of time is necessary. In addition, the quality of the delivered schedule, which is evaluated against a performance criterion, e.g., the total amount of deviations about the target arrival times, may not be overlooked due to its significant service and operational costs. Salehipour (2019) showed that considerable improvements in the schedule and also in service and operational costs are possible with the use of optimization techniques. Nonetheless, the available exact methods are unsuitable in practice because they cannot generate quality schedules quickly. For example, over the tested single-runway instances, the optimal schedule is only known for instances with up to 50 aircraft. The existing heuristic and meta-heuristics methods, on the other hand, are usually faster than the exact methods, but may not deliver quality schedules. We are only aware of one meta-heuristic that is able to outperform all existing non-exact methods (Girish, 2016). That method, includes a combination of hybrid algorithms and different neighborhood structures, meaning that its implementation requires advanced algorithmic techniques and parameters tuning.

2.4 Flow-shop

Sarper (1995) was the first who studied the earliness and tardiness objective function for flow shop. More precisely he considered two-machine flow shop with a common due date (i.e. $F2|d_i = d|\sum_i(E_i + T_i)$) and presented the mathematical formulation of the problem. He also proposed three heuristics and showed that for larger numbers of jobs the heuristic based on LPT dispatching rule outperforms other two methods. Sung and Min (2001) addressed two machine flow shop with at least one batching processing machine (BPM) with each batch having the same processing time. Moreover jobs share a common due date which is at least as late as the total processing time on the first machine. They addressed three cases of the problem: a) a discrete processing machine (DPM) is followed by a BPM b) two BPMs c) a DPM runs after a BPM. For the first two cases they propose a polynomial algorithm and since the third case is shown to be NP-Complete, they present a pseudo-polynomial algorithm. Yeung et al. (2004) studied two machine flow shop where jobs share a common due window whose size and location are known a priori (i.e. $F2|[e, d]|\sum_i(E_i+T_i)$ where e and d are earliest due date and the latest due date respectively). They show that there exists an optimal permutation schedule for the problem among other dominance properties. They also propose a heuristic based on some dominance rules which can solve near-optimally problem of 150 jobs in 20 seconds. They also present a branch-and-bound algorithm in which the initial solution is provided by their heuristic and two bounding functions base on pseudo-polynomial dynamic programming algorithm of Weng and Ventura (1996) and Johnson's rule Johnson (1954). Yoon and Ventura (2002) studied lot-streaming flow shop in which a job (lot) can be split into smaller sublots (operations) where the sublots can overlap. The study presents several mathematical programs by considering different constraints (e.g. no-wait, Limited capacity buffers). Moreover it proposes several neighborhood search mechanisms to tackle the problem heuristically.

In addition a few studies have studied earliness and tardiness performance measure for m-machine flow shop. For instance Mosheiov (2003) addressed a flow shop with unit processing times where jobs share a due date. He also considered a fairly different objective function of minimizing the maximum earliness/tardiness cost (i.e $F|p_{ij} = 1, d_i = d|\max_j(\alpha_i E_i + \beta_i T_i)$ where $\alpha_i = \beta_i$) and polynomially solved the problem for both restrictive and non-restrictive due dates. Arabameri and Salmasi (2013) investigated flow shop scheduling problem with sequence dependent setup times and no wait constraint where a job must visit the machines without any interruption ($F|s_{ij}, d_i, nwt|\max_i(\alpha_i E_i + \beta_i T_i)$). They proposed TS and PSO for the problem and showed that PSO outperforms TS for the large sized instances.

2.4.1 Permutation flow-shop

In a permutation flow shop jobs maintain the same processing sequence on all machines. Operational issues such as extra physical handling of jobs on the shop floor along with computational cost of non-permutation schedules (Schaller and Valente, 2019b) are among the main reasons for permutation flow shops being extensively studied in the literature (Ruiz and Maroto, 2005). Several authors have investigated this setting considering both earliness and tardiness penalties. Chandra et al. (2009) addressed a permutation flow shop where jobs share a common due date (i.e. $F|d_i = d, prmu|\sum_i(E_i + T_i)$) for which they developed a heuristic based on results for single machine unrestricted common due date problem. They divided the problem to 3 cases according to the value of common due date: 1) unrestricted 2) restricted 3) the due date is such that all jobs are tardy; and showed that the first case can be solved optimally by extending the results for single machine while for cases 2 and 3 they developed a heuristic in which a sequence and schedule is obtained based on bottleneck machine and is improved by local search. In some manufacturing systems due to expensive setup costs or limited availability of machines the insertion of idle time must be avoided unless next job to be processed is not ready. This type of idle time called forced idle time and has been considered by some studies. Zegordi et al. (1995) studied the permutation flow shop with weighted earliness-tardiness objective function (i.e. $F|d_i, prmu|\sum_i(\alpha_i E_i + \beta_i T_i)$) for which they proposed a simulated annealing equipped with the problem specific knowledge. More precisely in order to pair exchange, a measure called "priority index" is calculated which indicates the desirability of shifting a job forward and backward in a given sequence. Their computational tests suggest the superiority of proposed SA over the formal annealing heuristics. Later Madhushini et al. (2009) proposed a B&B algorithm for a wide range of permutation flow shops including sum of weighted flowtime, weighted tardiness and weighted earliness of jobs (i.e. $F|d_i, prmu|\sum_i(\gamma_i F_i + \alpha_i E_i + \beta_i T_i)$). In their algorithm the bounding function is job based with respect to weighted flowtime and weighted tardiness and machine based for weighted flowtime and weighted tardiness and obtained by solving the assignment problem. Schaller and Valente (2013b) studied $F|d_i, prmu|\sum_i(E_i+T_i)$ and developed a GA for the problem. By comparing the outcomes their algorithm with those of existing heuristic in the literature including Zegordi et al. (1995) they concluded that GA consistently generates solutions with a lower total earliness and tardiness than the other procedures tested. Later M'Hallah (2014a) presented a mathematical model of the problem and proposed a VNS which outperforms Schaller and Valente (2013b)'s GA and updates the upper bounds for 70% of instances. Considering the same problem, Fernandez-Viagas et al. (2016) proposed a constructive heuristic where jobs are appended to the partial sequence based on index which is updated iteratively for unscheduled jobs according to their idle time, completion time, earliness and tardiness. They also embedded the sequence delivered by constructive heuristic in local search methods and proposed several composite heuristics. In a separate study, Schaller and Valente (2013a) considered permutation flow shop with forced idle time in the presence of family set up times where jobs are classified to families according to their similarities and a set up time is required only two jobs from different families are processed one after the other (i.e. $F|d_i, prmu, familysetup|\sum_i(E_i + T_i)$). They proposed six heuristics from literature based on neighbourhood searches, variable greedy algorithms and genetic algorithms. Based on the computational results a genetic algorithm armed with job insertion and batch insertion local searches delivers better results for large sized instances. In spite of technological justification, restricting idle time to only forced one is at odds with earliness and tardiness objective (Sarper, 1995).

That is because inserting unforced idle time can possibly improve the objective by increasing the completion time of some early jobs. As a result some papers have considered the cases in which the machine can be kept idle upon availability of jobs. M'Hallah (2014b) studied permutation flow shop in which inserting unforced idle time is allowed (i.e. $F|d_i, prmu|\sum_i(E_i+T_i)$) and proposed an algorithm which combines VNS and mixed integer programming (MIP) that is the VNS obtains job sequence, and a mixed integer program delivers optimal idle times for the given sequence. For this problem, Schaller and Valente (2019b) modified several dispatching heuristics used for unforced idle time. Finally very recently Schaller and Valente (2019a) presented a B&B algorithm for two machine permutation flow shop with unforced idle times (i.e. $F2|d_i, prmu|\sum_i(E_i+T_i)$). To this end they presented two lower bounds based in the results for single machine (Schaller, 2007) and proved a number of dominance conditions for the problem. Based on these conditions they developed four B&B algorithms and tested their performance for instances with at most 30 jobs. The computational results suggest that a B&B with a node to represent a post partial sequence outperforms other methods.

2.4.2 Hybrid flow-shop

In a hybrid flow shop there are m stages with at least one containing more than one parallel machines. A small number of studies have attempted to adopt JIT philosophy for this more realistic manufacturing environment. Indeed only 1% of papers reviewed in Ruiz and Vázquez-Rodríguez (2010)'s comprehensive literature review of hybrid flow shop, deal with earliness and tardiness and more often than not due to the complexity of this setting most of the solution methodologies are restricted either to heuristics or meta-heuristics. Fakhrzad and Heydari (2008) studied $HF|d_i|\sum_i(\alpha_i E_i+\beta_i T_i)$ and proposed a three layer heuristic in which jobs are allocated to the machine, next several ordinary flow shop problems (i.e. with only one machine at each stage) are solved and finally resource levelling is performed by utilizing the remaining resources. The unweighted version of this problem (i.e. $HF|d_i|\sum_i(E_i+T_i)$) was studied by Han et al. (2015) for which they propose a dynamic co-evolution compact genetic algorithm. In their algorithm the population is represented as a probability distribution over the set of solutions and to avoid premature convergence a strategy called individual inheritance is applied. In order to minimize the inventory costs related to partially completed as well as finished products, Janiak et al. (2007b) considered an objective function comprised of three parts: the total weighted earliness, the total weighted tardiness and the total weighted waiting time (i.e. $HF|r_i, d_i|\sum_i(\alpha_i E_i + \beta_i T_i + \gamma_i W_i)$ where W_i denotes the total waiting time of the job i). They use a decomposition approach to solve the problem. For the timing subproblem they propose an approximation algorithm which delivers optimal schedules if $\gamma_i = 0$ for each $i \in N$. They also present three metaheuristics based on SA and TS to construct and manipulate sequences. By solving instances containing up to 20 and 10 stages, they showed that SA has better performance in terms of solution quality and computation time. Jolai et al. (2009) studied no-wait hybrid flow shop (or flexible flow lines) where each job has a due window and an ideal due date. Due to resource scarcity (i.e. machines) it may happen that some jobs are rejected. Associated with each job is a profit, earliness penalty and tardiness penalty and the objective is to schedule the jobs so the gained profit is maximized. They proposed a three phase GA for the problem and managed to solve instance with at most 50 jobs and 5 stages efficiently. Yan et al. (2014) studied a two stage flow shop where each job consists of $m + 1$ operations. The first m operations are unrelated and must be performed on m parallel dedicated machines in the first stage and finally assembled in the second stage by a single machine. The objective is the minimization of maximum makespan, maximum earliness and maximum tardiness (i.e. $\alpha E_{max} + \beta T_{max} + \gamma C_{max}$). For this problem they proposed a hybrid variable neighbourhood search – electromagnetism-like mechanism (VNS-EM) algorithm in which VNS is applied to improve the best particle of each generation. Their computational results suggest the superiority of VNS-EM over VNS and EM. As noted by Sabuncuoglu and Lejmi (1999), assuming only one point as the due date is unrealistic and in most manufacturing systems a due window is considered for the completion of each job, that is jobs completed before or after the window are regarded early and tardy respectively. Therefore recently Pan et al. (2017) studied hybrid flow shops with due windows to minimize weighted earliness and tardiness (i.e. $HF||[d_i^-, d_i^+]|\sum_i(\alpha_i E_i + \beta_i T_i)$) for which they proposed an Iterated Local Search and Iterated Greedy procedures.

Sequence-dependent setup times

There are only few studies minimizing earliness and tardiness in hybrid flow shop setting in the presence of sequence-dependent setup times. Behnamian et al. (2010a) studied hybrid flow shop with the presence of non-anticipatory sequence-dependent setup times between jobs at each stage i.e. the setup for a given job on stage t cannot be started until all the jobs on stage $t-1$ are finished. To minimize the sum of earliness and tardiness for jobs they proposed a hybrid algorithm consists of ACO, SA and VNS. In the proposed algorithm the initial solution is generated by ACO and improved by VNS/SA local search. In addition Behnamian and Zandieh (2013), examined the same objective function for a hybrid flow shop with sequence-dependent setup times and with position-dependent processing times where the processing time of each job at stage t may decrease due to positional learning effect. They proposed a hybrid algorithm in which VNS is used for intensification while SA and PSO are utilized to attain diversification. Behnamian et al. (2010b) investigated a hybrid flow shop in which jobs are processed in groups. Only processing two different groups of l and k at stage t requires setup times and it is sequence dependent (i.e. s^t_{lk}). Moreover each job has a $[d_{i1}, d_{i2}]$ due window where completion of i before d_{i1} or after d_{i2} incurs earliness and tardiness penalties respectively. The problem is to find the sequence of jobs belonging to a group as well as the order of groups on each machine to minimize the sum of earliness and tardiness of jobs. They proposed a hybrid metaheuristic based on PSO, VNS and SA in which PSO is used for exploration of solution space while VNS/SA-based local search is utilized to perform exploitation. Khare and Agrawal (2019) extended work of Behnamian et al. (2010a) and Pan et al. (2017) by considering sequence-dependent setup time and due windows for hybrid flow shop. The authors presented few metaheuristics to tackle the problem.

2.5 Job-shop

Job shop with earliness-tardiness pentalties was first considered in the late 1980s (Fox and Smith, 1984; Sadeh and Fox, 1990). Later Beck and Refalo (2001), and Beck and Refalo (2003) formally introduced the earliness-tardiness job shop scheduling (ETSP) problem where each job has earliness and tardiness costs and a due date (i.e. $J|d_i|\sum_i(\alpha_i E_i + \beta_i T_i)$). They proposed a hybrid approach using constraint programming and linear programming in which the information obtained during the search process is exchanged between these two solution techniques to achieve better results. They presented four approaches namely: Probe, CRT-All, ProbePlus and CRT-Root where the latter three are just modifications of Probe. Being a backtracking algorithm and used in a B&B framework, at each node Probe approach 1) constraint propagation techniques are executed to infer new constraints and domain reductions; 2) a linear relaxation of ETSP resulting from removing resource constraints and adding obtained domains reduction and constraint propagation is solved; 3) using the optimal start times obtained from step 2, new nodes are branched in case of exceeding (violating) resource constraints or B&B is continued by backtracking to the parent node. As in ETSP only last operation of each job incurs earliness or tardiness costs, at each node even a cost relevant subproblem (CRS) containing such operations can provide a lower bound for ETSP. Motivated by this fact the second variant (i.e. CRT-All) is created by adding two scheduling problems to the Probe procedure. In particular at each node first CRS is solved and then it is checked whether it the start times delivered by CRS can be extended to a global solution. Since solving these two scheduling problems can be time consuming, ProbePlus and CRS-Root are designed to solve CRS problems less frequently. By evaluating their algorithm on two sets of benchmarks, they showed that solving cost relevant subproblems can be beneficial while doing this at each node does not contribute much to a better performance. In an attempt to adapt local search techniques applied to makespan optimization to ETSP, Beck and Refalo (2002) also proposed a hybrid local search (HLS) algorithm by combining TS with LP in which only adjacent operations on a same machine lying on the critical path are swapped. Given a complete sequence of operations on each machine, HLS evaluates a generated neighbour by assigning optimal start time to each operation by solving the timing subproblem for ETSP. They reported slightly worse solutions than those reported in Beck and Refalo (2001). Danna et al. (2003) investigated three MIP heuristics for ETSP which are used together with a MIP solver such as CPLEX. The first heuristic called local branching defines a neighbourhood of a given incumbent solution by allowing at most r binary variables to be different in current and neighbour solutions. The created MIP (called sub-MIP) is then solved to explore better neighbours in the vicinity of incumbent solution. Differing from local branching, in relaxation induced neighbourhood search (RINS) the neighbourhood is defined by fixing a subset of variables to their values in the incumbent solution. Finally guided dives instead of defining neighbourhoods, steers the tree traversal to the region which are close to the incumbent solution. Testing their heuristics on the same benchmarks used by Beck and Refalo (2003), they showed that RINS outperforms other heuristics. In a separate study Danna and Perron (2003) also showed that when large neighbourhoods are embedded in constraint programming, they provide quality solutions. Another constraint programming based algorithm for ETSP was proposed by Kelbel and Hanzálek (2007); Kelbel and Hanzálek (2011). Inspired by the fact that only last operation of each job influences the objective value, they presented a new search procedure called cost directed initialization (CDI) for ETSP to explore the search space. Contrary to the ranked based procedure in which the search tree is constructed by assigning the values to variables in the increasing order, CDI 1) ranks the variables associated with the completion time of the last operations (i.e. C_{i,n_i} in the increasing size of their domains 2) and assigns a value to C_{i,n_i} so that it incurs the lowest earliness and tardiness cost for job i. It should be noted that CDI is only applied once and for other possible values of the variables ranking procedure is applied. According to the computational results, CP armed with CDI obtains better results than those of RINS. Contrary to due dates which can be violated, the deadlines must be respected that is each job must be completed by its given deadline. Yang et al. (2012b) studied another variant of ETSP in which each job i apart from its due date has a deadline $\bar{d}_i$ and must be completed in $[d_i, \bar{d}_i]$ (i.e. $J|[d_i, \bar{d}_i]| \sum_i(\alpha_i E_i + \beta_i T_i)$). They proposed a GA for the problem in which each chromosome is scheduled using forward and backward scheduling and chromosomes violating the deadlines undergo a repair strategy. Benchmarking against the MIP model, it is shown that GA outperforms MIP in all instances. The two machine case with a common due date (i.e. $J2|d_i = d| \sum_i(E_i + T_i)$) has also been studied by Al-Salem et al. (2016) for which they proposed a dynamic programming algorithm. In the aforementioned studies, jobs (and not operations) either have distinct due dates or share a common due date. Thus the earliness-tardiness penalties are also considered for each job (and not operation).

Another variant called just-in-time job shop scheduling (JIT-JSS) was introduced by Baptiste et al. (2008), in which each operation has a distinct due date, and earliness and tardiness weights (i.e. $J|d_{ij}|\sum_i \sum_i(\alpha_{ij}E_{ij}+\beta_{ij}T_{ij})$). To tackle the problem they used Lagrangian relaxation and obtained lower bounds. Traditionally there are two types of relaxations for job shop problems namely: 1) relaxing machine (resource) constraints 2) relaxing precedence constraints. Baptiste et al. (2008) compared the efficiency of these two relaxation techniques for JIT-JSS. In the case of relaxing precedence constraints, the resulting problem can be decomposed into m single machine scheduling sub-problems with earliness and tardiness penalty costs which can be solved using Sourd and Kedad-Sidhoum (2003)'s B&B. In order to relax the resource constraints they first propose a time index formulation of JIT-JSS and show that each sub-problem created by this relaxation merely involves scheduling of operations of a given job based on their precedence order for an arbitrary objective function which can solved by dynamic program discussed in Chen et al. (1998b). It is noteworthy that in both cases Lagrangian dual problem is solved by a standard subgradient procedure. According to their results the resource constraints relaxation deliver better lower bounds when the number of machines is large enough. While JSS literature is very rich on heuristics and local search methods, it is sparse on exact procedures primarily due to the inherent complexity and intractability of the problem. One of the few exact methods is due Lancia et al. (2011). The authors proposed two time indexed formulations for JSS with a min sum objective. The first formulation called BP is an ILP model with column generation with each column being a scheduling pattern of the operations of a given job. BP is solved by an ad hoc branch and price algorithm. The second model named CBP is based on network flow in which flows correspond to the schedules of single operations. CBP is solved in a branch and cut framework or directly using an MIP solver. Applying CBP for Baptiste et al. (2008) instance with up to 50 jobs and solving it as a standalone ILP by CPLEX, Lancia et al. (2011)'s results suggest that CPLEX has obtained the optimal values for instances with 20 and 30 operations but failed to prove the optimality. Also lower bounds delivered by CBP are tighter than those of Baptiste et al. (2008). Later Tanaka et al. (2015) introduced another time indexed formulation for JSS with a min sum objective in which precedence constraints among operations of a job are stated by their starting time. To obtain lower bounds for their proposed formulation they applied a new Lagrangian relaxation technique which integrate relaxing of resource and precedence constraints. In other words Lagrangian dual problem contains subproblems resource and precedence relaxations. To do this they duplicated the binary variables in their time indexed formulation. It is worth mentioning that the new formulation with duplicated variables is conceptually the same as formulating JSS by parallel dedicated machines introduced by Brucker et al. (1999). They evaluated their proposed relaxation on Baptiste et al. (2008) instance with 10 jobs with up to 100 operations. Although their results demonstrate significantly tighter lower bounds than those of Baptiste et al. (2008), the computation times reported are very long sometimes in excess of 7000 seconds for some instances. Large neighbourhood search (LNS) (Shaw, 1998) has been proved to be a very effective tool for solving hard combinatorial optimization problems. By hybridizing local search and CP/MIP, at each iteration LNS fixes a subset of variables and re-optimizes the rest by CP or MIP Carchrae and Beck (2009). Laborie and Godard (2007) proposed a self-adapting LNS (SA-LNS) for JIT-JSS. Given a set of LNs and completion strategies (CS), at each iteration of their algorithm a LN_i and CS_i are chosen and the relaxed solution is re-optimized based on CS_i. If this combination produces good solutions then the associated selection probabilities for LN_i and CS_i are increased. Applying their algorithm they managed to update the upper bounds for some instances of JIT-JSS with 15 and 20 jobs. By introducing a global constraint which makes use of both upper bound and lower bound obtained from relaxing the resource constraints Monette et al. (2009) proposed a more efficient CP for JIT-JSS. They also improved the performance of CP by a simple local search which is run each time a new solution is found by branch and bound. Laborie and Rogerie (2016) presented a new relaxation called temporal linear relaxation (TLR) for CP models which is a linear relaxation of temporal and assignment constraints. Testing TLR on some classical scheduling problems, they showed that this relaxation does not improve the performance of CP optimizer for JIT-JSS. Heuristics and meta-heuristics have also been applied to tackle JIT-JSS. Araujo et al. (2009) developed a genetic algorithm (GA) for the problem, which Dos Santos et al. (2010) later extended by designing a hybrid method, in which an evolutionary algorithm is used to explore the sequences and a mathematical programming model is used to find the optimal schedule for a given sequence. Their algorithm obtains quality solutions for benchmark instances of JIT-JSS. Yang et al. (2012a) proposed a GA, where each chromosome is processed through a three-stage decoding mechanism.

For instances with 10 and 15 jobs, they reported that their method outperforms those in the previous studies. Wang and Li (2014) applied an approach similar to that of Dos Santos et al. (2010). They used a variable neighbourhood search (VNS) algorithm to explore the sequences and a mathematical programming model to optimally schedule the jobs in a given sequence. Their algorithm outperforms the previous ones and obtains new best solutions for several instances of JIT-JSS.

One shortcoming of scheduling literature in general and job shop in particular is existence of different problem definitions and absence of a unified solution methodology being capable of handling a large class of problems (Bülbül and Kaminsky, 2013). Aiming at generality a few number of papers have attempted to present algorithms which can address a large class of job shop scheduling problems including ones with earliness-tardiness objectives. Gélinas and Soumis (2005) proposed a Dantzig-Wolfe decomposition for job shop with a min max objective where jobs do not necessarily visit all machines. To model the problem they assumed each operation has a time window to process which is updated (tightened) at each iteration of the algorithm. In their formulation the precedence constraints are kept in the master problem while the machine constraints and the time window constraints are relegated to the column generation subproblems which are single machine problems with time windows to minimize a peicewise linear objective function of completion times (i.e. $1|[e_u, \grave{d}_u]|\sum f_u(C_u)$). As a weaker version of precedence constraints are retained in the master problem as a result the proposed decomposition provides a lower bound for the job shop. Authors imbedded their formulation in a B&B framework and tested their algorithm for a JIT objective in which the maximum earliness and tardiness incurred by operations is minimized (i.e. min $g_{max} \geq |C_u - d_u|, u \in I$ where I is set of all operations). It is shown that the algorithm is efficient particularly for instances containing many jobs but few operations per job. Bülbül and Kaminsky (2013) proposed an iterative based decomposition method which generalizes bottleneck heuristic to solve job shop scheduling problems. At each iteration each unscheduled machine is solved as a subproblem and one which hurts the overall objective of already scheduled machines the most is selected as bottleneck. One advantage of this method is that it utilizes the information by the (partial) timing problem to specify the parameters of single machine subproblems which are solved at each iteration. They applied their algorithm for a job shop problem whose objective has two components: 1) intermediate holding costs and 2) a function of completion times (comprised of total weighted earliness-tardiness and makespan) and showed that their heuristic is competitive with existing solution methodologies. Bürgy and Bülbül (2018) recently presented an overarching formulation of JSS called JS-CONV with convex costs to which many existing variants of the problem with linear and non-linear objective functions can be mapped. Given a feasible sequence of jobs on each machine turns into an integer program with a convex function (the timing problem). It is shown that timing problem can be transformed to a linear program and efficiently solved using solution approach of Ahuja et al. (2003). By generalizing the notion of criticality to JS-CONV, Bürgy and Bülbül (2018) showed that cost of each feasible sequence S is only determined by critical arcs. As such to improve S, at least one critical arc in S to be swapped. Let $N(S)$ denote neighborhood containing all the neighbors generated by swapping critical arcs in S. They proved that all the neighbors in $N(S)$ are feasible and that $N(S)$ is opt-connected i.e. starting from S there is a finite array of neighbors which leads to optimal sequence. Applying swap neighborhood they proposed a tabu search with the timing algorithm at its heart. They employed the tabu search both for linear and non-linear JIT objective functions containing earliness, tardiness and storage costs and reported quality solutions.

Flexible job-shop

The flexible job-shop scheduling is an extension of classical job-shop in which at least one machine can process a given operation. Only few studies have investigated this problem for minimization of earliness and tardiness. Huang et al. (2013) considered a due window for each job where an early (tardy) penalty is only incurred if job's completion time is earlier (later) than its earliest (latest) due date (d_i^e ((d_i^t)). Moreover a sequence dependent setup time s_{lij} occurs between jobs l and i if l precedes i in station j. This problem can be denoted as $fJ|d_i = [d_i^e, d_i^t], s_{lij}| \sum_i(\alpha E_i + \beta T_i)$. They proposed a heuristic called two-pheromone ant colony (2PH-ACO) in which a certain number of ants seek the best route. Differing from traditional ACO, in 2PH-ACO a novel global pheromone updating rule is adopted to encourage the following ants to stay close to the best route. To this end the amount of pheromone intensifies at a certain node with the number of ants choosing this node in a particular order.

Their computational results suggest that 2PH-ACO outperforms traditional ACO. Motivated by mould fabrication process, Gomes et al. (2013) studied a flexible job shop problem with non-identical parallel machines at each station in which jobs may visit a certain machine or a set of machines more than once (aka reentrent process in the literature). They presented a mixed integer program in which the earliness and tardiness of orders alongside intermediate storage time are minimized. To better capture the make-to-order nature of mould-making process they propose a reactive scheduling algorithm to update the existing schedule upon arrival of a set of new orders. More specifically mip is first solved for old orders and the initial solution is obtained. Assuming that new orders arrive at time t_i, a modified MIP is then created to ensure that the new set operations cannot start before insertion time (i.e. t_i) and is solved for both old and new orders. When solving the modified MIP, certain variables for old orders are either fixed or kept free based on their values in the initial solution in relation to the insertion point. In remanufacturing environments the used products are rebuilt and renovated to function at good as or even better than new products. Introduction of remanufacturing jobs may cause interruption and result in non-started operations of existing jobs to be rescheduled. Gao et al. (2015) examined insertion of remanufacturing jobs for the flexible job shop setting to minimize the average earliness and tardiness (i.e. $\frac{\sum_{i \in N} |C_i - d_i|}{n}$) and proposed a group of heuristics for both scheduling and rescheduling.

2.6 Open-shop

The open-shop scheduling problem has been the least studied shop scheduling model with earliness-tardiness penalties. Lin (1998) was first to study the open shop scheduling problem with earliness and tardiness costs in which each job is characterized by a distinct due date, earliness and tardiness costs (i.e. $O|d_i| \sum_i (\alpha_i E_i + \beta_i T_i)$). He proposed an $O(n^3 m^2)$ heuristic based on spanning tree which uses an idle time rule, whenever a machine is freed, to schedule a tardy operation with minimum idle time or delay early unscheduled operations to the next level. The superiority of the proposed heuristic is established by comparing its results with those of obtained from four simple dispatching rules on instances as large as 5×25. For the same problem Doulabi et al. (2012) developed a simulated annealing algorithm with three swap-based neighborhoods for generating sequences, and proposed two mixed integer programs, i.e., sequence and position-based models, in order to determine job start times. Their method solved instances with up to 50 operations. Lauff and Werner (2004) investigated the complexity of open shop problems with a common due date to minimize the total earliness and tardiness with and without intermediate storage costs. While minimization of total earliness and tardiness for m-machine open shop with a non-restrictive due date (i.e. $Om|d_i = d| \sum_i (E_i + T_i)$) is polynomially solvable (Kubiak et al., 1990), they showed that two-machine open shop with storage costs for $d = 0$ as well as for a non-restrictive due date is strongly NP-hard.

2.7 Conclusion

In this chapter we reviewed some of the studies on minimization of earliness and tardiness for both machine and shop scheduling settings. As discussed owning to the complexity of the problems, many authors have resorted to heuristics or meta-heuristics as an alternative to exact methods to address large instances. In this context simple manipulation techniques are used to generate and improve sequences. On the other hand matheuristics as a new breed of algorithms made by the inter-operation of heuristics and mathematical programming techniques (Boschetti et al., 2009; Maniezzo et al., 2009) have been vastly overlooked and very little research has been done on them. Therefore in this PhD project we aim at matheuristic methods to address some of the drawbacks of existing solution methodologies. We believe exploiting recent advances on mathematical programming techniques will enable us to design robust and time effective heuristics for JIT machine and shop scheduling problems.

From scientific prospective, we attempt to propose algorithms which unlike the most studies in the extant literature operate by breaking down the original instance of the problem into smaller instances at the master level, and solving the smaller instances at the slave level, by optimizing a reduced formulation through applying an exact solver. From a practical point of view, we contribute to the literature by designing conceptually simple methods which contrary to the state-of-the-art methods utilizing various components within the heuristics and meta-heuristics, can be easily adapted for a wide class of hard optimization problems. Also, by using an exact solver inside our algorithmic framework and harnessing its power for sequencing the smaller instances which has been mostly overlooked in the literature we commit to using advanced and established methods. In short while the existing metaheuristics go to the trouble of guiding the neighborhoods to generate better sequences which requires meticulous parameter tuning, we offer algorithms which delegate the sequencing decision to the solver.

Chapter 3

Just-In-Time Single and Parallel Machine Scheduling

This chapter is based on following publications:

- M. M. Ahmadian and A. Salehipour, "Heuristics for flights arrival scheduling at airports." *International Transactions in Operational Research* (2020).
- M. M. Ahmadian and A. Salehipour, "A Matheuristic for Practical Flights Arrival and Departure Scheduling." *IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, pp. 1162-1166. IEEE, 2020.

3.1 Introduction

The classical parallel-machine scheduling problem is to assign n independent jobs to m machines. This chapter explores extended variants of this problem in which each job has a time interval to be finished (called due window), and machines require setup times (preparation), which are dependent on the sequence on each machine. Embracing Just-In-Time philosophy and the need for developing more realistic scheduling models are among the reasons cited for considering setup times and due dates (windows) simultaneously.

We propose efficient algorithms that are capable of delivering high quality schedules. Unlike most studies in the extant literature, our algorithms are matheuristic methods and operate by breaking down the original instance of the problem into smaller instances at the master level, and independently solving the smaller instances at the slave level through optimizing a compact formulation by an exact solver. From a practical point of view, one may acknowledge the conceptual simplicity of our algorithms as one of their merits. While the state-of-the-art methods utilize various components within the heuristics and meta-heuristics, leading therefore to implementation of advanced algorithmic techniques and parameters tunning, the structure of our matheuristics are both simple and straightforward. Also, by using an exact solver inside our algorithms as the local search, not only we contribute to a simpler framework, we also commit to using advanced and established solvers.

It is noteworthy that problems discussed in this chapter apart from manufacturing systems have certain applications in air traffic control. We address two such problems i.e. Aircraft landing problem (ALP) and Aircraft Sequencing Problem (ASP). The ALP on single and multiple runways aims to schedule a set of aircraft for landing in a given planning horizon, and no changes to this set (i.e., removal or addition of aircraft or runways) is permitted during the planning. As the input data, we are given minimum separation time units between every pair of aircraft, target landing times of aircraft, as well as their earliest and latest landing times and the penalties per unit of earliness and lateness. On the other hand the ASP is concerned with sequencing both inbound and outbound aircraft on a single runway. We propose two Relax-and-Solve (R&S) matheuristic algorithms called Relax_1 and Relax_2 for ALP and ASP respectively. The remainder of this chapter is organized as follows. Section 3.2 explains the ALP and ASP, defines the mathematical notations and formulates the problems as an MIP. Section 3.3 discusses the proposed R&S matheuristic algorithm (i.e Relax_1) for ALP. The components of the algorithm, including initial solution generation and improvement procedures will also be discussed in this section. In addition, a relaxed MIP is developed, which together with the presented speed-up procedures significantly improve the efficiency of solving the problem. The computational outcomes will be discussed in Section 3.3.6. Section 3.4 describes Relax_2 for ASP and provides the computational results of testing the Relax_2 on benchmark instances. Finally, the chapter ends with a few conclusions. Relax_1 and Relax_2 have previously appeared in the following publications:

- Relax_1 (for ALP (Section 3.3)):
  - M. M. Ahmadian and A. Salehipour, "Heuristics for flights arrival scheduling at airports." *International Transactions in Operational Research* (2020).

- Relax_2 (for ASP (Section 3.4)):
  - M. M. Ahmadian and A. Salehipour, "A Matheuristic for Practical Flights Arrival and Departure Scheduling." *IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, pp. 1162-1166. IEEE, 2020.

3.2 Problem statement

The ALP aims to determine an optimal allocation of a fleet of aircraft $I = \{1, \ldots, n\}$ to land on the airport's runways (a landing sequence), and also an optimal schedule of landings simultaneously. Although the target landing time of aircraft $i \in I$, i.e., T_i is given a priori, the scheduled or real landing time x_i, also known as the scheduled time of arrival (STA), is an operational-dependent variable which must be decided upon. This implies that an aircraft may not land on its target landing time or even close to this time. We assume that all runways are identical and accept all types of aircraft.

Minimizing the total cost of early and delayed landings about the given target landing times is the objective of the ALP. If aircraft i lands earlier than its target landing time T_i, i.e., $x_i \leq T_i$, its landing is penalized proportionally to the amount of earliness, which is $\alpha_i = \max(0, T_i - x_i)$. Conversely, if aircraft i is scheduled to land later than its target landing time, i.e., $x_j \geq T_j$, the penalty of its late landing is proportional to the amount of its delay, which is $\beta_i = \max(0, x_i - T_i)$. Per unit cost of early and late landing of aircraft i is given by parameters $c_i^- \geq 0$ and $c_i^+ \geq 0$. The total cost of early and late landings is therefore equal to $\sum_{i=1}^{n}(c_i^- \alpha_i + c_i^+ \beta_i)$. Considering different per unit cost for early and late landings generalizes the modeling since these two costs may not be identical in practice. Also, in practice, the target landing time of aircraft i may be selected from the time window $[E_i, L_i]$, which defines the earliest and latest landing times. Therefore, a feasible aircraft landing must be scheduled in this time window, that is $E_i \leq x_i \leq L_i$. It follows that $\alpha_i > 0$ if the decision variable x_i lies within the range $[E_i, T_i)$, and $\beta_i > 0$ if it lies within the range $(T_i, L_i]$.

A safe landing requires a separation time $s_{ij} \in \mathbb{R}^+, i, j \in I, i \neq j$ between every pair of ordered aircraft i and j landing on the same runway. Several factors, including the type of aircraft, impact the separation times; typically, the choice of the runway does not affect the separation times. Moreover, the separation time between every pair of ordered aircraft landing on the same runway is different from landing on different runways. We assume that the separation time between two aircraft landing on different runways is zero time unit. This assumption has been made by other authors in the literature, for example, see Pinol and Beasley (2006).

The ALP can be formulated as an MIP. Problem P1 presents the mathematical model for ALP (Pinol and Beasley, 2006; Salehipour et al., 2013). In problem P1, the non-negative variables $x_i \geq 0, \forall i \in I$ represent the scheduled landing time of aircraft i. Also, the non-negative variables $\alpha_i, \beta_i \geq 0, \forall i \in I$ show the amount of earliness and tardiness of aircraft i. The binary variables $y_{ij}, \forall i, j \in I, i \neq j$ take the value of 1 if aircraft i lands before j and 0 otherwise. The binary variables $\delta_{ij}, \forall i, j \in I, i \neq j$ are introduced to model if aircraft i and j land on the same runway, for which the variables take the value of 1, and 0 otherwise. The binary variables $\gamma_{ir}, \forall i \in I, r \in R$ take the value of 1 if aircraft i lands on runway r and 0 otherwise. In problem P1, M represents a very large constant, value of which may be obtained per instance as discussed in Beasley et al. (2000). Table 3.1 summarizes the mathematical notations used in problem P1.

Problem P1

$$\min z = \sum_{i \in I} (c_i^- \alpha_i + c_i^+ \beta_i) \tag{3.1}$$

Subject to

$$E_i \leq x_i \leq L_i, \quad \forall i \in I, \tag{3.2}$$

$$x_i - T_i = \alpha_i - \beta_i, \quad \forall i \in I, \tag{3.3}$$

$$x_j - x_i \geq s_{ij}\delta_{ij} - My_{ji}, \quad \forall i, j \in I, i \neq j, \tag{3.4}$$

$$y_{ij} + y_{ji} = 1, \quad \forall i, j \in I, i \neq j, \tag{3.5}$$

Table 3.1 : The mathematical notations.

Sets	
I	Set of aircraft, $I = \{1, \ldots, n\}, \lvert I \rvert = n$, indexed by i.
R	Set of runways, $R = \{1, \ldots, m\}, \lvert R \rvert = m$, indexed by m.
Parameters	
s_{ij}	Separation time units between two ordered aircraft i and j landing on the same runway, $s_{ij} > 0, i, j \in I, i \neq j$.
T_i	Target landing time of aircraft i, $T_i \geq 0, i \in I$.
E_i	Earliest landing time of aircraft i, $E_i \geq 0, i \in I$.
L_i	Latest landing time of aircraft i, $L_i \geq E_i, i \in I$.
c_i^-	Cost of early landing of aircraft i, $c_i^- \geq 0, i \in I$.
c_i^+	Cost of late landing of aircraft i, $c_i^+ \geq 0, i \in I$.
Variables	
x_i	Scheduled landing time (STA) of aircraft i, $x_i \geq 0, i \in I$.
α_i	Amount of landing earliness of aircraft i (landing before target landing time), $\alpha_i = \max(0, T_i - x_i)$, $\alpha_i \geq 0$.
β_i	Amount of landing lateness of aircraft i (landing after target landing time), $\beta_i = \max(0, x_i - T_i)$, $\beta_i \geq 0$.
y_{ij}	Whether aircraft i lands before aircraft j, $y_{ij} \in \{0, 1\}, i, j \in I, i \neq j$.
δ_{ij}	Whether aircraft i and j land on the same runway, $\delta_{ij} \in \{0, 1\}, i, j \in I, i \neq j$.
γ_{ir}	Whether aircraft i is allocated to runway r, $\gamma_{ir} \in \{0, 1\}, i \in I, r \in R$.

$$\delta_{ij} \geq \gamma_{ir} + \gamma_{jr} - 1, \quad \forall i, j \in I, i \neq j, r \in R, \tag{3.6}$$

$$\sum_{r \in R} \gamma_{ir} = 1, \quad \forall i \in I, \tag{3.7}$$

$$y_{ij}, \delta_{ij} \in \{0, 1\}, \quad \forall i, j \in I, i \neq j, \tag{3.8}$$

$$\gamma_{ir} \in \{0, 1\}, \quad \forall i \in I, r \in R, \tag{3.9}$$

$$x_i, \alpha_i, \beta_i \geq 0, \quad \forall i \in I. \tag{3.10}$$

The objective function (Equation (3.1)) minimizes the total cost of landing deviations about the target landing times. Constraints (3.2) ensure that every aircraft lands in its time window. Constraints (3.3) link the decision variables x_i and parameters T_i to decision variables α_i and β_i. Constraints (3.4) ensure that if two aircraft i and j land on the same runway, at least s_{ij} time units should be elapsed before aircraft j could be landed on that runway. Given a set of two aircraft, constraints (3.5) ensure that one lands before the other. Constraints (3.6) link the decision variables δ_{ij} and γ_{ir}. Constraints (3.7) imply that every aircraft lands on only one runway. Constraints (3.8) and (3.9) force decision variables y_{ij}, δ_{ij} and γ_{ir} to take only binary values. Finally, constraints (3.10) impose non-negativity for decision variables x_i, α_i and β_i.

As discussed earlier, the ASP includes inbound and outbound traffic. Moreover, contrary to ALP, the aircraft sequencing problem (ASP) aims to schedule the landing and take-off operations on a single runway such that the total weighted delays of all aircraft are minimized. Hence one can formulate ASP by slightly modifying Problem P1 and eliminating decision variables E_i, δ_{ij} and γ_{ir}.

3.3 Relax_1 for ALP

In this section, we propose a R&S matheuristic algorithm (i.e. Relax_1) for the ALP. Matheuristic algorithms are made by the inter-operation of heuristics and mathematical programming techniques (Boschetti et al., 2009; Maniezzo et al., 2009), and have been adapted for a wide range of optimization problems (Doi et al., 2018; Fuentes et al., 2018; Woo and Kim, 2018).

Given an initial sequence for aircraft landings, the R&S algorithm, which is also known as fix-and-optimize (Helber and Sahling, 2010), delivers an improved sequence by iteratively destructing (relaxing) a sub-sequence of the incumbent sequence, which includes a subset of consecutive aircraft, and re-constructing a feasible sequence by using optimization techniques. Indeed, the "relax" part nominates a subset of aircraft to change their position in the landing sequence, whereby the "solve" part determines a (new) landing order for the aircraft in the subset, and obtains a complete landing schedule for all aircraft. Algorithm 3.1 shows a high-level presentation of the R&S algorithm.

We use the solver CPLEX (ILOG, 2017) as the local search in the solve part of the proposed Relax_1. Our algorithm is conceptually simple, and we will show in Section 3.3.6 that it also delivers quality solutions for the ALP. Next, we discuss the solution representation, initial sequence generation, relax and solve operations and the speed-up techniques.

Algorithm 3.1: The relax-and-solve (R&S) matheuristic algorithm for the ALP.

```
Input: An initial sequence Π (an ordered set of aircraft) for landing.
while the stopping condition is not met do
    Relax();
    Solve();
end
return The best obtained landing schedule;
```

3.3.1 Generating an initial sequence

We show a sequence for the ALP by an ordered list of $2 \times n$ elements. The first row gives the landing positions in the sequence and the second row specifies the runway allocations. For example, the schedule illustrated in Table 3.4 with $m = 2$ can be represented as follows:

$$\Pi = \begin{pmatrix} 3 & 4 & 5 & 6 & 8 & 7 & 9 & 10 & 1 & 14 & 13 & 2 & 12 & 11 & 15 \\ 1 & 1 & 1 & 2 & 1 & 2 & 1 & 2 & 1 & 2 & 2 & 1 & 1 & 2 & 1 \end{pmatrix}.$$

We generate such an initial sequence of landings (which includes a landing sequence and runway allocations) for the Relax_1 by utilizing the earliest target landing time (ETLT) construction algorithm of Salehipour et al. (2013); (see Algorithm 3.2). The ETLT generates an initial sequence by sorting aircraft in non-decreasing order of their target landing times (i.e., by following the FCFS dispatching rule), and assigning the runways accordingly. Next we present the pseudocode of the ETLT algorithm and a numerical example of ETLT. It is noteworthy that ETLT has the running time of $O(n \log n)$.

The ETLT algorithm

The ETLT algorithm (Salehipour et al., 2013) is summarized in Algorithm 3.2.

A numerical example

Consider the instance Airland2 that includes 15 aircraft. Table 3.2 illustrates the landing data for this instance. The separation time between every pair of aircraft landing on the same runway is shown in Table 3.3. Table 3.4 represents the landing

Algorithm 3.2: The earliest target landing time (ETLT) construction heuristic.

Input: An instance of the ALP, where I and R are sets of aircraft and runways, $\omega = \{\}$ and $\Pi_{2\times n} = \{\}$.
Output: A feasible landing sequence Π for the ALP.

Initialization:
$r := 1$ (selecting the first runway);
$k := 1$;
Let $\omega := (\sigma(1), \sigma(2), \ldots, \sigma(n))$, where $T_{\sigma(1)} \leq T_{\sigma(2)} \leq ... \leq T_{\sigma(n)}$, be the sorted sequence of aircraft landings (sorted in non-decreasing order of target landing times);
$\Pi[1][k] := \omega[1]$; // The first aircraft (i.e., $\omega[1]$) appears first in Π
$\Pi[2][k] := r$; // Assign the first aircraft (i.e., $\omega[1]$) to runway 1
Remove the first element from ω;

Allocation:
while $\omega \neq \{\}$ **do**
    $k := k + 1$;
    if $|\omega| = 1$ **then**
        $\Pi[1][k] := \omega[1]$ and $\Pi[2][k] := r$;
        Remove the first element from ω;
    else
        if $T_{\omega[2]} < T_{\omega[1]} + s_{\omega[1],\omega[2]}$ **then**
            // Assign the first two aircraft in ω to two different runways
            $\Pi[1][k] := \omega[1]$ and $\Pi[2][k] := r$;
            $k := k + 1$;
            $r := r + 1$;
            if $r > m$ **then**
                $r := 1$;
            end
            $\Pi[1][k] := \omega[2]$ and $\Pi[2][k] := r$;
            Remove the first two elements from ω;
        else
            // Assign the first two aircraft in ω to the same runway
            $\Pi[1][k] := \omega[1]$ and $\Pi[2][k] := r$;
            $k := k + 1$;
            $\Pi[1][k] := \omega[2]$ and $\Pi[2][k] := r$;
            Remove the first two elements from ω;
        end
        $r := r + 1$;
        if $r > m$ **then**
            $r := 1$;
        end
    end
end
return Π;

sequences generated by the ETLT (Algorithm 3.2) on one and two runways and their schedule delivered by the CPLEX.

Table 3.2 : The landing data for Airland2.

Aircraft	E_i	T_i	L_i	c_i^-	c_i^+
1	129	155	559	10	10
2	190	250	732	10	10
3	84	93	501	30	30
4	89	98	509	30	30
5	100	111	536	30	30
6	107	120	552	30	30
7	109	121	550	30	30
8	109	120	544	30	30
9	115	128	557	30	30
10	134	151	610	30	30
11	266	341	837	10	10
12	251	313	778	10	10
13	160	181	674	30	30
14	152	171	637	30	30
15	276	342	815	10	10

3.3.2 Relaxing sequencing constraints for a subset of aircraft

The relax procedure selects a sub-sequence of a given sequence Π. That subset contains a number of consecutive aircraft in the sequence Π. The motivation behind the relax procedure is as follows. An optimal landing schedule for the given landing sequence Π can be obtained in polynomial time using problem P1 because given Π problem P1 turns into a linear program. It is clear that such a schedule is optimal for Π, and it may not be the global optimal schedule for the problem, implying that if the landing sequence changes, an improved schedule may be produced. Given Π, the relax procedure iteratively destructs the landing order for only a small number of aircraft so that the solve procedure can (optimally) re-order the nominated aircraft in a short amount of time.

Almost every heuristic and meta-heuristic algorithm for the ALP operates by iteratively manipulating the landing order of a small number of aircraft to generate new landing sequences. The quality of a generated sequence is evaluated by its schedule. For example, Awasthi et al. (2013) and Girish (2016) presented polynomial-time algorithms for delivering the optimal schedule, and Furini et al. (2015) and Pinol

Table 3.3 : The separation times between pairs of aircraft for Airland2.

(i,j)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	-	3	15	15	15	15	15	15	15	15	3	3	15	15	3
2	3	-	15	15	15	15	15	15	15	15	3	3	15	15	3
3	15	15	-	8	8	8	8	8	8	8	15	15	8	8	15
4	15	15	8	-	8	8	8	8	8	8	15	15	8	8	15
5	15	15	8	8	-	8	8	8	8	8	15	15	8	8	15
6	15	15	8	8	8	-	8	8	8	8	15	15	8	8	15
7	15	15	8	8	8	8	-	8	8	8	15	15	8	8	15
8	15	15	8	8	8	8	8	-	8	8	15	15	8	8	15
9	15	15	8	8	8	8	8	8	-	8	15	15	8	8	15
10	15	15	8	8	8	8	8	8	8	-	15	15	8	8	15
11	3	3	15	15	15	15	15	15	15	15	-	3	15	15	3
12	3	3	15	15	15	15	15	15	15	15	3	-	15	15	3
13	15	15	8	8	8	8	8	8	8	8	15	15	-	8	15
14	15	15	8	8	8	8	8	8	8	8	15	15	8	-	15
15	3	3	15	15	15	15	15	15	15	15	3	3	15	15	-

Table 3.4 : The sequences generated by Algorithm 3.2 and their associated schedule delivered by CPLEX for Airland2.

$m=1$	i	3	4	5	6	8	7	9	10	1	14	13	2	12	11	15	
	r	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
	x_i	88	96	104	112	120	128	136	144	159	174	182	250	313	341	344	
	$c_i^- \alpha_i + c_i^+ \beta_i$	150	60	210	240	0	210	240	210	40	90	30	0	0	0	20	$z = 1500$
$m=2$	i	3	4	5	6	8	7	9	10	1	14	13	2	12	11	15	
	r	1	1	1	2	1	2	1	2	1	2	2	1	1	2	1	
	x_i	90	98	111	113	120	121	128	151	155	171	181	250	313	341	342	
	$c_i^- \alpha_i + c_i^+ \beta_i$	90	0	0	210	0	0	0	0	0	0	0	0	0	0	0	$z = 300$

and Beasley (2006), used the solvers CPLEX and Gurobi for generating the optimal schedule. One difference between our matheuristic and the previous studies is that we use the available solvers for both re-sequencing and scheduling.

We may classify two approaches for the re-sequencing: "complete" and "partial". In complete approach, changes are made to either the entire or a part of the sequence. However, the sequence is considered as a whole and the re-sequencing occurs on the complete array of aircraft. For example, Pinol and Beasley (2006) change a sequence through recombination operators and Salehipour et al. (2013), Awasthi et al. (2013) and Sabar and Kendall (2015) perform swap and insertion moves for the same purpose. One major shortcoming regarding that approach is that if an exact solver is used within these algorithms its role is relegated to only scheduling (i.e., the solver is only used to obtain the optimal landing schedule for a given sequence). Therefore, the solvers are not utilized to manipulate the sequences and obtain improved ones. Indeed, moves made by the solvers are often very fruitful, and contrary to the traditional manipulation techniques (e.g., swap or insertion), which are mostly performed randomly or myopically and without considering their subsequent impact on the rest of the sequence, the solvers can obtain high quality sequences and schedules. Yet, long sequences associated with large instances are a limiting factor for the available solvers.

In the partial approach which is suitable for long aircraft sequences, the sequence is broken down into a number of sub-sequences and changes are made to one sub-sequence at a time while the rest of the sequence remains unchanged. Hu and Chen (2005) and Zhan et al. (2009) decomposed the problem into smaller sub-sequences by using the receding horizon control method. To this end, the aircraft whose target times are within a specific receding horizon are selected first, and then are scheduled. We note that the partial approach also allows the exact solvers to be used for re-sequencing. For example, Xiangwei et al. (2011) proposed a sliding window algorithm for the ALP. At each iteration, by sliding the window the algorithm chooses a specified number of unscheduled aircraft, and solves the original ALP associated with those aircraft (i.e., obtains both landing sequence and schedule). In order to maintain the connection to the preceding and succeeding aircraft, the landing times of the previously scheduled aircraft are fixed. The algorithm iterates until all aircraft have their landing times fixed. Clearly, fixing landing time of the aircraft can potentially lead to sub-optimal schedules. A similar idea was investigated by Girish (2016). The study used a state-of-the-art algorithm to deliver an optimal landing schedule for a given sequence.

Similar decomposition ideas have also been used for solving the ASP. To model a variant of the ASP in which each aircraft can be shifted by at most certain positions in the sequence both backward and forward, Furini et al. (2012); Furini et al. (2015) presented an MIP, and proposed a rolling horizon algorithm, which iteratively solves the MIP for a variable time window. The time window rolls forward at each iteration. The last aircraft of the current iteration represents the initial condition for the next iteration. Salehipour and Ahmadian (2017) and Salehipour et al. (2018) adapted the idea of rolling horizon for the ALP, and proposed novel relaxation neighborhoods in which re-sequencing and scheduling are simultaneously performed by the exact solvers. Our matheuristic shares certain similarities with the rolling horizon framework and those relaxation neighborhoods.

The relax procedure in Relax_1 breaks down the given landing sequence Π into d sub-sequences. We use two parameters of "relaxation center", denoted by $RC \in \mathbb{Z}^+$, and "relaxation radius", shown by $RR \in \mathbb{Z}^+$ to form the sub-sequences. The parameter RC determines the center position of the sub-sequence, i.e., the aircraft positioned in the middle of the sub-sequence, and the parameter RR specifies the size of the sub-sequence. Hence the sub-sequence $R \subset \Pi$ is formed by the middle position RC and RR positions backward and forward. We call R the "relaxed" sub-sequence and the remaining sub-sequence(s) the "non-relaxed". We set RC and RR such that a certain degree of overlapping between the currently and previously relaxed aircraft is obtained. Ideally, RR should take a small value so that the relaxed sub-sequence contains a small number of aircraft ensuring that the re-sequencing is efficiently performed. The parameter RC is updated during the progress of Relax_1. Thus each iteration of the relax procedure involves the formation of a new sub-sequence R.

Every sub-sequence R is relaxed through lifting the precedence constraints of the aircraft in the sub-sequence, letting therefore the aircraft change their landing position in the sequence. The relax procedure does not let the landing position of the aircraft other than those in the current R be changed. To have a better grasp of the relax procedure, consider the landing sequence illustrated in Table 3.4 with 15 aircraft and one runway (the associated instance and its data are given in Table 3.3):

$$\Pi = \begin{pmatrix} 3 & 4 & 5 & 6 & 8 & 7 & \mathbf{9} & \mathbf{10} & \mathbf{1} & 14 & 13 & 2 & 12 & 11 & 15 \\ 1 & 1 & 1 & 1 & 1 & 1 & \mathbf{1} & \mathbf{1} & \mathbf{1} & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

Because the operation of the relax procedure is slightly more complex with multiple runways, we first discuss the single-runway case. Given $RC = 8$, i.e., the 8th position in the sequence and $RR = 1$, the relaxed sub-sequence is $R = \begin{pmatrix} 9, 10, 1 \\ 1, 1, 1 \end{pmatrix}$, which is highlighted in Π. Therefore, the relax procedure relaxes the precedence constraints for those aircraft. Figure 3.1 shows how this is performed in relation to the whole sequence. The relaxed aircraft are represented by green and yellow

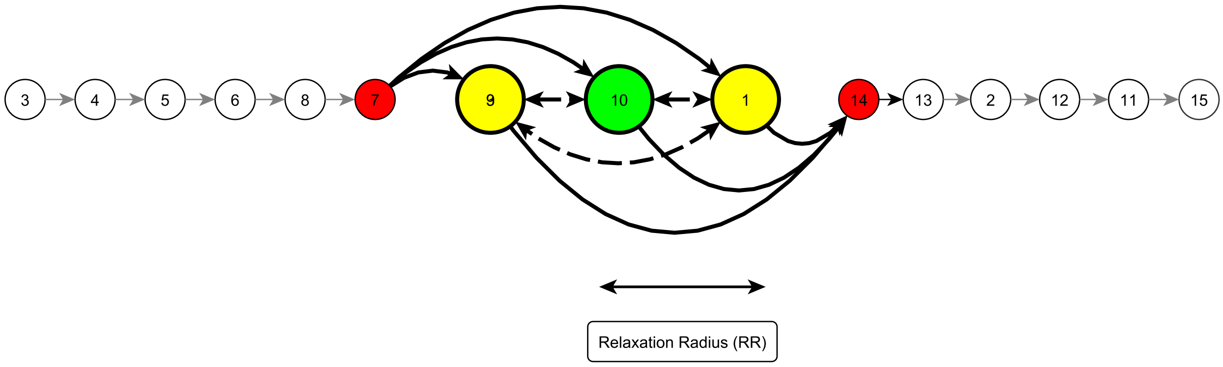

Figure 3.1 : Operation of the relax procedure in the instance with 15 aircraft and one runway. The relaxed aircraft are represented by green and yellow (the green vertex shows the relaxation center). Aircraft shown in red are immediate predecessor and successor of the relaxed sub-sequence. The conjunctive arcs specify the aircraft that are subject to only scheduling (their sequence is kept as is) and the disjunctive arcs (shown in dashed) represent the relaxed aircraft that are subject to both sequencing and scheduling. Arcs from vertex 7 and arcs to vertex 14 ensure that the relaxed aircraft will be re-sequenced only within the relaxed sub-sequence and that they are connected to the whole sequence.

vertices (the green vertex shows the relaxation center). As shown in the figure, arcs originating from the predecessor and ending in the successor of the relaxed sub-sequence, i.e., aircraft 7 and 14, vertices of which are shown in red, (i) enforce the relaxed aircraft to be re-sequenced only within the relaxed sub-sequence, that is between aircraft 7 and 14, and (ii) ensure that the relaxed sub-sequence is not disconnected from the complete sequence, by imposing the boundary constraints (arcs originating from vertex 7 and ending in vertex 14).

The operation of the relax procedure on multiple runways is more complicated because the relaxed aircraft are required to be allocated to runways before any sequencing and scheduling can be performed. Consider the landing sequence shown in Table 3.4 with two runways:

$$\Pi = \begin{pmatrix} 3 & 4 & 5 & 6 & 8 & 7 & \mathbf{9} & \mathbf{10} & \mathbf{1} & 14 & 13 & 2 & 12 & 11 & 15 \\ 1 & 1 & 1 & 2 & 1 & 2 & \mathbf{1} & \mathbf{2} & \mathbf{1} & 2 & 2 & 1 & 1 & 2 & 1 \end{pmatrix}.$$

Let the relax sub-sequence include aircraft 9, 10 and 1 (highlighted in the first row). As Figure 3.2 illustrates, aircraft 8 and 7 landing on runways 1 and 2 immediately precede the relaxed sub-sequence. Also, aircraft 14 and 2 that land on runways 2 and 1 immediately succeed the relaxed sub-sequence. Thus, the relax procedure operates such that the relaxed sub-sequence will be re-sequenced between those predecessors and successors. The relaxed aircraft can be assigned to any of the available runways.

Similar to the single-runway case, the order of aircraft in the non-relaxed sub-sequence(s) is kept unchanged and only the relaxed aircraft are subject to runway allocation and re-sequencing. Therefore, depending on the location of the relaxed sub-sequence (the beginning, middle or end of the sequence) a set of additional precedence constraints are required to ensure the "connectivity", i.e., the predecessor of the relaxed sub-sequence is also the predecessor to the successor of the relaxed sub-sequence. In our example, those constraints ensure that aircraft 8 lands before aircraft 2 on runway 1 (also 7 prior to 14 on runway 2). Additionally, those constraints establish the connectivity between the non-relaxed aircraft, i.e., aircraft 8 and 2 on runway 1 (aircraft 7 and 14 on runway 2) because if no relaxed aircraft is allocated to runway 1 (and 2), they still hold. In Figure 3.2, the highlighted black arcs between vertices 8 and 2 (and 7 and 14) illustrate these constraints (e.g., $x_2 \geq x_8 + s_{82}$ and $x_{14} \geq x_7 + s_{7,14}$). Moreover, the figure contains a set of conjunctive dashed arcs between the non-relaxed aircraft preceding the relaxed aircraft, i.e., originating from vertices 8 and 7, and between the relaxed aircraft and the succeeding non-relaxed aircraft, i.e., ending in vertices 14 and 2. Those arcs only exist if the relaxed aircraft land on the same runway of those non-relaxed aircraft.

The above discussion leads to the following set of connectivity constraints:

$$x_i \geq x_j + s_{ji} - M(2 - \gamma_{ir} - \gamma_{jr}), i = 9, 10, 1, j = 8, 7, r = 1, 2. \tag{3.11}$$

Similarly, the following constraints are defined between the relaxed aircraft and the succeeding aircraft, i.e., aircraft 2 and 14:

$$x_j \geq x_i + s_{ij} - M(2 - \gamma_{ir} - \gamma_{jr}), i = 9, 10, 1, j = 2, 14, r = 1, 2. \tag{3.12}$$

It should be pointed out that the relax procedure ensures that the relaxed aircraft will only be positioned within the relaxed sub-sequence and that they are

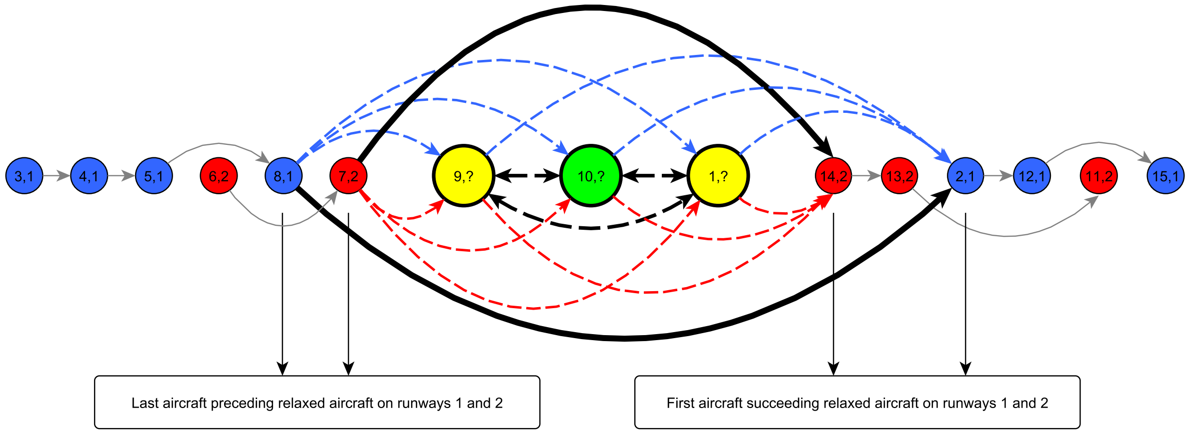

Figure 3.2 : Operation of the relax procedure in an instance with multiple runways, where the relax sub-sequence includes aircraft 9, 10 and 1 (a pair (i, r) represents aircraft i landing on runway r shown in green and yellow (the green represents the relaxation centre)). Aircraft shown in blue and red land on runway one and two respectively. The conjunctive arcs specify aircraft that keep their sequence and are subject to only scheduling. Disjunctive arcs within the relaxed sub-sequence (shown in dashed) represent the relaxed aircraft, which are subject to runway allocation, re-sequencing and scheduling.

always connected to the whole sequence. That is the primary advantage of the relax procedure over the traditional manipulations.

3.3.3 Solving partially relaxed sequence

The solve procedure aims at generating a (optimal) landing sequence for every sub-sequence R, and a (optimal) schedule for the whole sequence by utilizing the solver CPLEX (ILOG, 2017). For this reason, we construct a relaxed formulation of problem P1 by relaxing certain precedence constraints. Our relaxed formulation, denoted by problem P2, is computationally less challenging than solving problem P1 due to the smaller number of precedence constraints.

Problem P2

Constraints (3.4) in problem P1 determine the landing schedule for an aircraft by considering all scheduled aircraft, and ensuring that the separation times are met. Even a medium-sized instance of the ALP may involve a large number of constraints (3.4). We observe that in the majority of the tested instances such a large number

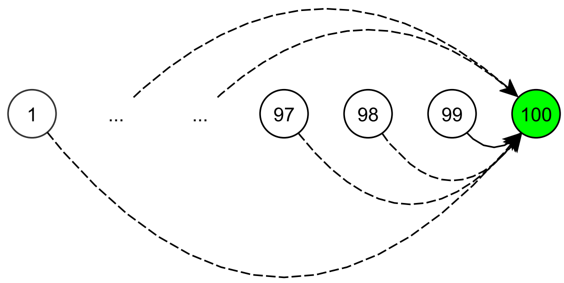

Figure 3.3 : Only a few immediate precedence constraints might be binding when scheduling an aircraft.

of constraints may be avoided without violating the separation time requirements. Therefore, we obtain problem P2 by relaxing a large number of constraints (3.4).

Let us start by a given instance with 100 aircraft and one runway. Given a landing sequence, problem P1 determines the landing schedule for every aircraft by considering the landing schedule of all of its preceding aircraft. This is equivalent to inequality (3.13):

$$x_j \geq \max_{1 \leq i \leq j-1} \{x_i + s_{ij}\}. \tag{3.13}$$

where aircraft j lands after aircraft i. Consider the landing sequence $\Pi = \begin{pmatrix} 1, 2, \ldots, 99, 100 \\ 1, 1, \ldots, 1, 1 \end{pmatrix}$. Assume that we want to determine the landing schedule for the last aircraft. It follows from constraints (3.4) in problem P1 that

$$x_{100} \geq x_1 + s_{1,100},$$
$$\vdots$$
$$x_{100} \geq x_{98} + s_{98,100},$$
$$x_{100} \geq x_{99} + s_{99,100}.$$

This is illustrated in Figure 3.3. However, because of the characteristic of the separation times, the binding constraints are typically due to certain immediate predecessors, and most likely due to the very immediate predecessor. Also, because problem P2 is built on a given sequence we only include the smallest number of constraints (3.4) for the non-relaxed sub-sequence(s), which is equal to $n-1$. It should be noted that the number of all constraints (3.4) in problem P1 is equal to $\sum_{i=1}^{n-1}(i) = \frac{n\times(n-1)}{2}$, and we therefore relax a huge number of those constraints in problem P2. We include all constraints (3.4) for the relaxed sub-sequence R though.

It is therefore clear that problem P2 is computationally more efficient than problem P1. This, however, has a drawback and that is the schedule obtained by problem P2 may not be feasible for the original problem P1. The reason is that we do not enforce the separation time requirement for all aircraft, except only for the adjacent aircraft in the non-relaxed sub-sequence(s). For example, for the presented instance with 100 aircraft, problem P1 introduces 4950 constraints (3.4), whereas problem P2 includes only 99 of those constraints (3.4). Due to the same reasoning, the objective function of problem P2 is a lower bound on the optimal objective function value of problem P1.

To implement problem P2 for the single-runway case, we use the following "or logical" (exclusive disjunction) constraints, which are features of CPLEX. Using the "or logical" constraints would lead to removal of the binary variables y_{ij} and constraints (3.5):

$$x_j \geq x_i + s_{ij} \quad \textbf{or} \quad x_i \geq x_j + s_{ji}, \quad j = i+1 \quad \textbf{or} \quad i = j+1. \tag{3.14}$$

We may follow the similar procedure and develop problem P2 for the multiple-runway case. This leads to the addition of constraints (3.15) and removal of constraints (3.4), (3.5) and (3.6) and binary variables y_{ij}:

$$\begin{aligned} x_j \geq x_i + s_{ij}(\gamma_{ir} + \gamma_{jr} - 1) \;\textbf{or}\; x_i \geq x_j + s_{ji}(\gamma_{ir} + \gamma_{jr} - 1),& \\ j = i+1 \quad \textbf{or} \quad i = j+1, r = 1, \ldots, m.& \end{aligned} \tag{3.15}$$

We note that constraints (3.15) do not include binary variables δ_{ij}. In summary, given a feasible sequence Π, the problem P2 is formed by the objective function (3.1), constraints (3.2), (3.3), (3.7), (3.15), and variables $\gamma_{ir}, x_i, \alpha_i, \beta_i$.

Next, we propose a procedure for dealing with the infeasible schedule generated by problem P2.

Feasibility of problem P2

Recall that problem P2 does not include all of constraints (3.4), meaning that the separation time requirement may not be applied between every pair of aircraft. Indeed, the separation time requirement is only applied between all aircraft in the relaxed sub-sequence and between the adjacent aircraft in the non-relaxed sub-sequence(s). This may result in the schedule generated by problem P2 to be infeasible for problem P1, particularly, if the separation times do not follow the triangular inequality.

The feasibility of the schedule produced by problem P2 can easily be verified, and then repaired if violated. Intuitively, if the landing schedule produced by problem P2 is given as the input to problem P1 (hence, all the decision variables in problem P1 turn into parameters), then solving problem P1 effectively results either in the same schedule as of problem P2, or in an infeasible status. The latter implies that some of the not-yet-added constraints (3.4) must be included in problem P2. Because we do not have a priori information on the number of those constraints and also to keep problem P2 efficiently solvable, we start by including constraints related to two immediate predecessors of an aircraft (initially we include one constraint (3.4) per aircraft, i.e., the landing schedule of an aircraft only depends on its immediate predecessor, which is shown in Figure 3.4a). To this end, we update problem P2 by adding an additional constraint per aircraft, which may be generated in a similar way to that of constraints (3.14) or (3.15). This is illustrated in Figure 3.4b. Again, we solve problem P2 and check the feasibility of the so obtained schedule against problem P1. In case of infeasibility, we consider three preceding aircraft (Figure 3.4c) for every aircraft, and follow the above procedure. No more than three preceding aircraft will be considered because there is no guarantee on the minimum number of preceding aircraft to be considered for each aircraft, in order to ensure that a feasible schedule is delivered. Therefore, if no feasible schedule is produced by problem P2, problem P1 that contains all preceding constraints is solved, by considering time limits, to deliver a feasible schedule.

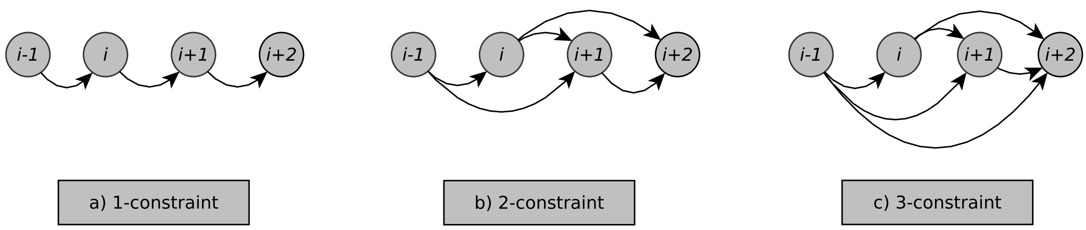

Figure 3.4 : Generating a feasible schedule for the ALP by problem P2: (a) the case of using one constraint (3.4) per aircraft in problem P2 (the default case), (b) the case of using two constraints (3.4) per aircraft, and (c) the case of using three constraints (3.4) per aircraft.

We use the solver CPLEX with a time limit for solving problem P2 because obtaining an optimal solution for problem P2 may still be a computational challenge. Setting reasonable time limits may not compromise the solution quality and can significantly improve the efficiency of solving the problem, which is important for real-world applications. We note that problem P2 reports an infeasible schedule only for one instance, out of the 49 tested instances in Section 3.3.6, which was then repaired by the above procedure. That indicates the effectiveness of problem P2 in generating feasible schedules for problem P1.

In order to further improve the efficiency of the proposed Relax_1, in what follows we propose speed-up techniques that further expedite the solve procedure.

Speed-up procedures

Fast algorithms are very important for real-world applications of the ALP. In this section, we propose two speed-up procedures to further reduce the computation time of the solve procedure. The first speed-up is proposed for the single-runway case and the second one is designed for the multiple-runway case.

Recall that in the single-runway case the Relax_1 starts from the beginning of a given sequence and progressively relaxes and solves a number of sub-sequences. Due to the penalties associated with early and late landings changing the landing position of an aircraft too further forward or backward in the sequence is unlikely to be "profitable", i.e., it may not improve the objective function value. Therefore, we restrict the re-sequencing of an aircraft within a few positions backward and forward, which we refer to the vicinity of the aircraft. The vicinity is controlled

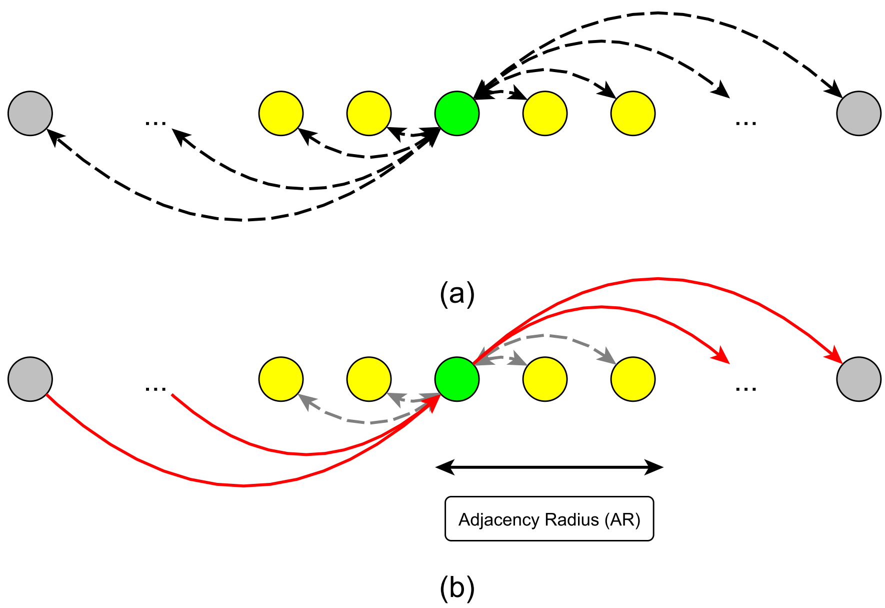

Figure 3.5 : The operation of speed-up procedure for the single-runway case: (a) a relaxed aircraft shown in green is connected either to neighbor aircraft (shown in yellow) that are located within a certain proximity, or to non-neighbor aircraft (shown in gray) by disjunctive arcs, and (b) due to parameter AR an aircraft is only re-sequenced with its neighbors that are within AR positions from the aircraft. The disjunctive dashed arcs highlight that and the conjunctive arcs are used to highlight the non-neighbor aircraft.

by the parameter $AR \in \mathbb{Z}^+$ ("adjacency radius"). This idea has been graphically illustrated in Figure 3.5.

For the multiple-runway case, we observe that by employing more runways the number of aircraft landings on each runway decreases, and therefore, the chance of scheduling aircraft landings on their target landing time increases. This results in a considerable number of aircraft to have an earliness or lateness penalty of zero. In most circumstances, relaxing such aircraft would not lead to a better schedule and it only increases the computational burden. Hence, before relaxing a sub-sequence the solve procedure pre-processes the landing penalties of the aircraft in the sub-sequence, as well as of the successor non-relaxed aircraft (if the successor non-relaxed aircraft are early or tardy delaying the relaxed aircraft may yield improved schedule) within the relaxation radius (RR); see Figure 3.6. If the landing penalties of all those aircraft are equal to zero, the solve procedure skips relaxing the sub-sequence and

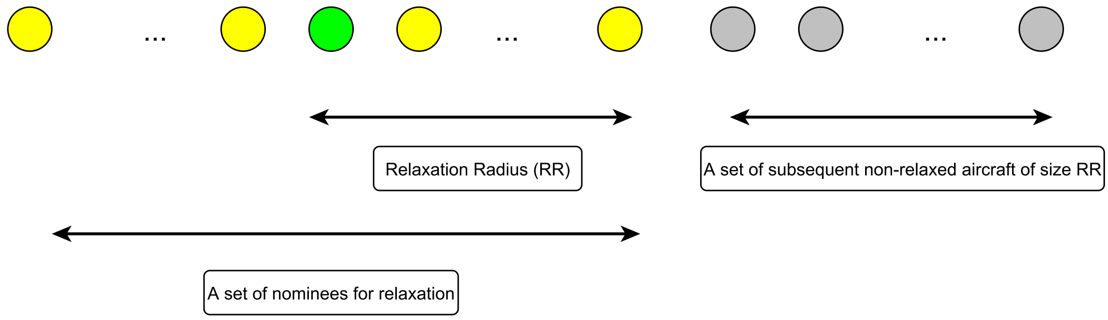

Figure 3.6 : A set of aircraft to be relaxed (shown in green and yellow), and a set of non-relaxed aircraft (shown in gray), both of which include the aircraft in the relaxation radius (RR). If the landing penalties of all those aircraft are equal to zero the solve procedure skips relaxing those aircraft and proceeds to the next sub-sequence.

considers the next sub-sequence.

We will discuss the impact of the speed-up procedures on the overall run time of the Relax_1 in Section 3.3.6.

3.3.4 Updating the incumbent sequence

Upon obtaining a schedule with an improved objective function value by the Relax_1, the incumbent sequence Π may need to be updated. This is easily performed by sorting the relaxed aircraft in non-decreasing order of their scheduled landing times (the value of decision variables x_i). It should be noted that because the order of non-relaxed aircraft remains unchanged, the update scheme keeps their order as is.

3.3.5 Operation of the R&S algorithm

Following the detailed discussion of the components of Relax_1 in previous sections, Figure 3.7 summarizes the implementation of the proposed algorithm for solving the ALP. In the flowchart, Π represents the initial sequence obtained by the ETLT construction heuristic (see Algorithm 3.2 in Section 3.3.1). The optimal schedule for Π is generated by solving problem P1 by CPLEX. We denote by $z_1(.)$ the objective function value of problem P1 and by $z_2(.)$ that of problem P2, both given a sequence of aircraft landings. As seen in Figure 3.7, Relax_1 starts iterating if and only if the initial sequence Π leads to an objective function value greater than zero. At each iteration, a set of consecutive aircraft is relaxed from their landing position in the incumbent sequence, and the aircraft are re-sequenced by solving problem P2 (or by problem P1 if problem P2 fails to produce a feasible landing schedule) and the current sequence is updated accordingly. That process continues until the stopping criterion is reached. It is important to note that if the objective function values of problems P1 and P2 for the obtained sequence do not match up and less than three immediate preceding aircraft were considered by problem P2 in the solve procedure (i.e., $f < 3$), Relax_1 is restarted by letting one more immediate preceding aircraft in problem P2, otherwise the obtained schedule is returned.

3.3.6 Computational results

We tested the proposed Relax_1 on 13 standard benchmark instances of the ALP available at OR Library *. The instances range from small (with 10 aircraft) to large ones (up to 500 aircraft), and include up to five runways. Therefore, we considered a total number of 49 instances, from which 13 instances use one runway and the remaining 36 instances utilize between two and five runways. We coded Relax_1 algorithm in the C++ programming language and implemented problems P1 and P2 by using the CPLEX Concert Technology version 12.8.0 (ILOG, 2017). Unless otherwise stated, we used default parameter settings for CPLEX. We performed the computational experiments on a Personal Computer with Intel® Core™ i5-6500 CPU clocked at 3.20GHz with 8GB of memory under Windows 10 operating system. The computing machine has four processors (threads). We used only one processor for CPLEX (both within Relax_1 and as the stand-alone).

Next, we explain the parameter tunning process followed by analyzing the effectiveness of the speed-up procedures in Section 3.3.6. The computational results of Relax_1 are reported in Section 3.3.6.

Parameter tunning

In order to tune the parameters of Relax_1, we do not run a full factorial design of experiments because the combination of values of the parameters would result

---

*http://people.brunel.ac.uk/~mastjjb/jeb/orlib/airlandinfo.html in a large number of experiments. Hence, we conduct some experiments in which a number of instances are solved using different values of parameters. We could not obtain a set of values for the parameters that work well on all instances. We, however, observe that the choice of values of the parameters depends on the size of the instances and further varies between the single-runway and multiple-runway cases. We therefore group the 13 instances with one runway into four classes of (1) small with up to 20 aircraft, medium that includes (2) between 20 and 150 aircraft and (3) between 150 and 250 aircraft, and (4) large with more than 250 aircraft. We also group the 36 instances with multiple runways into four classes of (1) small with up to 20 aircraft, medium in two different classes: (2) between 20 and 100 aircraft and (3) between 100 and 250 aircraft, and (4) large with more than 250 aircraft. Table 3.5 gives the values of the parameters, per each instance-group, that we use to solve all instances of the group.

We choose those values of the parameters because considering the impact of size of an instance they result in good quality solutions in short times. For example, the smaller values of d (number of sub-sequences) typically lead to larger sub-sequences but fewer of them, which are more challenging to solve in short times; and we observe that while that does not contribute to the solution's quality, it greatly increases the computation time. Regarding parameters RR and AR, due to earliness and tardiness penalties we observe that it is not beneficial to schedule an aircraft far from its target landing time, neither earlier nor later, implying that larger values for those parameters may not be beneficial. We impose time limit for the solver CPLEX as an effective computation time reduction strategy. We choose the time limits such that good solutions are obtained in short computation times.

The algorithm terminates if one of the following three stopping criteria is met:

- Maximum number of iterations: For both single- and multiple-runway cases we set that value to $\max(d, \min(\frac{n}{2}, 180))$.
- Maximum number of iterations without improvement: We set the value of this parameter to 15 and 5 for the single and multiple-runway cases. This criterion is only introduced when the number of executed iterations exceeds d (i.e., Relax_1 has been through the whole sequence at least once).

Table 3.5 : Value of parameters for Relax_1.

Runway	No. of sub-sequences (d)		Relaxation radius (RR)		Adjacency radius (AR)		Time limit (seconds)	
	n	Value	n	Value	n	Value	n	Value
Single	$n \leq 20$	4	$n \leq 150$	$\max(4, \frac{n}{25})$	$n \leq 250$	5	$n \leq 100$	1
	$20 < n \leq 150$	25	$150 < n \leq 250$	$\max(4, \frac{n}{50})$	$n > 250$	6	$n > 100$	2
	$150 < n \leq 250$	50	$n > 250$	$\max(4, \frac{n}{150})$				
	$n > 250$	180						
Multiple	$n \leq 20$	4	-	6	-	-	$n \leq 250$	1
	$20 < n \leq 100$	25	-	6	-	-	$n > 250$	2
	$100 < n \leq 250$	40	-	6	-	-		
	$n > 250$	80	-	6	-	-		

- Objective function value of zero: The value of zero for the objective function means that an optimal schedule is delivered (though, not every optimal schedule has the value of zero, see Table 3.4; we also note that the objective function cannot take a negative value).

Effectiveness of the speed-up procedures

As previously discussed in Section 3.3.3, we apply two speed-up procedures in order to improve the computation time of Relax_1. In this section we verify the effectiveness of those speed-up procedures by conducting two experiments on the four large instances with more than 100 aircraft. The first experiment is devoted to the speed-up for the single-runway case, and in the second experiment we analyze the speed-up for the multiple-runway case. We run Relax_1 for five times with and without the speed-ups. We report the average computation times of Relax_1 (over five runs) in Table 3.6 for single and multiple-runway cases. The table shows that the speed-up for the single-runway case can reduce the computation time of solving the large instances between 15% and 40%. Also, the speed-up for the multiple-runway case improves the run time of Relax_1 between 30% and 45%. The outcomes suggest the effectiveness of the speed-up procedures in reducing the run time of our proposed algorithm.

Comparison across state-of-the-art algorithms

In this section we compare the performance of Relax_1 and those of CPLEX (for solving problem P1) and RH-VAR2-LS of Girish (2016) that we re-implemented. We chose RH-VAR2-LS because it was shown to be the best performing algorithm in the literature for solving the ALP (Girish, 2016). We note that while we made a meticulous effort to re-implement the RH-VAR2-LS algorithm as close as possible to

Table 3.6 : The impact of the speed-up procedures on the computation time (in seconds) of Relax_1 for large instances ($n > 100$).

	Single runway		Multiple runways	
Instance	Speed up	No speed-up	Speed-up	No speed-up
Airland10	36.97	54.15	5.95	8.52
Airland11	5.82	6.87	6.42	9.68
Airland12	24.07	32.44	5.52	10.33
Airland13	63.83	105.68	21.95	32.33

its original implementation reported in that study, our re-implementation might be slightly different from the operation of the original algorithm. This is because the original study by Girish (2016) did not fully disclose all the components of the RH-VAR2-LS algorithm. In the following we list the components for which no details were given in Girish (2016):

- In section 3.4.4 of Girish (2016), the local search procedure containing four neighborhoods is explained. According to the paper "The local search procedure generates a set of neighborhood position vectors corresponding to each particle position vector $q \quad (q = 1, 2, \ldots, swarmsize)$. The best neighborhood (with the least total penalty cost) replaces the particle position vector if it is an improved solution". But no detail in this regard was given. More specifically, it is questionable how "a set of neighborhood position vectors" is generated, e.g., is the set formed individually for each neighborhood or a set containing neighbors from all neighborhoods is created? Is there any order among the neighborhoods? What is the size of this set? Does the size remain unchanged across all the instances or vary as the instance size grows? Does the size of set vary for single and multiple runway cases?

- Considering swap and remove-insert neighborhoods (discussed in 3.4.4 of Girish (2016)), are the positions selected uniformly or is the selection somehow guided?

- According to the explanation given in section 4.2.2 of Girish (2016), improved solutions obtained from local search must undergo a repair mechanism which is, according to the author similar to the one presented in Tasgetiren et al. (2004). However, the problem addressed in Tasgetiren et al. (2004) is different from

ALP (the paper studies single machine with objective of total weighted tardiness with no time windows). It is therefore not clear how repair mechanism works for the multiple-runway case.

- We also noted a case in which the original algorithm of Girish (2016) may get stuck. In section 3.4.4 of Girish (2016), the proposed rolling horizon is explained in which a set of aircraft in the so-called "optimization window" are optimized by HPSO-LS while the landing order of aircraft preceding the window is frozen. To initialize HPSO-LS, feasible landing orders and runway allocations for aircraft in the window are generated. The initial swarm is then scheduled with regard to preceding aircraft. But what if no feasible order or runway allocation exists for aircraft in the window with respect to preceding aircraft. We specifically faced this case for Airland 12 and 13 with 2 runways for which algorithm failed to generate initial swarm for some aircraft in the window and got stuck.

We warm-start the CPLEX with the same initial landing sequence that we use for Relax_1 and we let the CPLEX run for a maximum of one hour. We choose the long run time of one hour for the CPLEX to show that the CPLEX will not benefit from long computation times. Also, we terminate the RH-VAR2-LS algorithm by following the stopping rules discussed in Girish (2016). That results in the computation times of RH-VAR2-LS to be significantly longer than Relax_1, implying that RH-VAR2-LS benefits from extra run times. Despite these efforts, our re-implementation of the RH-VAR2-LS algorithm led to inferior solutions to those reported in the original study. For clarity, in Table 3.7 we report the outcomes of the re-implemented RH-VAR2-LS (column "RH-VAR2-LS (Girish, 2016)"), as well as the objective function values reported in Girish (2016) (column "BKS").

In Table 3.7, the first three columns give the details for each instance including the name and the number of aircraft (n) and runways (m). The fourth column reports the best known solution (BKS) for each instance, which is taken from Girish (2016). Columns five to seven show the objective values, optimality gaps upon termination and the computation times for CPLEX. The remaining columns show the outcomes of RH-VAR2-LS of Girish (2016), which we re-implement on our machine and those of Relax_1, respectively, where the columns z^*, z_{avg} and z_{std} show the best, average and standard deviation of the objective function values obtained over five runs by each algorithm, and columns T_{avg} and and T_{std} denote the average and standard deviation of the computation times of each algorithm over five runs.

According to Table 3.7, the inferior outcomes of the re-implemented RH-VAR2-LS indicate that under the same computational settings our Relax_1 is able to obtain superior schedules. Moreover, while the Relax_1 algorithm outperforms both CPLEX and the re-implemented RH-VAR2-LS, it is the fastest method, suggesting its suitability for practical settings. In particular, the Relax_1 delivers the same or superior solutions (and never worse) to both CPLEX and RH-VAR2-LS, has the smallest run times that is bounded by around 1 minute, and has the standard deviation of 0 for all but one instance.

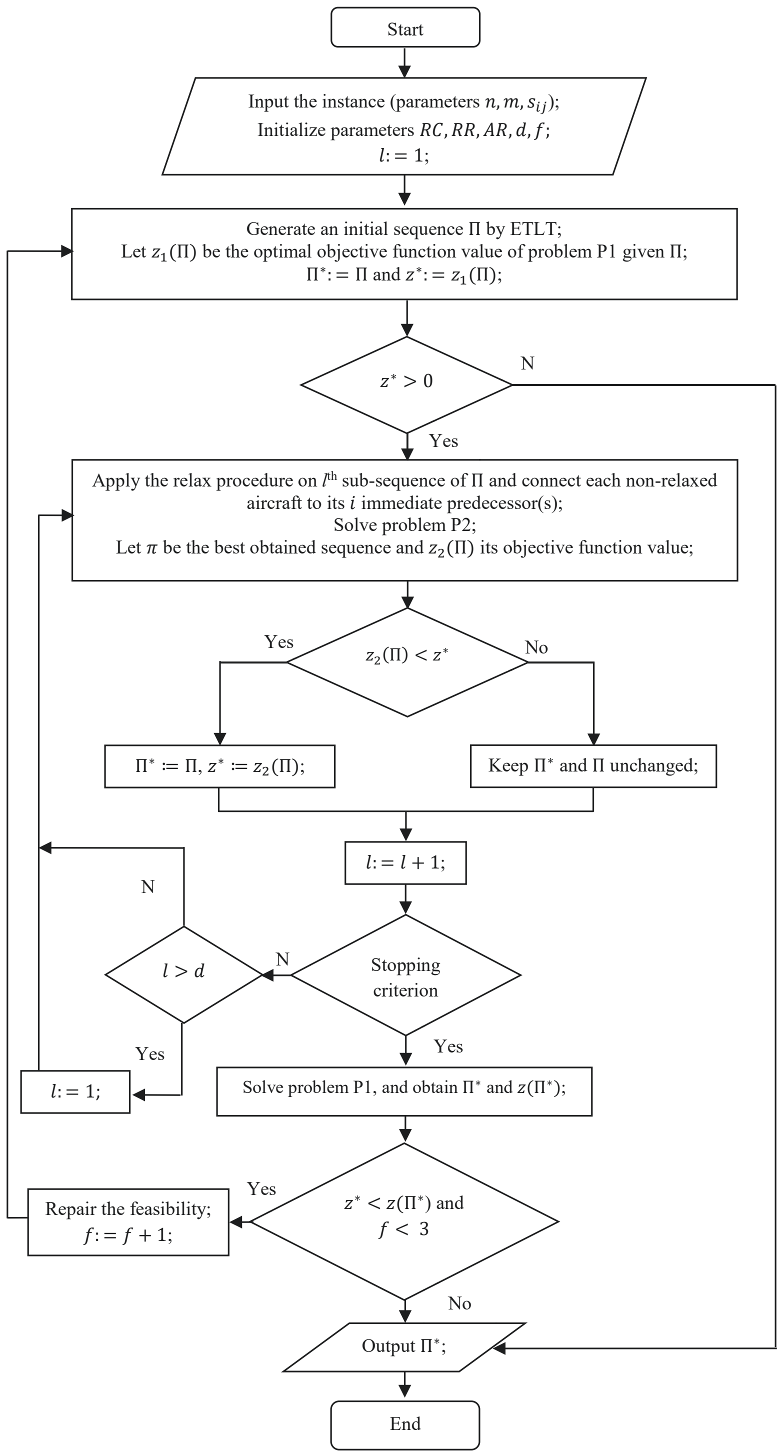

Figure 3.7 : Detailed operations of Relax_1 algorithm for the ALP.

Table 3.7 : The detailed outcomes of CPLEX, RH-VAR2-LS (Girish (2016)) and Relax_1 (for experiments CPLEX 12.8.0 was used).

Instance	n	m	BKS	CPLEX			RH-VAR2-LS (Girish, 2016)					Relax_1				
				$z*$	Gap (%)	T	$z*$	z_{avg}	z_{std}	T_{avg}	T_{std}	$z*$	z_{avg}	z_{std}	T_{avg}	T_{std}
Airland1	10	1	700	700	0	0.03	700	700	0	0.30	0.09	700	700	0	0.14	0.03
		2	90	90	0	0.03	90	90	0	0.27	0.26	90	90	0	0.29	0.05
		3	0	0	0	0.02	0	0	0	0.04	0.02	0	0	0	0.00	0.00
Airland2	15	1	1480	1480	0	0.08	1480	1492	10.95	0.42	0.16	1480	1480	0	0.21	0.02
		2	210	210	0	0.06	210	210	0	0.45	0.29	210	210	0	0.47	0.01
		3	0	0	0	0.02	0	0	0	0.05	0.03	0	0	0	1.08	0.44
Airland3	20	1	820	820	0	0.03	820	1108	280.57	0.37	0.20	820	820	0	0.14	0.00
		2	60	60	0	0.08	60	60	0	1.31	0.73	60	60	0	1.25	0.01
		3	0	0	0	0.03	0	0	0	0.09	0.03	0	0	0	0.01	0.00
Airland4	20	1	2520	2520	0	1.87	2520	2520	0	0.43	0.22	2520	2520	0	0.33	0.01
		2	640	640	0	8.94	640	640	0	1.31	0.67	640	640	0	2.77	0.07
		3	130	130	0	0.69	130	130	0	1.19	0.63	130	130	0	2.20	0.05
		4	0	0	0	0.05	0	0	0	0.08	0.04	0	0	0	0.35	0.01
Airland5	20	1	3100	3100	0	9.55	3680	4058	306.63	0.97	0.34	3100	3100	0	0.91	0.01
		2	650	650	0	8.94	650	680	22.36	1.30	0.69	650	650	0	3.63	0.02
		3	170	170	0	0.97	170	170	0	0.97	0.37	170	170	0	2.99	0.01
		4	0	0	0	0.05	0	0	0	0.08	0.04	0	0	0	0.33	0.00
Airland6	30	1	24442	24442	0	0	24442	24442	0	2.33	1.39	24442	24442	0	0.04	0.00
		2	554	554	0	0.14	568	614.2	36.72	4.27	2.24	554	554	0	0.62	0.01
		3	0	0	0	0.03	0	0	0	0.11	0.06	0	0	0	0.03	0.00
Airland7	44	1	1550	1550	0	0.16	1550	1550	0	4.76	2.46	1550	1550	0	0.17	0.01
		2	0	0	0	0.05	0	0	0	0.45	0.25	0	0	0	0.01	0.00
Airland8	50	1	1950	1950	0	0.19	2020	2206	181.19	2.01	1.17	1950	1950	0	1.48	0.02
		2	135	135	0	0.66	135	138	6.71	6.67	3.56	135	135	0	6.88	0.03
		3	0	0	0	0.2	0	0	0	0.33	0.17	0	0	0	0.02	0.00
Airland9	100	1	5611.7	5611.99	25.23%	3600	5871.12	6200.57	478.71	10.58	5.63	5611.7	5611.7	0	3.79	0.06
		2	444.1	444.1	6.33%	3600	444.1	456.116	23.23	34.59	18.34	444.1	444.1	0	4.16	0.07
		3	75.75	75.75	0	0.83	75.75	221.818	168.25	17.41	12.25	75.75	75.75	0	2.39	0.01
		4	0	0	0	0.37	0	100.642	203.84	8.60	3.70	0	0	0	0.11	0.00
Airland10	150	1	12292.2	12310.4	52.04%	3600	12616	13311.1	620.43	29.18	15.74	12292.2	12293	1.79	36.97	0.57
		2	1143.7	1143.7	79.26%	3600	1143.7	1143.81	0.10	63.15	33.30	1143.7	1143.7	0	8.43	0.03
		3	205.21	205.21	0	7.86	205.21	227.186	49.14	50.13	26.29	205.21	205.21	0	10.67	0.03
		4	34.22	34.22	0	2.22	34.22	166.47	183.40	22.84	14.34	34.22	34.22	0	3.88	0.01
		5	0	0	0	0.87	7.08	198.862	223.25	23.27	13.81	0	0	0	0.82	0.01
Airland11	200	1	12418.32	12418.32	37.74%	3600	12682.2	13190.4	303.19	30.80	16.40	12418.32	12418.32	0	5.82	0.02
		2	1330.91	1330.91	87.15%	3600	1330.91	1355.67	27.03	91.29	47.91	1330.91	1330.91	0	10.04	0.05
		3	253.07	253.07	0	11.89	253.07	264.198	24.52	69.46	35.42	253.07	253.07	0	9.15	0.03
		4	54.53	54.53	0	3.93	54.53	182.27	181.46	57.56	32.66	54.53	54.53	0	5.03	0.03
		5	0	0	0	1.61	44.41	114.074	114.86	35.31	14.97	0	0	0	1.48	0.01
Airland12	250	1	16122.18	16157	42.84%	3600	16466.8	17079.5	397.87	55.73	30.35	16122.18	16122.18	0	24.07	0.01
		2	1695.62	1695.62	90.87%	3600	1707.04	2202.068	1069.76	52.93	20.37	1695.62	1695.62	0	11.73	0.03
		3	221.97	221.97	0	22.23	221.97	289.164	148.54	68.04	36.34	221.97	221.97	0	5.71	0.01
		4	2.44	2.44	0	3.92	35.69	817.014	830.86	58.88	31.18	2.44	2.44	0	3.70	0.02
		5	0	0	0	2.37	57.04	142.02	71.01	48.93	29.17	0	0	0	0.93	0.03
Airland13	500	1	37064.11	37523.5	50.32%	3600	39361.2	40177.6	925.62	302.69	152.76	37077.4	37077.4	0	63.83	0.24
		2	3920.39	3957.02	97.65%	3600	3931.33	4001.452	97.12	138.03	4.45	3920.39	3920.39	0	41.04	0.03
		3	673.85	673.85	90.02%	3600	691.85	864.074	216.17	239.73	136.35	673.85	673.85	0	30.18	0.04
		4	89.95	89.95	0	16.86	89.95	179.634	120.05	201.63	98.77	89.95	89.95	0	12.72	0.02
		5	0	0	0	11.29	193.6	340.074	126.77	143.72	85.41	0	0	0	3.86	0.03

From Table 3.7, it is clear that the exact solvers such as CPLEX may not be a good choice for practical settings owing to their long run times and poor quality of the generated schedules.

In conclusion, the practitioner may choose the proposed Relax_1 due to the following reasons. Firstly, as Table 3.7 details, Relax_1 is the top performing method with respect to both the quality of the generated schedules and the run times. Secondly, the implementation of Relax_1 is simple and straightforward as Relax_1 only relies on a generic decomposition-based framework. Thirdly, Relax_1 utilizes the standard solver CPLEX, which has a demonstrated and established performance. The latter is important because it provides the practitioner with the extensive access to the support of CPLEX, and removes the need for developing customized heuristic and meta-heuristic methods which require extensive research and without guaranteeing the outcomes due to erratic performance. In our opinion, the conceptual simplicity of Relax_1 is a significant advantage because it makes Relax_1 algorithm to be easily adapted for other challenging combinatorial optimization problems.

3.4 Relax_2 for ASP

In this section we present another matheuristic (called Relax_2) to tackle ASP. We test the algorithm on two sets of 27 benchmark instances for the Milan international airport Furini et al. (2015). The instances range from 60 aircraft up to 170 aircraft and include one runway. It should be noted that the proposed Relax_2 slightly differs from Relax_1. The differences are as follows:

- To generate initial solutions we use a powerful single-machine scheduling problem solver called SiPSi originally developed by Tanaka and Fujikuma (2012) for the single machine scheduling problem with job release time and due-date and the objective function of minimizing the total earliness-tardiness, i.e., for problem $1|p_i, r_i, d_i| \sum(w_i^E E_i + w_i^T T_i)$, where r_i and d_i are the release time and due-date of job $i \in I$, and $E_i = \max\{0, d_i - C_i\}$ and $T_i = \max\{0, C_i - d_i\}$ show earliness and tardiness for job i, respectively (C_i is the completion time of job i, and w_i^E and w_i^T are the earliness and tardiness penalty coefficient per unit of earliness and tardiness). According to Kramer and Subramanian (2017), SiPSi is the best exact method for solving $1|p_i, r_i, d_i| \sum w_i^E E_i + w_i^T T_i)$. We customize

SiPSi so that we can use it to generate an initial sequence of landing/take-off for the ASP. For this reason, we construct an instance for SiPSi per instance of the ASP as following. We assume the identical processing time for all jobs, which we obtain as the minimum separation time among all ASP instances. That results in setting the processing time of jobs to 2. We let the values of release time and due-date be equal to the target landing time and we set $w_i^E = 0$ (the earliness penalty coefficient). We note that we set w_i^T as in the ASP instances, i.e., $w_i^T = w_i$. In short, we solve problem $1|p_i = 2, r_i = d_i| \sum w_i T_i$ by SiPSi, which results in a sequence of aircraft landing/take-off for the ASP;

- For instances with less than 60 aircraft we set relaxation radius to 5 (i.e., $RR = 5$) and for larger instances to 7 (i.e., $RR = 7$).

- We initialize the number of sub-sequences (d) by 15 for instances containing less than 150 aircraft and 30 otherwise. The value of d is updated after every d iterations as follows: if $d \leq 10$, then $d = 10$, otherwise $d = d - 5$. Using this dynamic updating scheme for d, we reduce the number of overlapping aircraft between sub-sequences as the search proceeds.

- We use three stopping criteria for the Relax_2, whichever occurs the first: (i) the maximum number of iterations of 60, (ii) the maximum number of iterations without improvement of 15, and (iii) obtaining a schedule with objective value of zero.

We run the Relax_2 algorithm for 5 times for each instance. Table 3.8 details the results for Relax_2. In the table, the first column gives the instance names, the second column reports the optimal objective function value for the instances as reported in Avella et al. (2017). Column three shows the objective value of the initial solutions obtained by SiPSi. Columns four and five detail the best solution delivered by Relax_2 across five runs and also the gap from the optimal solution, where the gap is obtained as $\dfrac{z^* - Opt}{z^*} \times 100$. Similarly, columns six and seven denote the average objective function value over five runs and the associated gap calculated as $\dfrac{z_{avg} - Opt}{z_{avg}} \times 100$. Finally column eight presents the average computation time in seconds over five runs.

Table 3.8 : The results of Relax_2 for 27 instances of ASP (for experiments CPLEX 12.8.0 was used).

Instance	Opt.	SiPSi	z^*	Gap z^*	z_{avg}	Gap z_{avg}	T_{avg}
FPT01	265	265	265	0	265	0	0.73
FPT02	293	301	293	0	293	0	1.45
FPT03	255	263	255	0	255	0	0.75
FPT04	268	276	268	0	268	0	0.76
FPT05	249	257	249	0	249	0	0.7
FPT06	167	167	167	0	167	0	0.5
FPT07	198	205	198	0	198	0	0.98
FPT08	167	179	167	0	167	0	0.86
FPT09	183	195	183	0	183	0	0.74
FPT10	211	223	211	0	211	0	0.76
FPT11	229	241	229	0	229	0	0.64
FPT12	207	207	207	0	207	0	0.4
FPT13	604	614	604	0	604	0	10.6
FPT14	1994	2012	1994	0	1995.6	0.08	25.88
FPT15	796	796	796	0	796	0	8.54
FPT16	1316	1349	1316	0	1316	0	23.51
FPT17	2368	2439	2370	0.08	2370.4	0.1	23.73
FPT18	1508	1775	1512	0.27	1512	0.27	23.7
FPT19	2115	2127	2115	0	2115	0	21.75
FPT20	3055	3184	3057	0.07	3057	0.07	30.13
FPT21	3577	4018	3579	0.06	3579	0.06	25.42
FPT22	2909	2958	2909	0	2909	0	25.78
FPT23	3649	3658	3649	0	3649	0	26.62
FPT24	3691	4132	3693	0.05	3693	0.05	21.99
FPT25	3786	3797	3786	0	3786	0	29.95
FPT26	4142	4203	4142	0	4142	0	38.46
FPT27	4171	4615	4177	0.14	4177	0.14	35.93

As Table 3.8 shows, Relax_2 delivers the optimal solution for 21 instances, out of 27. For the six instances that the Relax_2 does not report the optimal solution, its largest gap is 0.27% and its average gap is around 0.11%. Notably, the computation time of Relax_2 is never any greater than 40 seconds on average. Those highlight the Relax_2 as an ideal choice of the algorithm for larger real-world problems.

3.5 Conclusion

We proposed two efficient R&S matheuristic algorithms for solving the ALP and ASP. The core idea behind our algorithms is iterative deconstruction and re-construction of a sub-sequence of aircraft landings. By solving benchmark instances and comparing the outcomes and the state-of-the-art algorithm and the solver CPLEX we showed that the proposed algorithms obtain the best known solutions and are quick enough to be implemented in real-time. Those advantages of the R&S algorithms along with its algorithmic simplicity and ease of implementation contribute to the applicability of the proposed algorithms for real-world applications. This is very important because due to the typical short time window available for planning the aircraft landings delivering high quality landing schedules or updating the available schedules in a short time is very important, and therefore, fast and effective algorithms are paramount.

Although we utilized the solver CPLEX in the solve procedure of proposed algorithm, mainly due to the high performance of CPLEX, that does not limit the usability and applicability of the R&S algorithms because other exact methods including the branch-and-bound can be used instead. As future research directions, one may extend the R&S algorithm for the case of separation time requirements between pairs of aircraft landing on different runways. Another direction can be generalizing the R&S for the variants of nonidentical and dedicated runways. Also, investigations may be performed around adaptability of the R&S to other challenging combinatorial optimization problems.

Chapter 4

Just-In-Time Job Shop Scheduling

This chapter is based on following publications:

- M. M. Ahmadian and A. Salehipour, "The just-in-time job-shop scheduling problem with distinct due-dates for operations," *Journal of Heuristics*, pp. 1-30, 2020.
- M. M. Ahmadian, A. Salehipour and TCE. Cheng, "A meta-heuristic to solve the just-in-time job-shop scheduling problem," *European Journal of Operational Research*, 2020.

4.1 Introduction

The job-shop problem is a famous scheduling problem, in which every job has of a set of operations that needs to be performed in a specific order by a set of machines such that a performance criterion is optimized. Due to its complexity (Garey et al., 1976) and countless applications (French, 1982) the problem has attracted much attention. Nonetheless, only few studies address the performance criterion of minimizing the (weighted) earliness and tardiness. Such a criterion is pertinent to the just-in-time (JIT) policy because minimizing the earliness impacts, e.g., the warehousing and inventory costs, and minimizing the tardiness leads to shorter delivery times, and therefore, to a higher level of customer satisfaction.

The JIT job-shop scheduling problem is a variant of the classical job-shop scheduling, in which every job (operation) consists of a set of operations with a respective due-date and earliness and tardiness penalty coefficients, and any deviation from the due-date is penalized. More specifically, completing a job (an operation) before its due-date leads to earliness penalties and completing it after the due-date results in tardiness penalties. The objective function minimizes the total penalties of earliness and tardiness.

Generally speaking, two types of due-dates have been studied in the literature, namely the job-level and the operation-level. While the former involves a due-date for every job (and therefore all operations of the job share the same due-date), the latter assumes a due-date for each operation, implying that every operation has a distinct due-date. Models with the operation-level due-date characterize a broader range of environments; however, they are often more challenging to solve. In this chapter, we focus on the operation-level due-date, so we explore the more general case.

We propose two matheuristic algorithms (called Math_1 and Math_2) designed to take advantage of a novel decomposition method to tackle JIT-JSS. The method operates by decomposing JIT-JSS into smaller problems, delivering optimal or near-optimal sequences for the operations, and generating a schedule, i.e., determining the completion time for each operation. It is known that for sequencing and scheduling problems, including JIT-JSS, if a feasible sequence is provided, then an optimal schedule for the given sequence can be obtained in polynomial time (Pinedo, 2008: Chapter 4, page 74). Similar decomposition ideas have been used to address the routing scheduling problem. For example, Dumas et al. (1990) decomposed the vehicle routing problem with time windows into sequencing and scheduling sub-problems, and obtained the optimal service time schedule for a fixed path in $O(n)$ time, where n is the number of nodes or customers. Here, we consider JIT-JSS in which there are distinct due-dates, and earliness and tardiness penalty coefficients for the operations. In other words, we study the more general and difficult variant of the job-shop scheduling problem involving operation due-dates. Furthermore, we test our solution methods on the benchmark instances in Baptiste et al. (2008), where all the parameters including the due-dates, and earliness and tardiness penalty coefficients are given for each instance.

We organize the rest of the chapter as follows: Section 4.2 discusses some of the applications of JIT-JSS. In Section 4.3 we introduce JIT-JSS, define the notation, and formulate the problem as a mathematical program. Sections 4.4 and 4.5 proposes two matheuristic algorithm (i.e. Math_1 and Math_2) for solving the JIT-JSS problem, and explain the components of each algorithm. Sections 4.4.6 and 4.5.2 report the computational experiments and Section 4.6 concludes the chapter by suggesting topics for future research. Math_1 and Math_2 have previously appeared in the following publications:

- Math_1 (Section 4.4):
  - M. M. Ahmadian and A. Salehipour, "The just-in-time job-shop scheduling problem with distinct due-dates for operations," *Journal of Heuristics*, pp. 1-30, 2020.
- Math_2 (Section 4.5):
  - M. M. Ahmadian, A. Salehipour and TCE. Cheng, "A meta-heuristic to solve the just-in-time job-shop scheduling problem," *European Journal of Operational Research*, 2020.

4.2 Applications

JIT-JSS, i.e., the job-shop scheduling problem to minimize (weighted) earliness and tardiness has broad real-world applications. Below we discuss some of them.

Consider a railway transportation system in which a set of trains operates. A train visits a set of stations in a pre-specified order. Obviously, no two trains can be present simultaneously at any station. A train's stopping time at a station plus its travel time from the preceding station models the processing time of the train at the station. The traffic controller must schedule the trains visiting the stations such that the trains leave the stations at the ideal (desirable) departure times. Each time unit of the actual departure time before or after the ideal departure time is penalized. The traffic controller aims at constructing a train timetable such that the total deviation of the trains from their ideal departure times is minimized. We refer the interested reader to Flamini and Pacciarelli (2008) and Liu and Kozan (2011) for the details on the train scheduling problem.

Pham and Klinkert (2008) discussed an application of the job-shop scheduling in the context of hospital resource allocation where a set of patients in a hospital is to undergo surgeries. Typically, patient flow sequentially follows the three stages of pre-operative, peri-operative, and post-operative. Although the characteristics of the tasks performed on each patient (e.g., duration and processing route, are different), the tasks must be carried out in a pre-determined order. Each task is assigned an ideal completion time (due-date) and any deviation from the due-date should be avoided.

Suppose a pre-fab company has received orders for pre-built houses. Each order has a unique design, which in turn requires different manufacturing and installation of the modules, that is an order of completing the modules must be met per design. Since such houses are usually delivered very fast (often less than six months), rigorous scheduling of tasks is required. Also, the company may not be willing to deliver the houses to the owners before the delivery dates due to owner's reluctance to move in causing capital tied up in inventory. It is clear that any delay in delivery is penalized per contract.

4.3 Problem statement

Given $N = \{1, \ldots, n\}$ and $M = \{1, \ldots, m\}$ the sets of jobs and machines, in the JIT-JSS problem job i visits each machine exactly once and has m operations, where $\mathcal{O}_i = \{O_i^1, \ldots, O_i^m\}$ is the set of its operations and O_i^1 is the first scheduled operation of job i, O_i^2 is the second scheduled operation and so on. One may denote the set of all operations by $\mathcal{O} = \{O_i^k | O_i^k \in \mathcal{O}_i, \forall i \in N, k \in \{1, \ldots, m\}\}$. The order in which operations of a job visit machines (the "processing route") is known a priori. In this context, the machine that processes operation O_i^k is denoted by $\mathcal{M}(O_i^k) \in M$. Likewise, for each machine $M_j \in M$ we let $\mathcal{O}(M_j)$ denote the set of operations to be processed by machine M_j. All operations are available at time 0. Operation O_i^k has a due-date d_i^k, and earliness and tardiness penalty coefficients α_i^k and β_i^k per unit of deviation from d_i^k. Any deviation from the due-date results in either an earliness or a tardiness penalty. More precisely, completing operation O_i^k earlier than its due-date, i.e., $C_i^k \leq d_i^k$, incurs a penalty of $\alpha_i^k E_i^k$, where C_i^k is the completion time of O_i^k and $E_i^k = \max\{d_i^k - C_i^k, 0\}$ is the amount of earliness. Similarly, completing operation O_i^k after its due-date, i.e., $C_i^k \geq d_i^k$, incurs a penalty of $\beta_i^k T_i^k$, where $T_i^k = \max\{C_i^k - d_i^k, 0\}$ is the amount of tardiness.

Similar to the classical job-shop, each machine can process at most one operation at a time (the resource constraint). Also, preemption of the operations is not allowed. The JIT-JSS aims to obtain a feasible schedule, i.e., the completion time of the operations so as to minimize the total weighted earliness and tardiness penalties. The mathematical notations used in the problem formulation have been summarized in Table 4.1.

Table 4.1 : The mathematical notations used in the JIT-JSS formulation.

Sets	
N	Set of jobs, $N = \{1, \ldots, n\}$.
M	Set of machines, $M = \{1, \ldots, m\}$.
$\mathcal{O}$ ($\mathcal{O}_i$)	Set of all operations (set of operations of job $i \in N$). Also, we let O_i^k be the kth scheduled operation of job $i \in N, k \in \{1, \ldots, m\}$.
Parameters	
p_i^k	Processing time of operation $O_i^k, p_i^k \geq 0, i \in N, k \in \{1, \ldots, m\}$.
d_i^k	Due-date of operation $O_i^k, d_i^k \geq 0, i \in N, k \in \{1, \ldots, m\}$.
α_i^k	Earliness penalty coefficient associated with operation $O_i^k, \alpha_i^k \geq 0, i \in N, k \in \{1, \ldots, m\}$.
β_i^k	Tardiness penalty coefficient associated with operation $O_i^k, \beta_i^k \geq 0, i \in N, k \in \{1, \ldots, m\}$.
Variables	
C_i^k	Completion time of operation $O_i^k, C_i^k \geq 0, i \in N, k \in \{1, \ldots, m\}$.
ES_i^k	Earliest start time of operation $O_i^k, ES_i^k \geq 0, i \in N, k \in \{1, \ldots, m\}$.
EC_i^k	Earliest completion time of operation $O_i^k, EC_i^k \geq 0, i \in N, k \in \{1, \ldots, m\}$.
E_i^k	Earliness of operation $O_i^k, E_i^k \geq 0, i \in N, k \in \{1, \ldots, m\}$.
T_i^k	Tardiness of operation $O_i^k, T_i^k \geq 0, i \in N, k \in \{1, \ldots, m\}$.
C_{max}	Makespan, i.e., $C_{max} = \max_{i \in N}\{C_i^m\}$.
L_{max}	Maximum lateness.

A mathematical model for JIT-JSS

Baptiste et al. (2008) proposed a mathematical model for the JIT-JSS problem by using the so called "logical or" constraints. Problem$_{\mathcal{JIT}-\mathcal{JSS}}$ shows this.

Problem$_{\mathcal{JIT}-\mathcal{JSS}}$

$$z = \min \sum_{i=1}^{n} \sum_{k=1}^{m} (\alpha_i^k E_i^k + \beta_i^k T_i^k) \tag{4.1}$$

subject to

$$C_i^k - p_i^k \geq 0, \quad i \in N, k \in M, \tag{4.2}$$

$$C_i^k \leq C_i^{k+1} - p_i^{k+1}, \quad i \in N, k \in \{1, \ldots, m-1\}, \tag{4.3}$$

$$C_i^k \geq C_l^h + p_i^k \quad \textbf{or} \quad C_l^h \geq C_i^k + p_l^h, \quad j \in M, O_i^k, O_l^h \in \mathcal{O}(M_j), \tag{4.4}$$

$$E_i^k \geq d_i^k - C_i^k, \quad i \in N, k \in M, \tag{4.5}$$

$$T_i^k \geq C_i^k - d_i^k, \quad i \in N, k \in M, \tag{4.6}$$

$$E_i^k \geq 0, \quad i \in N, k \in M, \tag{4.7}$$

$$T_i^k \geq 0, \quad i \in N, k \in M. \tag{4.8}$$

The objective function (4.1) minimizes the total weighted earliness and tardiness penalties. Constraints (4.2) ensure that the start time of each operation must not be earlier than the time 0. Constraints (4.3) specify the precedence relations among the job's operations. Constraints (4.4) impose that each machine can process at most one operation at a time. Constraints (4.5) and (4.6) define the earliness and tardiness. Constraints (4.7) and (4.8) ensure that the earliness and tardiness only take non-negative values.

Although Problem$_{\mathcal{JIT}-\mathcal{JSS}}$ does not include any binary variables, and instead, uses the "logical or" constraints (4.4) (ILOG, 2017), it is still very difficult to solve. However, given a sequence (order) Π of performing the jobs' operations on the machines, it follows that either of the "logical or" constraints (4.4) holds. Therefore, constraints (4.4) turn into linear inequalities, which we denote by (4_Π), specifying the order given by Π on each machine. By substituting constraints (4.4) by (4_Π), Problem$_{\mathcal{JIT}-\mathcal{JSS}}$ is then a linear program (LP) for which the optimal completion time of the operations can be obtained in polynomial time. In addition, we may use Problem$_{\mathcal{JIT}-\mathcal{JSS}}$ to optimize a partial sequence, i.e., optimizing the sequence of a few operations at a time. If we let a small number of operations in the partial sequence, we may be able to deliver an optimal order of execution for those operations. This idea is used in Section 4.4 to develop a matheuristic algorithm. Next, we illustrate a small example of the JIT-JSS problem.

Example 1

Table 4.2 gives the processing order and time of jobs' operations on three machines (4×3 instance). Table 4.3 shows due-date of the operations and earliness and tardiness penalty coefficients. For example, operation 1 of job 1 has a due-date of 4 ($d_1^1 = 4$), and earliness and tardiness penalty coefficients of 0.29 and 0.30 per unit of deviation from the due-date ($\alpha_1^1 = 0.29$ and $\beta_1^1 = 0.30$). A feasible schedule for this instance is shown by the Gantt chart of Figure 4.1, in which (i, j) denotes the operation of job i on machine j, i.e., O_i^j. As the figure shows, four operations of O_1^2 and O_3^1 (both on M_2), and O_1^1 and O_2^1 (both on M_3) are early, i.e., they finish before their due-date. Here, O_4^1 is the only tardy operation that is performed on M_2. The rest of the operations finish on their due-dates. The objective function value of this schedule is equal to $z = \alpha_1^1 \times E_1^1 + \alpha_1^2 \times E_1^2 + \alpha_2^1 \times E_2^1 + \alpha_3^1 \times E_3^1 + \beta_4^1 \times T_4^1 = 0.29 \times 1 + 0.23 \times 1 + 0.10 \times 1 + 0.12 \times 1 + 0.80 \times 3 = 3.14$.

Table 4.2 : Technological order and processing time of the operations for 4×3 instance.

Job i	Operation 1 $\mathcal{M}(O_i^1)$ and p_i^1	Operation 2 $\mathcal{M}(O_i^2)$ and p_i^2	Operation 3 $\mathcal{M}(O_i^3)$ and p_i^3
1	M_3 and 3	M_2 and 3	M_1 and 3
2	M_3 and 5	M_2 and 4	M_1 and 2
3	M_2 and 2	M_3 and 1	M_1 and 3
4	M_2 and 3	M_1 and 4	M_3 and 1

Table 4.3 : Due-date and earliness and tardiness penalty coefficients of the operations for 4×3 instance.

Job i	Operation 1			Operation 2			Operation 3		
	d_i^1	α_i^1	β_i^1	d_i^2	α_i^2	β_i^2	d_i^3	α_i^3	β_i^3
1	4	0.29	0.30	7	0.10	0.15	15	0.21	0.89
2	9	0.23	0.29	12	0.26	0.75	20	0.22	0.77
3	9	0.12	0.69	15	0.18	0.95	18	0.13	0.30
4	12	0.26	0.80	24	0.22	0.17	25	0.29	0.51

4.4 Math_1 for JIT-JSS

In this section we discuss our first proposed matheuristic algorithm (i.e. Math_1) based on variable neighborhood search (VNS) for the JIT-JSS problem.

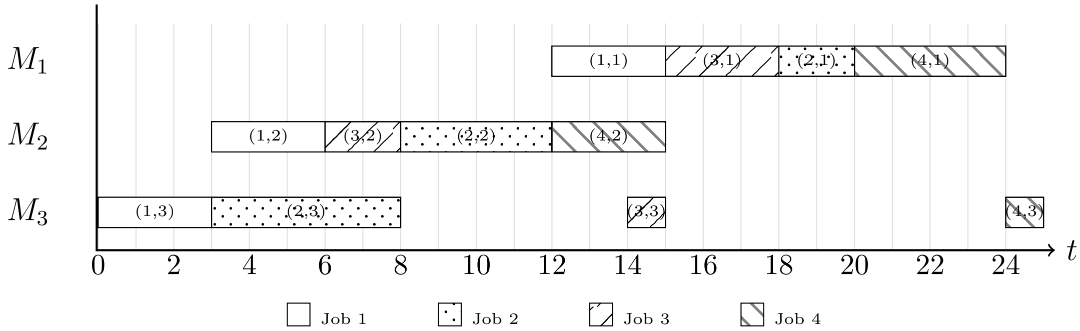

Figure 4.1 : A feasible schedule (of completing the operations) for 4×3 instance. A pair (i, j) represents execution of job i on machine j.

Given an initial sequence, the proposed matheuristic algorithm operates by decomposing JIT-JSS into sub-problems, i.e., smaller instances each with only a few operations and machines, delivering optimal or near optimal sequences of operations for the sub-problems and obtaining a feasible schedule (the completion time for the operations) for the complete problem. The algorithm forms the sub-problems by applying two neighborhoods. Then, the algorithm uses the available optimization solvers such as CPLEX (ILOG, 2017) to solve the generated sub-problems.

As discussed in Chapter 2 (see Section 2.5), Dos Santos et al. (2010) and Wang and Li (2014) applied heuristic algorithms to obtain a sequence and then solvers to determine the schedule. However, we use the solvers for two purposes: (1) obtaining the execution order of the operations in the sub-problem, and (2) determining the schedule for all operations. In this respect, the proposed algorithm is an "integrative" matheuristic (Raidl and Puchinger, 2008), in which VNS meta-heuristic works at the master level, controlling therefore the slave local search procedure. The local search includes optimizing the (reduced) mathematical program that includes only a few variables by the solver. One can cite the ease of implementation and utilizing the power of available solvers as the advantages of this approach, and restricting the estimation of lower bounds for performance assessment as its pitfall.

In what follows we first discuss encoding and decoding a sequence, and then we explain in detail each component of the proposed matheuristic algorithm.

4.4.1 Sequence encoding and decoding

We represent a sequence for JIT-JSS by using the operation-based encoding developed for the job-shop scheduling problem (Gen et al., 1994). We formally define a feasible sequence of the execution order of the operations on the machines as $\Pi = (\pi_1, \pi_2, \ldots, \pi_{n\times m})$, where π_1 is the first element of Π and $\pi_{n\times m}$ is the last element of Π. For an n-job and m-machine instance, this representation gives a total order over $n \times m$ operations, in which each job appears exactly m times, where the kth occurrence, $k \in \{1, \ldots, m\}$, of job i represents its kth operation denoted by O_i^k. It follows that a sequence admits a representation if and only if for each job $i \in N$ the order induced on $\mathcal{O}_i$ is exactly the one fixed by the instance, i.e., $(O_i^1, O_i^2, \ldots, O_i^m)$.

Consider the instance $I_{3\times4}$ given earlier in Section 4.3. The sequence presented as $\Pi = (1, 1, 2, 3, 2, 4, 3, 1, 3, 2, 4, 4)$ specifies the execution order of the operations of $I_{3\times4}$, where the first "1" ("2" or "3") represents the first operation of job "1" ("2" or "3"), and so on. That is,

$$\begin{array}{lcccccccccccc}
\Pi = & (\mathbf{1} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{2} & \mathbf{4} & \mathbf{3} & \mathbf{1} & \mathbf{3} & \mathbf{2} & \mathbf{4} & \mathbf{4}) \\
 & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
 & M_3 & M_2 & M_3 & M_2 & M_2 & M_2 & M_3 & M_1 & M_1 & M_1 & M_1 & M_3
\end{array}$$

In order to deduce a schedule from sequence Π, we utilize the solver CPLEX. Given sequence Π, it follows that for each $j \in M$, the operations in $\mathcal{O}(M_j)$ are executed in the order induced by Π on $\mathcal{O}(M_j)$. This implies that only one of the "logical or" constraints (4.4) now holds due to the execution order of the operations that is given by Π. That order is expressed by constraints 4_Π. It is also clear that by replacing constraints (4.4) by (4_Π), Problem$_{\mathcal{JIT}-\mathcal{JSS}}$ turns into an LP.

As an example, Table 4.4 presents constraints (4_Π) for the instance $I_{3\times4}$, where Π is given as above. Solving the resulting LP by CPLEX leads to an optimal schedule for Π, which is illustrated in Figure 4.1.

It should be noted that even though the schedule returned by CPLEX is optimal for the given sequence Π, it is possible that more than one schedule is associated with Π if multiple optimal schedules exist for Π. Moreover, because there is no

Table 4.4 : Constraints (4_Π) for the instance $I_{3\times 4}$ that are induced by $\Pi = (1,1,2,3,2,4,3,1,3,2,4,4)$ on $\mathcal{O}(M_j), j = 1,2,3$.

M_1	M_2	M_3
$C_3^3 \geq C_1^3 + p_3^3$	$C_3^1 \geq C_1^2 + p_3^1$	$C_2^1 \geq C_1^1 + p_2^1$
$C_2^3 \geq C_3^3 + p_2^3$	$C_2^2 \geq C_3^1 + p_2^2$	$C_3^2 \geq C_2^1 + p_3^2$
$C_4^2 \geq C_2^3 + p_4^2$	$C_4^1 \geq C_2^2 + p_4^1$	$C_4^3 \geq C_3^2 + p_4^3$

deadline and hard time window for performing the operations in JIT-JSS, every sequence admits a schedule.

4.4.2 Generating an initial sequence

The VNS algorithm that will be discussed in Section 4.5.1 requires an initial sequence, i.e., a feasible order of executing the operations on the machines. For this purpose, we implement two methods of Giffler Thompson (GT) and Shifting Bottleneck Heuristic (SBH), and we initialize VNS with the initial sequence generated by one of them. The probability of GT being selected to start the VNS algorithm with is 0.6 and that of SBH is 0.4. Therefore, the methods GT and SBH do not have the equal chance to be selected to generate an initial sequence. The reason for including two methods with different probabilities lies in performing five runs for the VNS algorithm. We will later discuss this in more details in Section 4.4.6. Let Π denote the initial sequence generated by either GT or SBH. As detailed in Section 4.4.1, we replace constraints (4.4) by (4_Π) and solve Problem$_{\mathcal{JIT}-\mathcal{JSS}}$ by CPLEX. That process leads to an optimal schedule for Π with the objective function value $z(\Pi)$. We use Π and $z(\Pi)$ to initialize Math_1.

Next, we explain GT and SBH methods.

The Giffler Thompson method

The well-known GT constructive algorithm (Giffler and Thompson, 1960) can be used to generate an initial sequence Π for JIT-JSS. Given the earliest completion time C^* among all schedulable operations (whose predecessors have already been scheduled) and its associated machine M^*, GT determines all operations that can be started prior to C^*. Any conflict among the operations is then settled by using a dispatching rule, e.g., the random selection (as in the original GT), or the earliest due-date (EDD) first (as in the present study). The process continues until all operations are scheduled. Algorithm 4.1 summarizes GT.

Algorithm 4.1: The Giffler Thompson (GT) algorithm.

Input: An instance of the JIT-JSS problem.
Output: A feasible sequence of operations on the machines for the input instance.

Initialization: $F = \{O_i^1 | i \in N\}$, i.e., adding the first operation of each job to set F; the empty sequence Π of size $n \times m$; $ES_i^1 = 0$ and $EC_i^1 = p_i^k, \forall O_i^1 \in F$.
while *F is not empty* **do**
    Find the earliest completion time C^* and its associated machine M^* for the operations in F (i.e., $C^* = \min_{O_i^k \in F} \{EC_i^k\}$);
    $F' = \{O_i^k \in F | ES_i^k < C^*, \mathcal{M}(O_i^k) = M^*\}$ (building the conflict set);
    Select $O' \in F'$ to be scheduled next (by using the earliest due-date (EDD) first dispatching rule) and append it to Π;
    Remove O' from F, and add its immediate job successor (if any) to F;
    Update the earliest start and completion times for all $O_i^k \in F$;
end
return Π;

The Shifting Bottleneck Heuristic method

An initial sequence Π can alternatively be generated by the SBH algorithm (Adams et al., 1988). SBH has been applied for various job-shop scheduling problems, see for example, Mason et al. (2002); Mönch and Drießel (2005) and Mönch et al. (2007). SBH aims to minimize the maximum completion time of all operations. It selects the machine with the maximum lateness as the "bottleneck" and re-sequences the jobs on previously scheduled machines according to the bottleneck. The process continues until no unscheduled machines remain. The SBH procedure is summarized in Algorithm 4.2. In Algorithm 4.2, the conjunctive arcs show the order of operations to be processed on the machines and the disjunctive arcs denote the pairs of operations that must be executed on the same machine, yet their order to be decided. The graph with the disjunctive and conjunctive arcs is known the disjunctive graph, where the operations are shown by vertices.

Algorithm 4.2: The Shifting Bottleneck Heuristic (SBH) algorithm.

Input: An instance of the JIT-JSS problem.
Output: A feasible sequence of operations on the machines for the input instance.
Initialization: Set $M' = \{\}$ of scheduled machines and graph G with all the conjunctive arcs (no disjunctive arcs); the empty sequence Π of size $n \times m$.
while $M \neq M'$ **do**
  Step 1: Searching the machines yet to be scheduled.
  for $M_j \in M \setminus M'$ **do**
    Generate an instance of $1|r_i|L_{max}$ for M_j; `// i.e., minimize the maximum lateness on a given machine subject to the jobs' release time`
    Compute the maximum lateness on machine M_j (i.e., $L_{max}(M_j)$);
  end
  Step 2: Bottleneck selection and sequencing.
  Let $M_u \in \underset{M_j \in M \setminus M'}{\arg\max} \, (L_{max}(M_j))$;
  Sequence M_u according to sequence obtained for $1|r_i|L_{max}$ and update G by fixing the disjunctive arcs for M_u;
  $M' = M' \cup \{M_u\}$.
  Step 3: Re-sequencing the already scheduled machines.
  for $M_j \in M' \setminus \{M_u\}$ **do**
    Delete disjunctive arcs for M_j from G;
    Form $1|r_i|L_{max}$ for M_j, find the sequence that minimizes $L_{max}(M_j)$ and insert the corresponding disjunctive arcs in graph G;
  end
end
Specify the final order of jobs on each machine from G and update Π accordingly;
return Π;

4.4.3 The improvement algorithm

Our improvement engine, within the proposed matheuristic, is the VNS algorithm. VNS is a meta-heuristic, which systematically changes the neighborhood structures to avoid being trapped in local optima (Mladenović and Hansen, 1997).

VNS includes two major steps: Shake and local search. Starting from an initial sequence, in the shake phase, the algorithm randomly generates a neighbor S' from one of the already defined neighborhood structures. Then, S' is passed to the local search phase for improvement. The improved S'' is replaced by the current best sequence, and the local search continues. If no improvement is observed, the algorithm returns to the shake phase. The algorithm continues until the stopping criterion is met. Algorithm 4.3 outlines the proposed Math_1. In the local search phase of Math_1 we apply two relaxation-based neighborhoods denoted by N_1 and N_2 (to be discussed shortly).

It should be noted that our proposed Math_1 differs from the traditional VNS in the following ways: (1) we only use one neighborhood structure, and that N_1 for the shake phase, (2) we explore several neighbor sequences in each neighborhood structure, precisely n_c neighbor sequences, rather than only one (see `Sub-procedure LS` in Algorithm 4.3), and S' is updated if an improved sequence is explored (in which the search also continues for n_c iterations, i.e., the number of visiting neighbors is determined by n_c), and (3) while in the traditional VNS once an improved neighbor is obtained, the search continues from the first neighborhood, we continue the search with the next neighborhood structure.

Next, we explain the relaxation neighborhoods.

4.4.4 Relaxation neighborhoods

In this section, we explain the idea of relaxation neighborhoods. We start by introducing the general concept and that how the relaxation neighborhoods differ from their traditional counterparts. Then, we explain the parameters and the pseudo-code (Algorithm 4.4) of the relaxation neighborhoods, followed by an illustrative example.

Consider the sequence Π. At each execution of the relaxation neighborhoods, a small sub-set $R \subset \mathcal{O}$ of operations is chosen to be relaxed. That is, the order imposed by Π is relaxed for the operations in R, meaning that they are subject to re-ordering for possible improvement. However, order of the remaining non-relaxed operations, i.e., $NR = \mathcal{O} \setminus R$ is kept unchanged. As detailed in Section 4.3 we use the solver CPLEX to solve Problem$_{\mathcal{JIT}-\mathcal{JSS}}$ given R and NR, which leads to

Algorithm 4.3: Math_1 for the JIT-JSS problem.

Input: An initial sequence Π and its associated objective function value $z(\Pi)$; a set of neighborhood structures $N_\kappa, \kappa = 1, 2$ to be used in the local search.
Output: An improved sequence.

while *the stopping condition is not met* **do**
    for $i := 1$ *to* 2 **do**
        Shake:
        $\Pi' \leftarrow N_1(\Pi)$;
        Local search:
        $\Pi'' \leftarrow$ `LS`(Π', i);
        if $z(\Pi'') < z(\Pi)$ **then**
            $\Pi := \Pi''$;
            $z(\Pi) := z(\Pi'')$;
        end
    end
end
return Π;

`Sub-procedure LS(`Π'`, `κ`) //the local search in VNS`
if $\kappa = 1$ **then**
    for $f := 1$ *to* n_c **do**
        $temp \leftarrow N_1(\Pi')$;
        if $z(temp) < z(\Pi')$ **then**
            $\Pi' := temp$;
            $z(\Pi') := z(temp)$;
        end
    end
else
    for $f := 0$ *to* $n_c - 1$ **do**
        $temp \leftarrow N_2(\Pi', f, n_c)$;
        if $z(temp) < z(\Pi')$ **then**
            $\Pi' := temp$;
            $z(\Pi') := z(temp)$;
        end
    end
end
return Π';

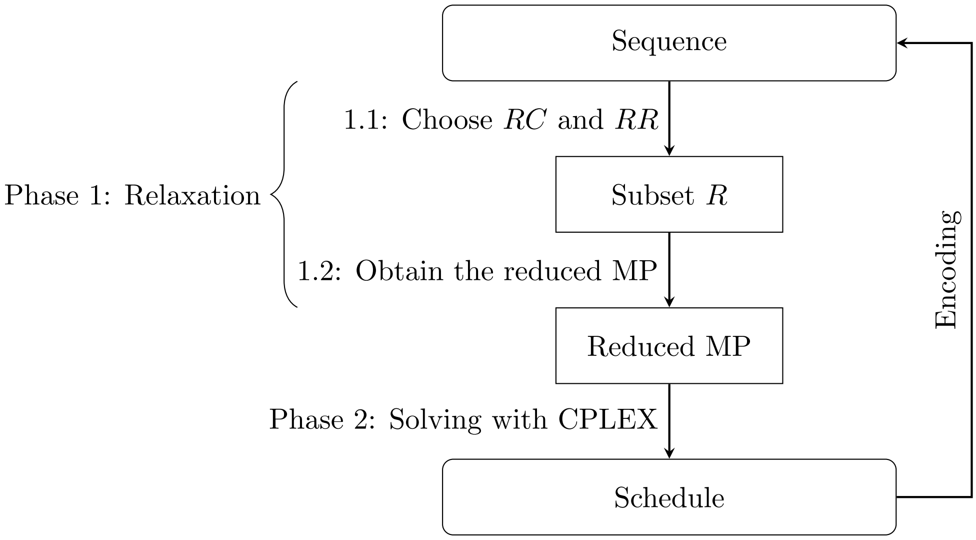

Figure 4.2 : The global process of relaxation neighborhoods.

possible re-ordering and scheduling of the operations in R, and only re-scheduling of the operations in NR. We stop the solver after T_{limit} seconds, and if a new schedule is obtained, its associated sequence is re-encoded and the search continues. The process of relaxation neighborhood is illustrated in Figure 4.2.

Dos Santos et al. (2010) and Wang and Li (2014) used recombination operators and swapping and insertion moves to generate new (improved) sequences of performing the operations. Such a sequence implies that the order of operations' execution is known, turning therefore constraints (4.4) in Problem$_{\mathcal{JIT}-\mathcal{JSS}}$ into linear inequities. Then, the schedule, i.e., the completion time of the operations is obtained through solving Problem$_{\mathcal{JIT}-\mathcal{JSS}}$ by the available solvers, e.g., CPLEX. Two major shortcomings of those procedures include (1) ignoring the impact of the moves on the rest of the sequence (because the manipulations are mostly applied either randomly or myopically), which leads to low quality schedules, and (2) utilizing the solvers only to deliver the optimal schedule for a given sequence, instead of using the solver for both sequencing and scheduling.

Salehipour and Ahmadian (2017) and Salehipour et al. (2018) proposed novel relaxation neighborhoods in the context of aircraft landing problem, which aim to destruct a sequence of landing aircraft and construct a new (improved) sequence, in order to overcome those limitations. Those relaxation neighborhoods take a sequence as input and guide an exact solver to only re-order the landing of a subset of aircraft. We follow a similar process for solving JIT-JSS. Also, regardless of only re-ordering the operations in R, the schedule is obtained for all the operations, i.e., for the operations in R and NR, minimizing therefore the impact of the random and myopic moves. In addition, CPLEX is used for both sequencing and scheduling, through re-ordering the operations in R and scheduling all operations. Particularly, we keep a few operations in R so that the solver may efficiently deliver the optimal order for executing the operations in R.

The generation of R, i.e., the set of consecutive operations to be relaxed in Π, is controlled by two parameters: "relaxation center", denoted by $RC \in \mathbb{Z}^+$, that determines the operation positioned in the middle of R, and "relaxation radius", represented by $RR \in \mathbb{Z}^+$, that defines the number of operations to the left and to the right of RC. We implement two variants of relaxation neighborhoods. In the first variant, which is denoted by N_1, the parameter RC is randomly selected. We utilize N_1 both in the shake and in the local search phases. Contrary to N_1, the second variant, i.e., N_2, which is only used in the local search phase, starts from the beginning of a given sequence and progressively relaxes a number of operations at a time until it reaches the end of the sequence. Within N_1 and N_2, a reduced MP for the given sequence Π is formed (see Step 3 in Algorithm 4.4). In particular, for each machine j, the step identifies the relaxed and non-relaxed operations (denoted by R_j and NR_j) and introduces constraints (4.4) and (4_{NR_j}) accordingly. Then, in order to restore the connectivity between R_j and NR_j, constraints (4.9) and (4.10) are added. We formally explain N_1 and N_2 in Algorithm 4.4.

4.4.5 Re-encoding scheme

It follows that only the order of the operations in R can be changed by the solver, whereas the operations in NR that are processed on the same machine are executed in the same order induced by the given sequence Π. Because there are no binary variables in Problem$_{\mathcal{JIT}-\mathcal{JSS}}$ that determine the relative order of executing the operations on the machines, the order of the operations in R in the new schedule cannot be deduced directly. In order to obtain a sequence from the newly delivered schedule, we sort the operations belonging to R in non-decreasing order of their completion time (we keep the order of the operations in NR unchanged). The ties

Algorithm 4.4: Relaxation neighborhoods N_κ, $\kappa \in \{1,2\}$.

Input: A sequence $\Pi = (\pi_1, \pi_2, \ldots, \pi_{n\times m})$; parameters f, n_c (for N_2).
Output: A sequence.

Step 1: Set the relaxation center, relaxation radius and time limit.
Calculate RC (for N_2 use f and n_c), RR, T_{limit} according to Table 4.6;

Step 2: Formation of the relaxed and non-relaxed sub-sets.
Let R and NR specify relaxed and non-relaxed subsets where
$R = \{\pi_r | \max\{1, RC - RR\} \leq r \leq \min\{n \times m, RC + RR\}\}$ and $NR = \mathcal{O} \setminus R$;

Step 3: Form the reduced MP.
Create constraints (4.1)–(4.3),(4.5)–(4.8),(4.9), (4.10) for all the operations;

`//Imposing resource and connectivity constraints among operations in` R `and` NR
for $j \in M$ **do**
    Let $R_j = R \cap \mathcal{O}(M_j)$ and $NR_j = (\mathcal{O} \setminus R) \cap \mathcal{O}(M_j)$;
    Add the "logical or" constraints (4.4) for each $(O_a, O_b) \in R_j^2$ (i.e., the machine precedence to be decided among the relaxed operations in R_j);
    Add the linear constraints (4_{NR_j}) for the order induced by Π on NR_j (i.e., deciding the machine precedence among the non-relaxed operations);
    O_{pre} (if exists) denotes the last operation of NR_j in the sequence Π, preceding operations in R_j;
    O_{suc} (if exists) represents the first operation of NR_j in the sequence Π, succeeding operations in R_j;

    for $O_i^k \in R_j$ **do**

$$\text{If } O_{pre} \text{ exists, generate constraint } C_{pre} \leq C_i^k - p_i^k; \tag{4.9}$$

$$\text{If } O_{suc} \text{ exists, generate constraint } C_{suc} \geq C_i^k + p_{suc}; \tag{4.10}$$

    end
end
Step 4: Re-sequencing and scheduling.
Solve the reduced MP (see Step 3) by CPLEX. Interrupt the solving procedure after T_{limit} seconds; Encode the sequence according to the generated schedule;

Step 5: Return the sequence constructed in Step 4;

can be broken arbitrarily for the operations with the same completion time. This implies that several sequences might be associated with a schedule. We, however, re-encode only one of those sequences to represent the schedule.

Next, we illustrate a small example to elaborate the relaxation neighborhoods and the formation of the sets of relaxed and non-relaxed operations.

Example 2

Consider the instance $I_{3\times 4}$ discussed in Section 4.3. Assume that the following initial sequence Π is given. The associated schedule for Π obtained by CPLEX is illustrated in Figure 4.1.

$$\begin{array}{rcccccccccccc} \Pi = & (1 & 1 & 2 & \mathbf{3} & \mathbf{2} & \mathbf{4} & 3 & 1 & 3 & 2 & 4 & 4) \\ & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ & M_3 & M_2 & M_3 & M_2 & M_2 & M_2 & M_3 & M_1 & M_1 & M_1 & M_1 & M_3 \end{array}$$

Phase 1: Relaxation

Assume that $RC = 5$ and $RR = 1$, which leads to $R = \{O_3^1, O_2^2, O_4^1\}$ and $NR = \{O_1^1, O_1^2, O_2^1, O_3^2, O_1^3, O_3^3, O_2^3, O_4^2, O_4^3\}$. Figure 4.3 illustrates the process of relaxation. The nodes that show the operations in R are shown in green and yellow (the green node denotes the relaxation center). The non-relaxed operations and their execution order on machines 1 and 3 is represented by NR. Constraints (4_{NR}) which are shown in columns 1 and 3 in Table 4.4, determine the execution order and the completion time for the operations in NR. The dashed arcs in Figure 4.3 denote the relaxed operations contained in R, which can be expressed by constraints (4.4) as following:

$$\begin{aligned} C_3^1 \geq C_2^2 + p_3^1 \quad &\textbf{or} \quad C_2^2 \geq C_3^1 + p_2^2, \\ C_2^2 \geq C_4^1 + p_2^2 \quad &\textbf{or} \quad C_4^1 \geq C_2^2 + p_4^1, \\ C_4^1 \geq C_3^1 + p_4^1 \quad &\textbf{or} \quad C_3^1 \geq C_4^1 + p_3^1. \end{aligned}$$

Following Algorithm 4.4, in order to restore the connectivity between the operations in R and NR, constraints (4.9) and (4.10) must be introduced. To this end, two operations O_{pre} and O_{suc} must be identified on every machine on which the operations in R are executed. In this example, the operations in R are executed on

M_2. Also, it follows that the last operation in NR that precedes the operations in R is $O_{pre} = O_1^2$, which is shown by the red node in Figure 4.3. As a result, constraints (4.9) are instantiated as follows:

$$C_1^2 \leq C_3^1 - p_3^1,$$
$$C_1^2 \leq C_2^2 - p_2^2,$$
$$C_1^2 \leq C_4^1 - p_4^1.$$

We show those three constraints in Figure 4.3 by three black arcs leaving the red node $(1,2)$, ensuring that $O_{pre} = O_1^2$ precedes the relaxed operations on M_2. We note that we did not introduce constraints (4.10) because no operation O_{suc} on M_2 exists.

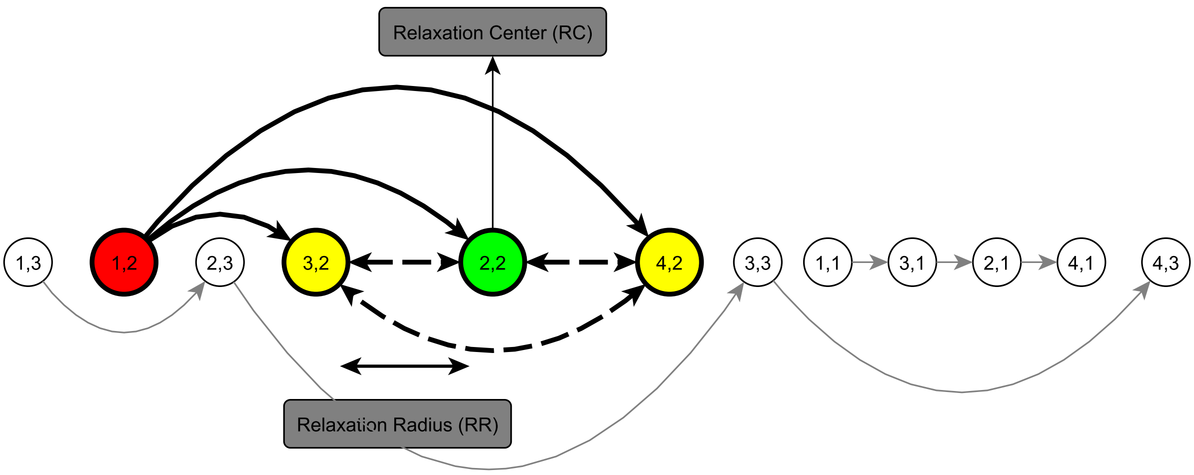

Figure 4.3 : An example of the relaxation neighborhood for 4×3 instance. A pair (i, j) represents job i on machine j. The relaxed jobs are represented by green and yellow (the green vertex shows the relaxation center). Job shown in red is immediate predecessor of the relaxed sub-sequence.

Phase 2: Solving with CPLEX

In this phase, the reduced MP obtained in phase 1 is solved by CPLEX. Assume that CPLEX is stopped after 1 second of running and the best schedule explored is reported. Furthermore, suppose that within 1 second, CPLEX explored all the neighbor sequences detailed in Table 4.5 (since R includes 3 operations, there are

6 distinct orders for the operations in R, including the incumbent sequence). It follows that neighbor 1 leads to the best schedule because it has the least objective function value, shown by the * in the table.

Table 4.5 : The objective function values of the incumbent and neighbor sequences as explored by CPLEX for the reduced MP.

Neighbor	Sequence	Objective value (z)
Incumbent	(1, 1, 2, **3**, **2**, **4**, 3, 1, 3, 2, 4, 4)	3.14
1	(1, 1, 2, **3**, **4**, **2**, 3, 1, 3, 2, 4, 4)	3*
2	(1, 1, 2, **2**, **3**, **4**, 3, 1, 3, 2, 4, 4)	10.82
3	(1, 1, 2, **2**, **4**, **3**, 3, 1, 3, 2, 4, 4)	19.39
4	(1, 1, 2, **4**, **3**, **2**, 3, 1, 3, 2, 4, 4)	4.8
5	(1, 1, 2, **4**, **2**, **3**, 3, 1, 3, 2, 4, 4)	11.61

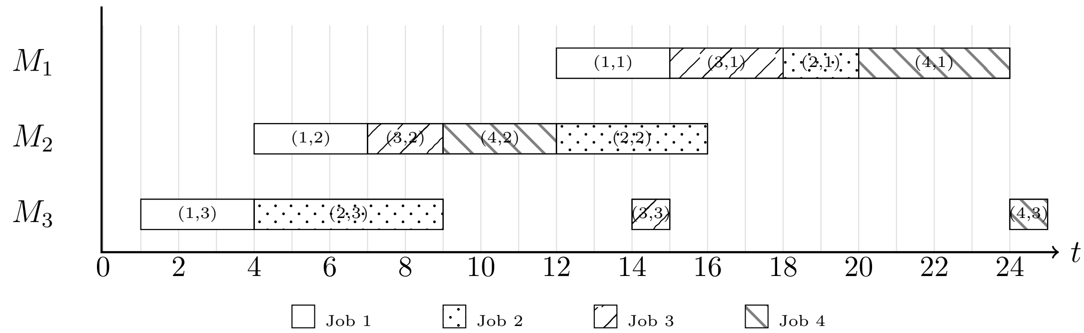

Figure 4.4 : A feasible schedule for the instance $I_{4\times3}$. Pair (i, j) represents job i on machine j.

The schedule of neighbor 1 obtained by CPLEX is shown by the Gantt chart in Figure 4.4. To obtain the new sequence, the relaxed operations are sorted in non-decreasing order of their completion time while the order of the non-relaxed operations is kept unchanged:

$$
\begin{array}{ccccccccccccc}
 & M_3 & M_2 & M_3 & M_2 & M_2 & M_2 & M_3 & M_1 & M_1 & M_1 & M_1 & M_3 \\
 & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\
\Pi' = & (1 & 1 & 2 & \mathbf{3} & \mathbf{4} & \mathbf{2} & 3 & 1 & 3 & 2 & 4 & 4) \\
 & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
C_i^k\!: & 4 & 7 & 9 & \mathbf{9} & \mathbf{12} & \mathbf{16} & 15 & 15 & 18 & 20 & 24 & 25
\end{array}
$$

4.4.6 Computational results

We evaluate Math_1 on the 72 benchmark instances of Baptiste et al. (2008). Those instances include three different sizes for jobs and machines, where $n \in \{10, 15, 20\}$ and $m \in \{2, 5, 10\}$, and therefore, the instances range from 20 to 200 operations. For each combination of n and m eight instances were generated. We refer the interested reader to Baptiste et al. (2008) for details.

Instances

The 72 benchmark instances of JIT-JSS provided in Baptiste et al. (2008) range from 20 to 200 operations. Each instance is named using the format `n-m-DD-W-ID`, where $n \in \{10, 15, 20\}$ and $m \in \{2, 5, 10\}$ denote the numbers of jobs and machines, respectively. For each combination of n and m, i.e., $\binom{\{\{10,15,20\},\{2,5,10\}\}}{\{n,m\}}$, eight instances were generated by randomly choosing `DD`, `W`, and `ID`. In particular,

- if the difference between the due-dates of consecutive operations of the same job, i.e., `DD`, is equal to the processing time of the last operation, then `DD = tight`. If it is equal to the processing time of the last operation plus a random value in the range $[0, 10]$, then `DD = loose`.
- `W`, which is the relation between the earliness and tardiness penalty coefficients, is either `equal` if both α and β are chosen randomly in the range $[0.1, 1]$, or `tard` if α is taken in the range $[0.1, 0.3]$ and β in the range $[0.1, 1]$.
- `ID` is either 1 or 2, indexing either of the two instances generated for each combination of the other parameters.

We code Math_1 in the C++ programming language (for the implementation of SBH, we used the code provided by Applegate and Cook (1991), which is publicly available at `http://www.math.uwaterloo.ca/~bico/jobshop/`). We implement Problem$_{\mathcal{JIT}-\mathcal{JSS}}$ by using the CPLEX Concert Technology version 12.6.0 (ILOG, 2017) with the default parameters, except the time limit and the number of processors (threads). Within Math_1 we only use one thread for CPLEX, and in the stand alone CPLEX we utilize four threads. We perform the computational experiments on a Personal Computer with Intel® Core™ i5-430M CPU clocked at 2.26GHz with 4GB of memory under Windows 10 operating system.

The algorithm terminates if one of the following three criteria is met: (1) the maximum number of iterations is reached; we set this to 10; (2) the maximum number of iterations without an improvement is recorded, which is set to 2 (i.e., if the algorithm cannot deliver an improved solution after 2 consecutive iterations); and, (3) the maximum computation time is elapsed; we set this to 200 seconds. Table 4.6 details the value of parameters that we use in the algorithm. The parameters n_c and T_{limit} are chosen according to the problem size. We choose the value of parameters RC and RR upon each execution of N_1 or N_2 according to the distribution functions given in Table 4.6 and Figure 4.5. In the table, $R(a, b)$ denotes a discrete uniform random number in the ranges $[a, b]$. In Figure 4.5, a sequence is broken down into three sections (shown by different patterns). For example, an operation from the first or last 20% of operations might be chosen as RC with the probability of 0.2.

Table 4.6 : Value of the parameters.

Parameter	N_1	N_2
n_c	15	$= \begin{cases} 5, & \text{if } n \times m < 100 \\ 10, & \text{if } n \times m \geq 100 \end{cases}$
RR	$\dfrac{n \times m}{R(5,10)}$	$= \begin{cases} 5, & \text{if } n \times m < 50 \\ 10, & \text{if } 50 \leq n \times m \leq 150 \\ 15, & \text{if } n \times m > 150 \end{cases}$
RC	See Figure 4.5	$\dfrac{f \times n \times m}{n_c} + R(0,5)$
T_{limit} (sec)	1	$= \begin{cases} 1, & \text{if } n \times m < 50 \\ 2, & \text{if } 50 \leq n \times m \leq 100 \\ 3, & \text{if } n \times m > 100 \end{cases}$

We compare the algorithm's outcomes and the best solution obtained by the state-of-the-art methods that solved the same instances, as well as the solver CPLEX.

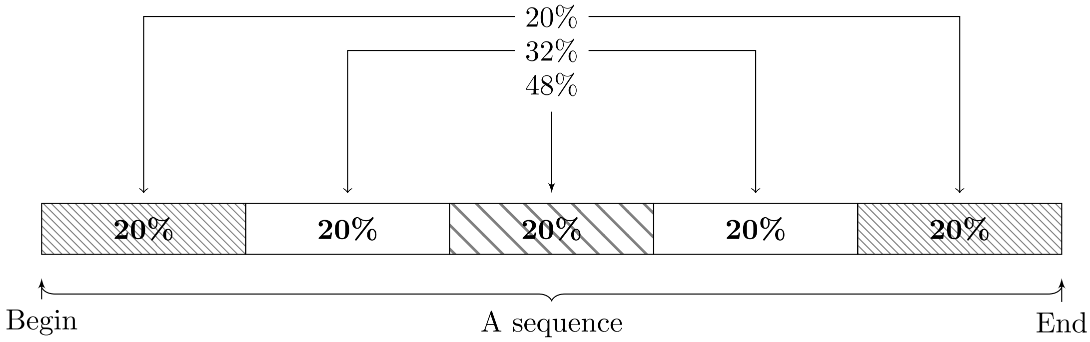

Figure 4.5 : Distribution function to choose the value of parameter RC for N_1.

Those state-of-the-art methods include (1) the LNS of Laborie and Godard (2007) (only available for instances with 15 and 20 jobs), (2) the CP and LNS of Monette et al. (2009), (3) the EA of Dos Santos et al. (2010), (4) the GA of Yang et al. (2012a) (only available for instances with 10 and 15 jobs), and (5) the VNS of Wang and Li (2014).

We show the details of the results in Table 4.7. The outcomes include the best results over 5 runs obtained by Math_1, the previous methods and the solver CPLEX. The first and second columns show the name of the instances and the number of operations. The third and fourth columns report the best objective function values obtained by CPLEX and previous methods. Recall that the outcomes of CPLEX are obtained by using four threads; we also set the time limit of 1,800 seconds for the stand alone CPLEX. The fifth and sixth columns present the best objective function values and computation times of Math_1. The value of time is averaged over five runs. The reason for five runs is twofold. First, to mitigate, to some extent, the fluctuations in the performance of CPLEX. In particular, because we use CPLEX heuristically (by setting a time limit) it may not deliver the same solution for an instance over different algorithm's runs. Second, because there are certain randomized components in Math_1, the multiple-run is a reasonable strategy to ensure a fair level of performance. The last two columns of Table 4.7 show the amount of improvement over CPLEX (Impr 1 in %) and over the previous methods (Impr 2 in %). The values of Impr 1 and 2 are calculated as $\frac{z'-z}{z'} \times 100$, where z' is the objective function value reported by either CPLEX (for Impr 1) or previous methods (for Impr 2), and z is that of Math_1. Therefore, the positive values indicate the improved solutions obtained in this study. The value of zero means that Math_1 obtains the same solution as of CPLEX or the state-of-the-art methods. In the table, the best objective function values are highlighted.

According to Table 4.7, despite the long computation time of CPLEX (1,800 seconds), CPLEX underperforms Math_1 for 51 instances, i.e., for about 71% of instances. In addition, Math_1 obtains superior solutions to the previous methods for 40 instances, out of 72; in other words, for nearly 56% of the instances. The matheuristic also produces the same quality solution as of the previous methods for 20 instances. Overall, Math_1 delivers solutions that are at least as good as the best available ones in the literature for almost 84% of the instances. We note that while each run of our matheuristic algorithm takes at most three minutes, on average, the quality solutions reported in the table are obtained within almost 15 minutes because we run the algorithm for 5 times.

While the amount of improvement yielded by Math_1 is still considerable for instances with 10 jobs, it is significant for larger instances with 15 and 20 jobs. Additionally, a large number of new best solutions is obtained for those instances. For example, the algorithm delivers 16 and 19 new best solutions for instances with 15 and 20 jobs. Therefore, Math_1 obtains a total number of 35 new best solutions for those 48 instances, where the average amount of improvement is nearly 5%. The amount of improvement over CPLEX is far greater and is in the order of almost 43%.

It should be noted that the run time of some of the state-of-the-art methods was not reported. For example, while each run of the algorithms proposed by Monette et al. (2009) was reported to take 600 seconds, the run time of the VNS of Wang and Li (2014) that is among the top performing algorithms for JIT-JSS of its time is not available. Therefore, we did not perform a method-wise comparison, and instead, carried out an instance-wise evaluation, which indicates that our proposed method offers an efficient solution approach for the problem. It is worth re-emphasizing that the state-of-the-art methods utilize a broad range of heuristic and meta-heuristic algorithms, as well as solvers, and it has been a challenge to overcome the best previous outcomes.

We also compare the Math_1's solutions and the best solutions of the literature and those of solver CPLEX though their values of gap from the best lower bound. We detail the values of the lower bound and those of the calculated gaps in Table 4.8. The first column denotes the name of the instances. Columns under the headings "LBR" and "LBP", which are due to Baptiste et al. (2008), report the values of the lower bound given by the Lagrangian resource constraints relaxation and the Lagrangian precedence constraints relaxation, respectively. As Baptiste et al. (2008) set the time limit for their Lagrangian relaxations, some lower bounds are therefore not available because their algorithm was stopped before delivering the lower bounds. Such cases have been shown with the "-" in Table 4.8. Also, column "LBRP" lists the lower bounds by the job-level and machine-level Lagrangian relaxation of Tanaka et al. (2015) which are given only for instances with 10 jobs. The column "Best LB" denotes the best value of the lower bound over the available schemes. The last three columns report the percentage of gap between the solution of a method and the best LB, where the gap of an instance is calculated as $|\frac{Best\ LB - z}{Best\ LB}| \times 100$, and z denotes the objective value of the tested method.

As Table 4.8 shows, Math_1 leads to the smallest average gap, which is around 13%, whereby the average gap of CPLEX is nearly 30% and that of the best of the literature is around 15%.

4.5 Math_2 for JIT-JSS

So far we designed Math_1 armed with novel relaxation neighbourhoods. We also showed the how our proposed mathueristic can obtain superior solutions by updating the best known solution for nearly 56% of instances. However, the following points remained unaddressed:

- In Section 4.4.4 several differences between conventional VNS and our proposed method (such as using only one neighbourhood in the shake) were listed. Yet their merit was never explored;
- Although major shortcomings of traditional manipulation techniques compared to relaxation neighbourhoods were discussed (see Table 4.7), the impact of our novel neighbourhoods on the solution quality was not examined.

As a result in this section we propose another variant of VNS (i.e. Math_2) which

Table 4.7 : The detailed computational results (for experiments CPLEX 12.6.0 was used).

Instance	$n \times m$	CPLEX	Best of literature	Math_1	Time	Impr 1	Impr 2
tight-equal-1-10x2	20	**461.96**	**461.96**	**461.96**	6.06	0.00	0.00
tight-equal-2-10x2	20	**448.32**	**448.32**	**448.32**	6.93	0.00	0.00
tight-equal-1-10x5	50	722.61	**689.11**	**689.11**	11.28	4.64	0.00
tight-equal-2-10x5	50	779.18	**763.24**	**763.24**	9.63	2.05	0.00
tight-equal-1-10x10	100	**1276.23**	1277.44	**1276.23**	21.10	0.00	0.09
tight-equal-2-10x10	100	1887.24	1878.26	**1866.92**	21.75	1.08	0.60
loose-equal-1-10x2	20	**224.84**	**224.84**	**224.84**	7.47	0.00	0.00
loose-equal-2-10x2	20	324.43	**319.37**	324.43	4.18	0.00	-1.58
loose-equal-1-10x5	50	**1721.54**	1740.08	1767.07	15.89	-2.64	-1.55
loose-equal-2-10x5	50	971.96	**967.73**	971.96	8.01	0.00	-0.44
loose-equal-1-10x10	100	364.39	364.39	**360.74**	17.80	1.00	1.00
loose-equal-2-10x10	100	**249.85**	**249.85**	**249.85**	24.63	0.00	0.00
tight-tard-1-10x2	20	**179.46**	**179.46**	**179.46**	6.86	0.00	0.00
tight-tard-2-10x2	20	**145.37**	**145.37**	**145.37**	5.45	0.00	0.00
tight-tard-1-10x5	50	**385.93**	387.3	387.3	12.79	-0.35	0.00
tight-tard-2-10x5	50	**627.45**	632.67	635.93	9.72	-1.35	-0.52
tight-tard-1-10x10	100	**668.14**	687.65	687.65	23.48	-2.92	0.00
tight-tard-2-10x10	100	783.47	779.3	**777.85**	18.38	0.72	0.19
loose-tard-1-10x2	20	**416.44**	**416.44**	**416.44**	3.91	0.00	0.00
loose-tard-2-10x2	20	**137.94**	**137.94**	**137.94**	7.80	0.00	0.00
loose-tard-1-10x5	50	**175.08**	**175.08**	**175.08**	13.30	0.00	0.00
loose-tard-2-10x5	50	518.33	**499.93**	504.36	12.61	2.70	-0.89
loose-tard-1-10x10	100	**375.71**	383.86	**375.71**	19.27	0.00	2.12
loose-tard-2-10x10	100	**144.94**	**144.94**	**144.94**	12.22	0.00	0.00
tight-equal-1-15x2	30	3400.13	**3344.54**	**3344.54**	55.14	1.63	0.00
tight-equal-2-15x2	30	1496.92	**1479.76**	**1479.76**	35.36	1.15	0.00
tight-equal-1-15x5	75	1341.29	1363.08	**1318.68**	72.89	1.69	3.26
tight-equal-2-15x5	75	**2670.97**	2693.24	2897.51	67.22	-8.48	-7.58
tight-equal-1-15x10	150	7665.92	**6848.97**	6950.03	102.19	9.34	-1.48
tight-equal-2-15x10	150	5352.15	5365.23	**4750.25**	119.92	11.25	11.46
loose-equal-1-15x2	30	1066.64	1041.7	**1041.33**	25.61	2.37	0.04
loose-equal-2-15x2	30	522.22	**497.97**	505.16	22.15	3.27	-1.44
loose-equal-1-15x5	75	3280	3267.79	**3207.45**	100.36	2.21	1.85
loose-equal-2-15x5	75	3449.47	3357.13	**3276.3**	81.20	5.02	2.41
loose-equal-1-15x10	150	1005.92	986.43	**947.52**	102.00	5.81	3.94
loose-equal-2-15x10	150	1634.89	1563.03	**1522.04**	124.45	6.90	2.62
tight-tard-1-15x2	30	807.45	**790.5**	**790.5**	72.20	2.10	0.00
tight-tard-2-15x2	30	**905.37**	**905.37**	**905.37**	34.30	0.00	0.00
tight-tard-1-15x5	75	1384.44	1389.81	**1359.18**	95.03	1.82	2.20
tight-tard-2-15x5	75	714.88	701.16	**679.45**	78.28	4.96	3.10
tight-tard-1-15x10	150	858.83	813.46	**776.39**	123.63	9.60	4.56
tight-tard-2-15x10	150	1442.82	1304.27	**1232.37**	125.16	14.59	5.51
loose-tard-1-15x2	30	661.74	**654.84**	**654.84**	51.09	1.04	0.00
loose-tard-2-15x2	30	285.16	291.43	**279.71**	34.22	1.91	4.02
loose-tard-1-15x5	75	1404.9	1315.53	**1281.26**	102.37	8.80	2.61
loose-tard-2-15x5	75	**332.85**	386.25	341.03	136.25	-2.46	11.71
loose-tard-1-15x10	150	283.13	282.35	**277.24**	106.32	2.08	1.81
loose-tard-2-15x10	150	679.35	658.9	**587.9**	110.10	13.46	10.78
tight-equal-1-20x2	40	1951.02	1940.3	**1933.72**	111.06	0.89	0.34
tight-equal-2-20x2	40	977.1	**943.7**	951.28	96.56	2.64	-0.80
tight-equal-1-20x5	100	3009.43	**2853.31**	2869.97	180.71	4.63	-0.58
tight-equal-2-20x5	100	7523.08	6915.06	**6643.07**	120.73	11.70	3.93
tight-equal-1-20x10	200	16300.6	10520.4	**10426.5**	133.16	36.04	0.89
tight-equal-2-20x10	200	12101.8	**7201.05**	7303.93	169.12	39.65	-1.43
loose-equal-1-20x2	40	2578.31	2550.53	**2547.68**	75.37	1.19	0.11
loose-equal-2-20x2	40	3194.94	3109.29	**3069.13**	88.11	3.94	1.29
loose-equal-1-20x5	100	8579.99	7646.9	**7496.27**	179.54	12.63	1.97
loose-equal-2-20x5	100	7978.99	7294.5	**7053.66**	182.89	11.60	3.30
loose-equal-1-20x10	200	6891.01	5022.49	**4920.46**	150.24	28.60	2.03
loose-equal-2-20x10	200	3681.51	1816.53	**1626.53**	166.58	55.82	10.46
tight-tard-1-20x2	40	1866.89	1682.72	**1671.87**	132.16	10.45	0.64
tight-tard-2-20x2	40	1466.63	**1452.05**	**1452.05**	104.45	0.99	0.00
tight-tard-1-20x5	100	3776.46	3640	**3612.93**	173.98	4.33	0.74
tight-tard-2-20x5	100	1944.21	1873.8	**1809.02**	172.75	6.95	3.46
tight-tard-1-20x10	200	11748.3	4778.16	**4397.66**	107.83	62.57	7.96
tight-tard-2-20x10	200	6699.4	3270.09	**3198.11**	159.33	52.26	2.20
loose-tard-1-20x2	40	1393.36	1204.92	**1204.43**	109.74	13.56	0.04
loose-tard-2-20x2	40	802.32	**774.22**	782.39	75.06	2.48	-1.06
loose-tard-1-20x5	100	3259.1	2973.23	**2968.86**	164.47	8.91	0.15
loose-tard-2-20x5	100	4128.23	3654.86	**3609.4**	142.93	12.57	1.24
loose-tard-1-20x10	200	8219.27	5100.46	**5039.34**	164.78	38.69	1.20
loose-tard-2-20x10	200	1678.07	1588.7	**1440.72**	167.90	14.14	9.31
Average					77.88	42.71	4.51

allows us to investigate the above mentioned points. In particular the new VNS:

1. includes traditional manipulation techniques i.e. swap and remove-insert;

2. utilizes N_1 (relax-2 hereafter) as relaxation neighbourhood;

3. uses a different relaxation neighbourhood called relax-1 in the shake phase.

It is worth mentioning that in Math_2, the initial sequence for JIT-JSS is generated through procedure GT (discussed in algorithm 4.1) while the solution representation and re-encoding of scheduled sequences are performed as per Sections 4.4.1 and 4.4.5 respectively.

4.5.1 Improvement algorithm

After obtaining a feasible initial sequence for JIT-JSS (through procedure GT (discussed in algorithm 4.1), we improve the sequence by applying the VNS algorithm. Algorithm 4.5 summarizes the proposed Math_2.

In the shake phase of Math_2, we apply the "relax-1" neighbourhood, which relaxes several machine precedence constraints in the given sequence. Specifically, relax-1 randomly selects two machines on which the machine precedence constraints for performing several operations are relaxed. According to our computational results, the relax-1 neighbourhood is computationally inexpensive and is able to produce very good quality solutions as well.

After performing the shake phase, Math_2 proceeds to the local search phase, where the three neighbourhoods of "relax-2", "remove-insert" and "swap" are applied. We discuss these neighbourhood structures in the following sections. Figure 4.6 gives a flowchart of the proposed Math_2.

Relax-1 neighbourhood

As illustrated in Figure 4.6, in the shake phase the "relax-1" neighbourhood is applied. To have a better grasp of this neighbourhood, consider the instance 4×3 discussed in Section 4.3. Recall that Figure 4.1 shows the schedule for the following sequence: $\Pi = (1, 1, 2, \mathbf{3}, \mathbf{2}, \mathbf{4}, 3, 1, 3, 2, 4, 4)$. The second row of the following array specifies the processing machines for the operations of Π.

Algorithm 4.5: Math_2 for JIT-JSS.

Input: An initial sequence Π, generated by GT and its associated objective value $z(\Pi)$; a set of neighbourhood structures $N_\kappa, \kappa = 1, 2, 3$, to be used in the local search.
Output: An improved sequence for JIT-JSS.

while *the stopping condition is not met* **do**
  $\kappa := 1$;
  while $\kappa \leq 3$ **do**
    Shake:
    $\Pi' \leftarrow$ `relax-1(`Π`)`;
    Local Search:
    $\Pi'' \leftarrow$ `LS(`Π'`, `κ`); //See Sub-procedure`
    Move or not:
    if $z(\Pi'') < z(\Pi)$ **then**
      $z(\Pi) := z(\Pi'')$
      $\Pi := \Pi''$;
    else
      $\kappa := \kappa + 1$;
    end
  end
end
return *the best obtained sequence* Π *and its objective value* $z(\Pi)$;

`Sub-procedure LS(`Π'`, `κ`) //the local search in VNS`
$iter := 1$;
while $iter \leq iter\_max$ **do**
  $temp \leftarrow N_\kappa(\Pi')$;
  if $z(temp) < z(\Pi')$ **then**
    $z(\Pi') := z(temp)$;
    $\Pi' := temp$;
  else
    $iter := iter + 1$;
  end
end
return Π';

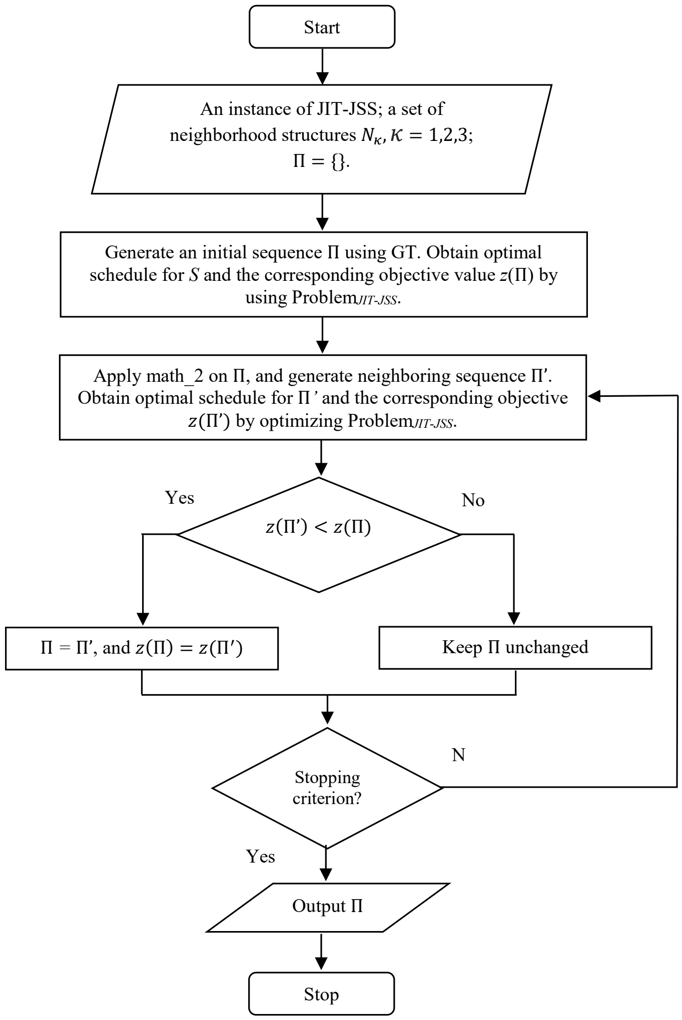

Figure 4.6 : The flowchart of Math_2 for solving JIT-JSS.

$$\begin{pmatrix} Job & \mathbf{1} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{2} & \mathbf{4} & \mathbf{3} & \mathbf{1} & \mathbf{3} & 2 & 4 & 4 \\ Machine & \mathbf{3} & \mathbf{2} & \mathbf{3} & \mathbf{2} & \mathbf{2} & \mathbf{2} & \mathbf{3} & \mathbf{1} & \mathbf{1} & 1 & 1 & 3 \end{pmatrix}.$$

Considering Π again, suppose $RC = 5$, i.e., the 5th position in the sequence, and $RR = 4$. Then the relaxed sub-sequence is $R = (1, 1, 2, 3, 2, 4, 3, 1, 3)$ (or, equivalently, $(O_1^1, O_1^2, O_2^1, O_3^1, O_2^2, O_4^1, O_3^2, O_1^3, O_3^3)$). Scanning through the operations in R, it is clear that they are being performed on machines 1, 2, and 3. However, in the relax-1 neighbourhood, which we use in the shake phase, only two machines are taken into consideration. Assume that machines 1 and 3 are randomly chosen for this reason. This means that the machine precedence constraints for the selected

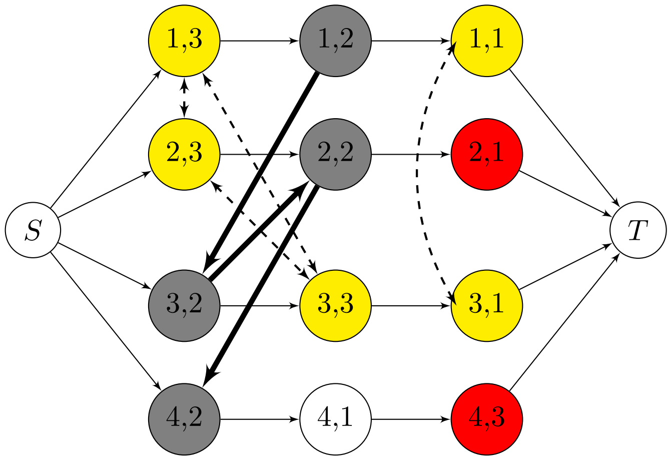

Figure 4.7 : An example of the relax-1 neighbourhood for the instance 4×3. A pair (i, j) represents job i on machine j. As machine 2 has not been selected, its associated operations in R (shown in grey) are performed in the given order by Π. The thick conjunctive arcs impose the order in which grey operations to be performed on machine 2.

operations on these two machines are relaxed. However, because machine 2 is not selected, its associated operations in R, i.e., O_1^2, O_3^1, O_2^2, and O_4^1, are performed in the given order by Π, i.e., we do not change their order. Figure 4.7 shows the relaxed operations on machines 1 and 3 (in yellow) and the non-relaxed operations on machine 2 (in gray). The bold conjunctive arcs specify the order in which the operations in gray are performed on machine 2. Figure 4.8 shows restoring the connectivity between the relaxed and non-relaxed sub-sequences by imposing a few conjunctive arcs ending at the red nodes.

It is worth mentioning that while the relax-1 and relax-2 neighbourhoods are conceptually very similar, their functions in the proposed VNS are quite different. We use the relax-1 neighbourhood to introduce diversification into the search and explore new regions of the solution space. We apply relax-2, however, in the local search to intensify the search and obtain superior solutions through exploiting the incumbent solution.

Remove-Insert neighbourhood

Given a sequence, a randomly selected operation is removed from its original position and inserted into another randomly selected position. Figure 4.9 shows an

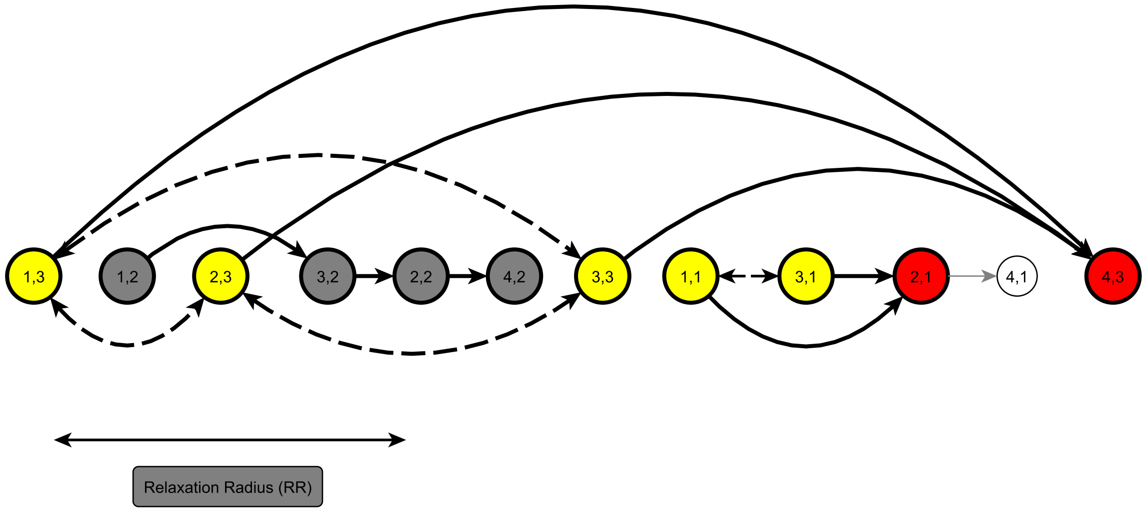

Figure 4.8 : An example of the relax-1 neighbourhood for the instance 4×3. The conjunctive arcs ending at red nodes guarantee the connectivity between relaxed and non-relaxed operations. A pair (i, j) represents job i on machine j.

example of the remove-insert neighbourhood for the instance 4×3. N_2 in Algorithm 4.5 denotes this neighbourhood.

a: 1 1 **2** 3 2 4 3 1 3 2 4 4

b: 1 1 3 2 4 **2** 3 1 3 2 4 4

Figure 4.9 : An example of the remove-insert neighbourhood in the instance 4×3: (a) before the remove-insert operation and (b) after the remove-insert operation.

Swap neighbourhood

This neighbourhood swaps the positions of two randomly chosen operations in a sequence. Figure 4.10 shows an example of the swap neighbourhood for the instance 4×3. In Algorithm 4.5, N_3 denotes this neighbourhood.

4.5.2 Computational results

To assess the performance of Math_2 (see Algorithm 4.5), we test it on the 72 benchmark instances introduced in Section 4.4.6. We first tune the value of parameters of Math_2 and examine the impact of neighbourhoods on solution quality.

a: 1 1 **2** 3 2 4 3 1 **3** 2 4 4

b: 1 1 **3** 3 2 4 3 1 **2** 2 4 4

Figure 4.10 : Swapping two operations in the instance 4×3; (a) before the swap and (b) after the swap.

Then we adopt the best setting for Math_2. We compare the performance of Math_2 and the EA and VNS algorithms of Dos Santos et al. (2010) and Wang and Li (2014), which we re-implement on the same machine as of our Math_2. We also solve Problem$_{\mathcal{JIT}-\mathcal{JSS}}$ using CPLEX Concert Technology version 12.8.0 (ILOG, 2017). Next, we tune the value of parameters and analyse the impact of neighbourhoods, and report the computational results.

Parameter tuning

We run two experiments to tune the values of the parameters of Math_2 and to examine the impact of neighbourhoods on solution quality. The first experiment assesses the best neighbourhood(s) for shake, the order of the neighbourhoods in the local search, and the effects of the values of two parameters of RC and RR on solution quality. The second experiment evaluates the contribution of the relax-2 neighbourhood to solution quality.

Experiment 1

We investigate the impacts of the following on solution quality of Math_2:

- Neighbourhood(s) in the shake phase: As discussed in Section 4.5.1, Math_2 differs from the classic one in that it uses relax-1 in the shake phase. To verify the superiority of the results delivered by our proposed shake, we test two variants: (1) relax-1 in shake (denoted as shake 1) and relax-2, swap, and remove-insert in shake (denoted as shake 2).

- The order of the neighbourhoods in the local search phase: The order of the neighbourhoods may affect solution quality. Since we include the three neighbourhoods of swap, remove-insert, and relax-2 in Math_2, there are six distinct orders for implementing these neighbourhoods. We test the performance of Math_2 under all these six orders of neighbourhoods.

- Parameter RC: When choosing the value of RC, one can randomly and uniformly pick an operation from a given sequence. This, however, might not always lead to quality solutions. Therefore, we test three simple non-uniform distributions (see Figures 4.11 to 4.13) from which the value of RC in each iteration of `Sub-procedure LS` in Algorithm 4.5 is chosen. For example, the distribution function shown in Figure 4.11 breaks down a given sequence into four distinct sections (shown in various patterns) with different probabilities of occurrence, i.e., 0.20, 0.20, 0.25, and 0.35. As Figure 4.11 illustrates, there is a 20% chance that the value of RC is randomly chosen from either the first or the last 5% of the operations in the given sequence.

- Parameter RR: It is reasonable to choose the value of RR proportionate to the size of the instance. We test two sets of values for RR detailed in Table 4.10, denoted as RR 1 and RR 2. For example, under RR 1, the value of the parameter RR is equal to $\frac{n \times m}{3}$ for instances with ten jobs, while under RR 2, the value is equal to $\frac{n \times m}{6}$.

We select 12 instances, out of 72, ranging from 10 to 20 jobs and 5 to 10 machines and run Math_2 for five times. We exclude instances with two machines. Testing two possibilities for shake, six distinct neighbourhood orders for local search, two sets of the value of RR, and three distributions (to choose the value of RC, and also to guide the swap and remove-insert neighbourhoods, i.e., to select a pair of positions for the operations in swap, and the removal and insertion positions in remove-insert) result in 4,320 tested combinations ($12 \times 5 \times 2 \times 6 \times 2 \times 3$). For the computational results of this section, we set the values of the parameter $iter\_max$ for relax-2, swap, and remove-insert to 10, 20, and 20, respectively. We also set the computational time limit of CPLEX (within Math_2) to one second for both the relax-1 and relax-2 neighbourhoods. From our initial experiments, we observe that setting the CPLEX time limit to greater values does not considerably improve the quality of the solutions but it leads to an increase in the computational time for Math_2. To terminate Math_2, we use two criteria (whichever occurs earlier) of (1) the maximum number of iterations that we set to $\lfloor \frac{n \times m}{4} \rfloor$, where $\lfloor x \rfloor$ is the greatest integer less than or equal to x, and (2) the maximum number of iterations without an improvement that we set to three, meaning that Math_2 terminates if it cannot find an improved solution after three consecutive iterations. We summarize the outcomes in Table 4.9.

In Table 4.9, we calculate "Avg (best)" as $\frac{\sum_{k=1}^{12}(\frac{z_k - z_k^*}{z_k^*} \times 100)}{12}$, where z_k^* is the best known solution in the literature for the instance k and z_k is the best solution delivered by Math_2 in five runs for the same instance. Therefore, a negative value implies an improved solution obtained by Math_2. We also compute the average computational time (in seconds) over all the 12 instances that we solve. As the table shows, the largest average improvement is equal to -4.31 (highlighted in the table) and is obtained via shake 1 that includes relax-1; the neighbourhood order of relax-2, remove-insert, and swap; distribution 1 for selecting RC; and RR 1 for selecting the parameter RR. We therefore use this setting to solve the remaining instances, which will be discussed in Section 4.5.2. It should be noted that the RR 2 setting for selecting the value of parameter RR leads to a significantly faster Math_2, though its solution quality is inferior to that of RR 1.

Table 4.8 : The best known lower bounds.

Instance	LBR	LBP	LBRP	Best LB	Gap (%) CPLEX	Best of literature	Math_1
tight-equal-1-10x2	434	433	461.96	461.96	0	0	0
tight-equal-2-10x2	357	418	448.32	448.32	0	0	0
tight-equal-1-10x5	660	536	688.674	688.674	4.93	0.06	0.06
tight-equal-2-10x5	592	612	763.24	763.24	2.09	0	0
tight-equal-1-10x10	1126	812	1184.395	1184.395	7.75	7.86	7.75
tight-equal-2-10x10	1535	819	1659.251	1659.251	13.74	13.2	12.52
loose-equal-1-10x2	218	219	224.84	224.84	0	0	0
loose-equal-2-10x2	313	298	317.542	317.542	2.17	0.58	2.17
loose-equal-1-10x5	1263	1205	1680.105	1680.105	2.47	3.57	5.18
loose-equal-2-10x5	878	780	945.206	945.206	2.83	2.38	2.83
loose-equal-1-10x10	331	294	355.401	355.401	2.53	2.53	1.5
loose-equal-2-10x10	246	211	249.85	249.85	0	0	0
tight-tard-1-10x2	168	174	179.25	179.25	0.12	0.12	0.12
tight-tard-2-10x2	143	138	145.37	145.37	0	0	0
tight-tard-1-10x5	361	322	371.174	371.174	3.98	4.34	4.34
tight-tard-2-10x5	420	461	610.905	610.905	2.71	3.56	4.1
tight-tard-1-10x10	574	408	599.612	599.612	11.43	14.68	14.68
tight-tard-2-10x10	666	469	717.61	717.61	9.18	8.6	8.39
loose-tard-1-10x2	413	408	416.44	416.44	0	0	0
loose-tard-2-10x2	135	137	137.94	137.94	0	0	0
loose-tard-1-10x5	168	159	175.08	175.08	0	0	0
loose-tard-2-10x5	355	313	467.437	467.437	10.89	6.95	7.9
loose-tard-1-10x10	356	314	368.823	368.823	1.87	4.08	1.87
loose-tard-2-10x10	138	119	144.94	144.94	0	0	0
tight-equal-1-15x2	2902	3316	-	3316	2.54	0.86	0.86
tight-equal-2-15x2	1253	1449	-	1449	3.31	2.12	2.12
tight-equal-1-15x5	964	1052	-	1052	27.5	29.57	25.35
tight-equal-2-15x5	1630	1992	-	1992	34.08	35.2	45.46
tight-equal-1-15x10	4389	3662	-	4389	74.66	56.05	58.35
tight-equal-2-15x10	3539	2564	-	3539	51.23	51.6	34.23
loose-equal-1-15x2	1014	1032	-	1032	3.36	0.94	0.9
loose-equal-2-15x2	490	472	-	490	6.58	1.63	3.09
loose-equal-1-15x5	2449	2763	-	2763	18.71	18.27	16.09
loose-equal-2-15x5	2818	2773	-	2818	22.41	19.13	16.26
loose-equal-1-15x10	758	628	-	758	32.71	30.14	25
loose-equal-2-15x10	1242	979	-	1242	31.63	25.85	22.55
tight-tard-1-15x2	720	786	-	786	2.73	0.57	0.57
tight-tard-2-15x2	843	886	-	886	2.19	2.19	2.19
tight-tard-1-15x5	1008	1014	-	1014	36.53	37.06	34.04
tight-tard-2-15x5	626	547	-	626	14.2	12.01	8.54
tight-tard-1-15x10	649	467	-	649	32.33	25.34	19.63
tight-tard-2-15x10	955	761	-	955	51.08	36.57	29.04
loose-tard-1-15x2	616	650	-	650	1.81	0.74	0.74
loose-tard-2-15x2	278	277	-	278	2.58	4.83	0.62
loose-tard-1-15x5	1098	1005	-	1098	27.95	19.81	16.69
loose-tard-2-15x5	314	313	-	314	6	23.01	8.61
loose-tard-1-15x10	258	233	-	258	9.74	9.44	7.46
loose-tard-2-15x10	476	454	-	476	42.72	38.42	23.51
tight-equal-1-20x2	1747	1901	-	1901	2.63	2.07	1.72
tight-equal-2-20x2	858	912	-	912	7.14	3.48	4.31
tight-equal-1-20x5	2506	2244	-	2506	20.09	13.86	14.52
tight-equal-2-20x5	4923	5817	-	5817	29.33	18.88	14.2
tight-equal-1-20x10	6656	6708	-	6708	143	56.83	55.43
tight-equal-2-20x10	5705	-	-	5705	112.13	26.22	28.03
loose-equal-1-20x2	2388	2546	-	2546	1.27	0.18	0.07
loose-equal-2-20x2	2970	3013	-	3013	6.04	3.2	1.86
loose-equal-1-20x5	5571	6697	-	6697	28.12	14.18	11.93
loose-equal-2-20x5	5496	6017	-	6017	32.61	21.23	17.23
loose-equal-1-20x10	3538	3099	-	3538	94.77	41.96	39.07
loose-equal-2-20x10	1344	1150	-	1344	173.92	35.16	21.02
tight-tard-1-20x2	1515	-	-	1515	23.23	11.07	10.35
tight-tard-2-20x2	1375	1327	-	1375	6.66	5.6	5.6
tight-tard-1-20x5	2507	3244	-	3244	16.41	12.21	11.37
tight-tard-2-20x5	1633	-	-	1633	19.06	14.75	10.78
tight-tard-1-20x10	3003	2764	-	3003	291.22	59.11	46.44
tight-tard-2-20x10	2740	-	-	2740	144.5	19.35	16.72
loose-tard-1-20x2	1194	1189	-	1194	16.7	0.91	0.87
loose-tard-2-20x2	734	735	-	735	9.16	5.34	6.45
loose-tard-1-20x5	2177	2524	-	2524	29.12	17.8	17.63
loose-tard-2-20x5	2643	3060	-	3060	34.91	19.44	17.95
loose-tard-1-20x10	2462	2436	-	2462	233.85	107.17	104.68
loose-tard-2-20x10	1226	-	-	1226	36.87	29.58	17.51
Average					29.58	15.19	13.21

Table 4.9 : Summary of the computational results in experiment 1.

	Order*	Distribution 1				Distribution 2				Distribution 3			
		RR 1		RR 2		RR 1		RR 2		RR 1		RR 2	
		Avg (Best)	Avg (Time)	Avg (Best)	Avg (Time)	Avg (Best)	Avg (Time)	Avg (Best)	Avg (Time)	Avg (Best)	Avg (Time)	Avg (Best)	Avg (Time)
Shake 1	R2-S-RI	-4.28	152.05	-2.67	64.06	-4.07	165.86	-2.28	65.14	-4.23	169.61	-2.80	72.97
	R2-RI-S	**-4.31**	170.62	-2.78	71.59	-3.87	161.09	-2.05	67.10	-3.75	153.64	-1.91	69.60
	S-R2-RI	-3.97	151.56	-2.72	63.68	-3.63	156.35	-2.53	65.66	-3.97	146.39	-2.17	67.71
	S-RI-R2	-3.79	157.29	-2.26	62.65	-3.65	144.33	-2.51	63.98	-3.61	153.97	-1.94	69.81
	RI-S-R2	-3.63	155.66	-2.93	62.53	-3.35	155.81	-2.66	65.81	-3.94	157.94	-2.51	73.97
	RI-R2-S	-3.79	169.51	-2.64	73.07	-3.83	149.33	-3.04	59.45	-3.85	170.14	-2.87	73.24
Shake 2	R2-S-RI	-3.66	162.55	-2.09	83.04	-3.76	156.03	-2.61	74.32	-3.65	154.13	-2.64	86.62
	R2-RI-S	-3.20	136.73	-1.98	73.31	-3.27	136.86	-2.32	77.86	-3.68	150.48	-2.19	74.64
	S-R2-RI	-2.96	158.44	-2.68	87.79	-3.84	156.21	-2.43	75.47	-3.36	135.54	-1.99	86.67
	S-RI-R2	-3.22	142.20	-2.08	67.86	-2.90	137.27	-2.31	80.34	-2.78	143.98	-2.85	84.34
	RI-S-R2	-3.23	145.96	-3.32	78.66	-3.40	137.42	-3.20	68.12	-3.72	149.68	-2.28	77.75
	RI-R2-S	-3.56	146.41	-2.75	73.06	-3.59	148.82	-2.63	76.93	-3.31	144.74	-2.07	87.47

*R2: relax-2, RI: remove-insert, S: swap.

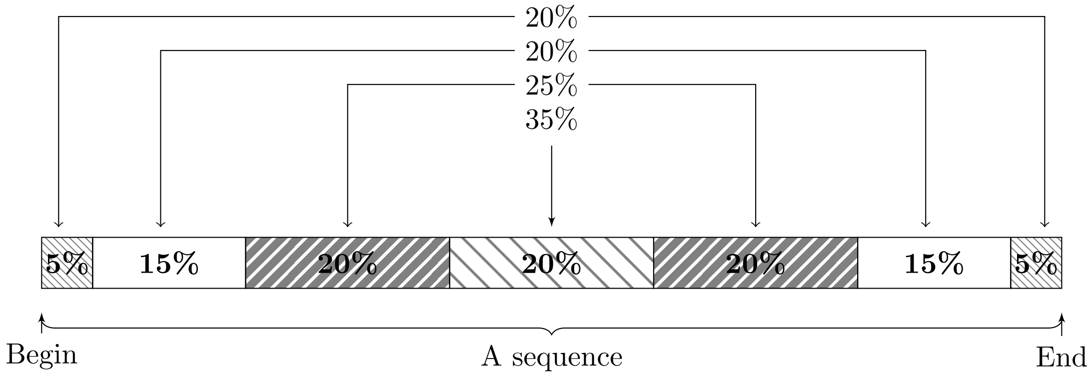

Figure 4.11 : Distribution function 1 to choose the value of parameter RC from for relax-1 and relax-2 and to guide swap and remove-insert neighbourhoods, i.e., to select a pair of positions for the operations in swap and the removal and insertion positions in remove-insert.

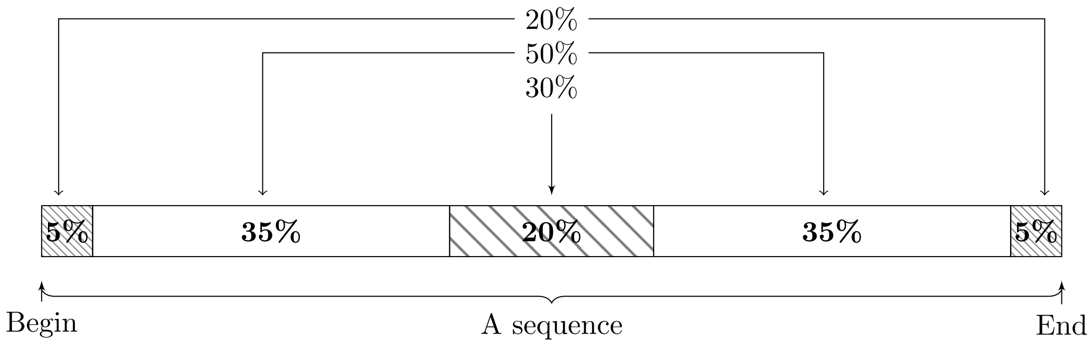

Figure 4.12 : Distribution function 2 to choose the value of parameter RC from for relax-1 and relax-2 and to guide swap and remove-insert neighbourhoods.

Experiment 2

Since the relax-2 neighbourhood has distinct characteristics from those of the traditional neighbourhoods such as swap and remove-insert, we investigate its impact on solution quality. We conduct experiment 2, in which we exclude the relax-2 neighbourhood from the local search so that the order of the implemented neighbourhoods included remove-insert and swap. We do not change the remaining settings, i.e., we use relax-1 in shake, distribution 1, and the setting RR 1 for selecting the values of the parameters RC and RR (see Section 4.5.2). We solve the same 12 instances discussed in Section 4.5.2 and summarize the outcomes in Table 4.11.

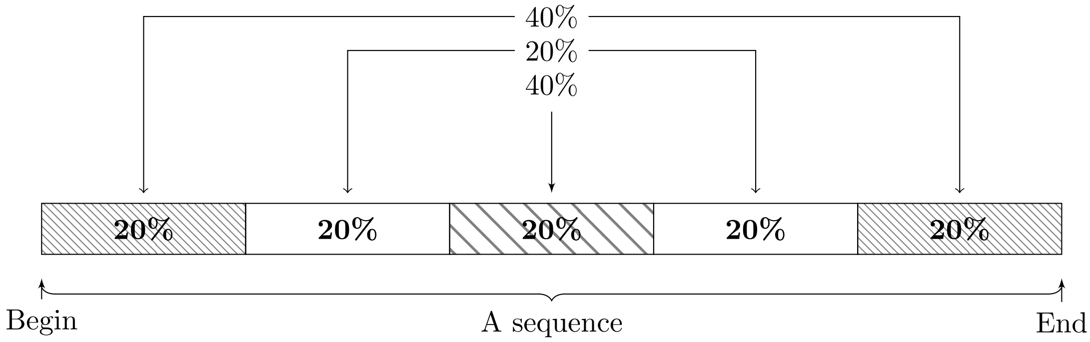

Figure 4.13 : Distribution function 3 to choose the value of parameter RC from for relax-1 and relax-2 and to guide swap and remove-insert neighbourhoods.

Table 4.10 : The values of the parameter RR in experiment 1.

RR 1	RR 2
$\begin{cases} \dfrac{n \times m}{3}, & \text{if} \quad n = 10 \\ \dfrac{n \times m}{4}, & \text{if} \quad n = 15 \\ \dfrac{n \times m}{5}, & \text{if} \quad n = 20 \end{cases}$	$\begin{cases} \dfrac{n \times m}{6}, & \text{if} \quad n = 10 \\ \dfrac{n \times m}{8}, & \text{if} \quad n = 15 \\ \dfrac{n \times m}{10}, & \text{if} \quad n = 20 \end{cases}$

Similar to Table 4.9, we compute "Avg (best)" as $\frac{\sum_{k=1}^{12}(\frac{z_k - z_k^*}{z_k^*} \times 100)}{12}$, where z_k^* is the best known solution in the literature for the instance k and z_k is the best solution obtained by Math_2 in five runs for the same instance. Moreover, we calculate the average computational time, i.e., Avg (Time), over all the 12 instances that we solve. According to the results, it is clear that the relax-2 neighbourhood is very effective in obtaining high quality solutions. We note that while, on average, the removal of the relax-2 neighbourhood leads to slightly better solutions than the best known

Table 4.11 : Impact of the relax-2 neighbourhood on solution quality.

Order of neighbourhoods	Avg (best)	Avg (time)
R2-RI-S*	-4.31	170.62
RI-S	-0.32	16.34
*R2: relax-2, RI: remove-insert, S: swap.		

ones in the literature, i.e., 0.32%, its inclusion significantly enhances the solutions by nearly 4% ($4.31 - 0.32 = 3.99$).

Comparison against state-of-the-art solution methods

We compare the performance of Math_2 with that of solver CPLEX, and the EA and VNS algorithms of Dos Santos et al. (2010), and Wang and Li (2014), which we re-implement on the same computer as that for implementing Math_2. To make fair comparisons, we initialize CPLEX and VNS with the same initial solution, i.e., we use the initial solutions returned by GT to initialize both CPLEX and Math_2. We opted to re-implement only EA and VNS, rather than all the available methods for solving the same instances (Laborie and Godard, 2007; Monette et al., 2009; Yang et al., 2012a), since the two chosen algorithms are conceptually similar to Math_2 algorithm in that the algorithms operate by iteratively solving the mixed integer program, and 87.5% of the best known solutions in the literature are due to the two algorithms.

It should be noted that while we make a meticulous effort to re-implement EA and VNS as close as possible to their original implementation reported in Dos Santos et al. (2010), and Wang and Li (2014), our re-implementation may be slightly different from the operations of the algorithms as explained in the original studies. This is because the original works did not fully discuss all the components of those algorithms. In addition, we provide an additional computational time allowance for those algorithms in order to ensure a fair comparison. As such, for each value of n, we set the run times of EA and VNS to the longest running time of Math_2 over all the instances with the same value of n and five runs of the algorithm. Therefore, we ran EA and VNS for 96, 308 and 603 seconds for instances with 10, 15 and 20 jobs.

Table 4.12 summarizes the outcomes of Math_2, and those obtained by CPLEX and our re-implementation of EA of Dos Santos et al. (2010), and VNS of Wang and Li (2014). The results show that Math_2 performs well against all the three methods of CPLEX, EA, and VNS. Specifically, for 61 instances, out of 72, Math_2 delivers solutions that are either of same or superior quality to the best solutions produced by CPLEX, EA, and VNS. This is significant and is equal to nearly 85% of instances. Interestingly, even the average solution of Math_2 competes well with the best solutions produced by CPLEX, EA, and VNS for almost 43% of the instances. The average solution of Math_2 outperforms the best solution of CPLEX and the average solutions of EA and VNS for 45 instances, i.e., for 62.5% of the instances. To complement the analysis of the results in Table 4.12 and to validate the statistical significance of the superior performance of Math_2 over the other three methods, we conduct Wilcoxon signed rank tests at the significance level of 5%, i.e., $\alpha = 0.05$, on pairwise comparisons of the average performance between Math_2 versus CPLEX, EA of Dos Santos et al. (2010), and VNS of Wang and Li (2014) for instances with 10, 15, and 20 jobs. The null hypothesis is that there is no significant difference in the average objective function value between two compared methods. We use the alternative hypothesis to test whether the average objective function value of Math_2 is less than that of the other method. Table 4.13 reports the p-values of the tests. It is noted that when the p-value is less than the value of the significance level, it indicates that there is a significant difference between the two compared methods. Table 4.13 shows that all the null hypotheses are rejected, except for the test of Math_2 versus CPLEX for $n = 10$, which is expected as CPLEX produces quality solutions at the expense of long computing times (up to 1,800 seconds). The rejections of the tests of Math_2 versus EA and the VNS further demonstrate that Math_2 armed with novel relaxation neighbourhoods is capable of obtaining high quality solutions that are much superior to those obtained by previous top performing methods.

Table 4.12 : Summary of the computational results for Math_2.

Criterion	Number of jobs (n)			Total
	10	15	20	
Number of instances for which the best solution delivered by Math_2 is equal to or better than the best among CPLEX, EA and VNS	23	18	20	61
Number of instances for which the average solution delivered by Math_2 is equal to or better than the best among CPLEX, EA and VNS	11	11	9	31
Number of instances for which the average solution delivered by Math_2 is equal to or better than the average among CPLEX, EA and VNS	14	14	17	45

Tables 4.14 to 4.16 provide the details of the computational results, where each

Table 4.13 : The p-values of the Wilcoxon signed rank tests.

Test	$n = 10$	$n = 15$	$n = 20$
Math_2 versus CPLEX	0.843	0.022	0.000
Math_2 versus EA (Dos Santos et al. (2010))	0.000	0.002	0.001
Math_2 versus VNS (Wang and Li (2014))	0.000	0.000	0.000

table presents the results associated with each value of n. The tables compare the best results obtained by Math_2 versus the best results obtained by CPLEX, and those produced by our re-implementation of EA of Dos Santos et al. (2010), and VNS of Wang and Li (2014). In Tables 4.14 to 4.16, the first column shows the names of the instances. The second and third columns denote the outcomes of CPLEX obtained under the time limit of 1,800 seconds, including the best obtained objective function value ("z") and the computational time in seconds. Columns four to six show the outcomes of EA, namely the best objective function value ("z_{best}"), the average objective function value over five runs ("z_{avg}"), and the pre-set computational time in seconds ("Time (s.)"), and columns seven to nine denote those of Wang and Li (2014)'s VNS. The last four columns present the outcomes of Math_2. Within those, the column "z_0" shows the objective function value of the initial solution, and the remaining three columns report the best and average objective function values, and the average of the computational time (in seconds) over the five runs. We run EA, Wang and Li (2014)'s VNS, and Math_2 algorithms for five runs in order to mitigate fluctuations in the performance of the solver CPLEX. Since we apply CPLEX heuristically by setting a time limit and we use the default values for its parameters, except the time limit and number of processors, CPLEX may not deliver the same solution for an instance over different runs of an algorithm. In the tables, the numbers in bold indicate the best solutions among four methods and the numbers in the parentheses show the numbers of occurrence of the best solution for the instances.

According to Table 4.14, Math_2 delivers the largest number of best solutions among the four tested methods, i.e., for 23 instances. CPLEX produces 13 best solutions. Neither of EA and VNS obtains the best solutions for more than 54% of the instances. For the majority of instances, our algorithm obtains the best solutions in multiple runs of the algorithm, indicating its reliable performance. Math_2 is also the fastest method. Recall that we use the longest running time of Math_2 over all the instances with the same value of n and five runs of the algorithm as the computational time limit for EA and VNS. Against such a background, while on average the computational time of Math_2 is around one minute, EA and VNS have average computational times of more than 1.5 minutes. It should be noted that Math_2 significantly improves the initial solutions reported in column "z_0", implying that its effectiveness may not only be attributed to the quality of the initial solutions.

Table 4.15 reports the outcomes of the four methods for solving the instances with 15 jobs. As the numbers in bold show, Math_2 obtains the largest number of best solutions, achieving that for 18 instances. This is equal to 75% of the instances, whereby none of the methods of CPLEX, EA, and VNS can obtain the best solutions for more than 42% of the instances. Our algorithm is also significantly faster. The longest running time of our algorithm for all the 24 instances and five runs is around 308 seconds, which we set as the computational time limits for both EA and VNS. Nevertheless, on average, Math_2 is able to obtain the best solutions in less time.

According to Table 4.16, which shows the outcomes of CPLEX, EA, VNS, and Math_2 for instances with 20 jobs, our algorithm outperforms all the three methods for comparison. For example, within 30 minutes of running, CPLEX is unable to report a single best solution, and EA and VNS report only six and one best solutions, respectively. These are for about 25% and 4% of the instances. Our algorithm, however, produces the best solutions for 20 instances, achieving that in less than five minutes on average, whereas EA and VNS are let run for ten minutes. The performance of Math_2 is further acknowledged by Figures 4.14 and 4.15, which illustrate the progress of CPLEX, EA, VNS, and Math_2 in reporting the best solution for two instances of "`tight-equal-1-20×10`" and "`loose-tard-2-20×10`". As seen from the figures, the proposed VNS manages to find much better objective values in shorter computational times.

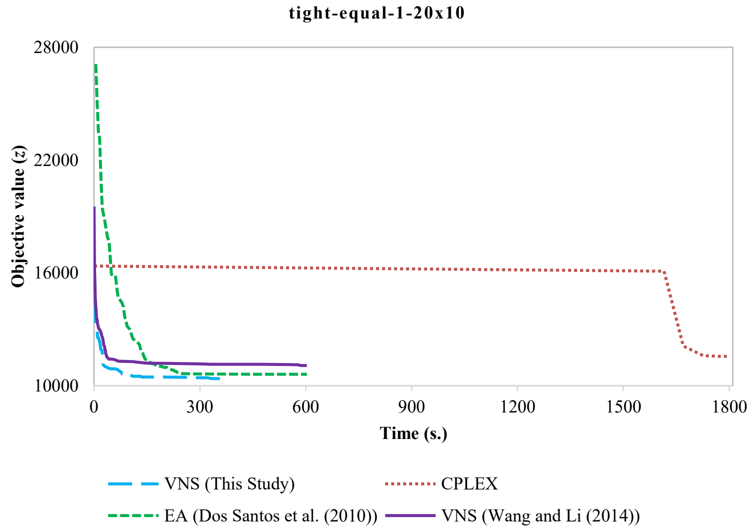

Figure 4.14 : Changes in the objective value z under CPLEX, EA, VNS, and Math_2 for "`tight-equal-1-20×10`".

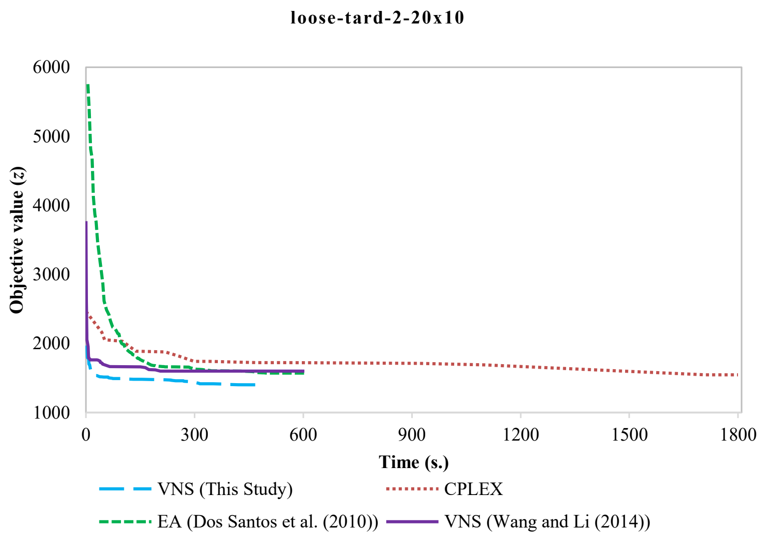

Figure 4.15 : Changes in the objective value z under CPLEX, EA, VNS, and Math_2 for "`loose-tard-2-20×10`".

Table 4.14 : Computational results for instances with 10 jobs (for experiments CPLEX 12.8.0 was used).

Instance	CPLEX		EA (Dos Santos et al. (2010))			VNS (Wang and Li (2014))			Math_2			
	z	Time (s.)	z_{best}	z_{avg}	Time (s.)	z_{best}	z_{avg}	Time (s.)	z_0	z_{best}	z_{avg}	Time (s.)
tight-equal-1-10x2	**461.96**	1800	**461.96 (5)**	461.96	96	**461.96 (4)**	466.22	96	860.57	**461.96 (5)**	461.96	38.38
tight-equal-2-10x2	**448.32**	412.31	**448.32 (5)**	448.32	96	**448.32 (5)**	448.32	96	592.69	**448.32 (5)**	448.32	33.47
tight-equal-1-10x5	**689.11**	1800	**689.11 (1)**	735.30	96	**689.11 (5)**	689.11	96	1141.07	**689.11 (4)**	695.81	41.16
tight-equal-2-10x5	**763.24**	1800	770.49 (1)	782.95	96	764.03 (1)	809.80	96	1059.49	**763.24 (5)**	763.24	46.76
tight-equal-1-10x10	1281.66	1800	**1276.23 (1)**	1297.51	96	1277.44 (1)	1312.24	96	2786.21	**1276.23 (3)**	1280.09	68.14
tight-equal-2-10x10	1885.25	1800	1885.25 (1)	1944.50	96	1871.23 (1)	1877.24	96	3238.05	**1866.92 (3)**	1875.33	73.80
loose-equal-1-10x2	**224.84**	1598.6	**224.84 (5)**	224.84	96	**224.84 (5)**	224.84	96	345.3	**224.84 (5)**	224.84	36.97
loose-equal-2-10x2	**319.37**	1800	324.43 (5)	324.43	96	**319.37 (1)**	327.50	96	642.8	**319.37 (3)**	321.39	35.36
loose-equal-1-10x5	1738.05	1800	1733.38 (1)	1771.63	96	1738.05 (2)	1752.69	96	2459.63	**1719.63 (2)**	1739.46	51.30
loose-equal-2-10x5	971.96	1800	968.89 (1)	1001.45	96	**967.73 (2)**	981.14	96	1534.52	**967.73 (3)**	970.29	49.74
loose-equal-1-10x10	**360.74**	1800	366.77 (1)	375.88	96	365.81 (1)	368.18	96	724.95	364.39 (5)	364.39	51.99
loose-equal-2-10x10	251.86	1800	**249.85 (1)**	254.78	96	252.99 (1)	256.05	96	419.72	**249.85 (5)**	249.85	69.81
tight-tard-1-10x2	**179.46**	1260	**179.46 (1)**	179.84	96	**179.46 (5)**	179.46	96	296.55	**179.46 (5)**	179.46	19.72
tight-tard-2-10x2	**145.37**	1800	**145.37 (5)**	145.37	96	**145.37 (2)**	145.54	96	279.52	**145.37 (3)**	148.33	44.94
tight-tard-1-10x5	**379.19**	1800	380.82 (1)	387.98	96	380 (1)	389.81	96	533.08	**379.19 (1)**	387.09	58.96
tight-tard-2-10x5	635.93	1800	**627.45 (2)**	633.80	96	628.68 (1)	651.52	96	893.52	**627.45 (5)**	627.45	52.23
tight-tard-1-10x10	687.65	1800	717.85 (1)	760.61	96	746.03 (1)	759.14	96	1254.37	**668.14 (2)**	694.74	67.19
tight-tard-2-10x10	779.17	1800	780.82 (1)	795.78	96	785.79 (1)	803.61	96	1416.39	**777.85 (1)**	781.93	60.98
loose-tard-1-10x2	**416.44**	255.09	**416.44 (5)**	416.44	96	**416.44 (4)**	416.51	96	506.44	**416.44 (5)**	416.44	9.45
loose-tard-2-10x2	**137.94**	215.33	**137.94 (5)**	137.94	96	**137.94 (4)**	137.95	96	248.63	**137.94 (5)**	137.94	13.76
loose-tard-1-10x5	176.83	1800	**175.08 (2)**	177.92	96	**175.08 (1)**	177.56	96	275.14	**175.08 (5)**	175.08	45.23
loose-tard-2-10x5	492.05	1800	525.88 (1)	529.25	96	499.93 (1)	511.88	96	689.74	**485.06 (1)**	494.73	51.90
loose-tard-1-10x10	375.71	1800	383.84 (1)	387.84	96	380.26 (1)	389.82	96	592.82	**374.2 (3)**	376.00	54.97
loose-tard-2-10x10	**144.94**	1800	**144.94 (3)**	148.49	96	**144.99 (1)**	147.03	96	263.52	**144.94 (5)**	144.94	52.73

Numbers in bold indicate the best solution among four methods.
Numbers in the parentheses indicate the number of occurrence of the best solution over five runs for the instance.

Table 4.15 : Computational results for instances with 15 jobs (for experiments CPLEX 12.8.0 was used).

Instance	CPLEX		EA (Dos Santos et al. (2010))			VNS (Wang and Li (2014))			Math_2			
	z	Time (s.)	z_{best}	z_{avg}	Time (s.)	z_{best}	z_{avg}	Time (s.)	z_0	z_{best}	z_{avg}	Time (s.)
tight-equal-1-15x2	3366.66	1800	**3344.54 (2)**	3351.96	308	**3344.54 (1)**	3349.22	308	5138.98	**3344.54 (5)**	3344.54	60.14
tight-equal-2-15x2	**1479.76**	1800	**1479.76 (3)**	1480.77	308	**1479.76 (2)**	1481.79	308	1773.94	**1479.76 (5)**	1479.76	62.18
tight-equal-1-15x5	1412.71	1800	1369.06 (1)	1382.62	308	1345.21 (1)	1395.29	308	1974.39	**1342.3 (1)**	1358.95	111.03
tight-equal-2-15x5	2782.17	1800	**2630.06 (1)**	2695.47	308	2709.13 (1)	2738.18	308	4131.59	2641.28 (2)	2644.80	117.32
tight-equal-1-15x10	7167.59	1800	7380.76 (1)	7669.13	308	8120.43 (1)	8261.36	308	10123.2	**6820.87 (1)**	6937.49	119.53
tight-equal-2-15x10	5407.42	1800	4893.44 (1)	5159.20	308	4966.34 (1)	5158.85	308	8483.69	**4760.48 (1)**	4904.92	162.27
loose-equal-1-15x2	**1041.33**	1800	**1041.33 (3)**	1045.32	308	**1041.33 (2)**	1041.98	308	1470.05	**1041.33 (4)**	1044.61	56.73
loose-equal-2-15x2	**497.97**	1800	**497.97 (2)**	505.66	308	514.72 (1)	532.61	308	946.96	512.54 (1)	516.95	68.90
loose-equal-1-15x5	3457.57	1800	**3220.16 (1)**	3305.47	308	3321.54 (1)	3431.96	308	4795.27	3272.46 (1)	3337.26	149.59
loose-equal-2-15x5	3437.32	1800	3328.86 (1)	3357.53	308	3310.27 (1)	3352.09	308	4710.14	**3260.81 (1)**	3291.32	134.43
loose-equal-1-15x10	**875.74**	1800	1008.28 (1)	1114.67	308	977.43 (1)	998.03	308	1633.74	906.49 (1)	928.39	167.90
loose-equal-2-15x10	1587.09	1800	1720.18 (1)	1774.68	308	1584.44 (1)	1628.08	308	3134.21	**1487.33 (1)**	1540.40	141.20
tight-tard-1-15x2	806.92	1800	**790.5 (2)**	790.67	308	790.66 (1)	803.35	308	1131.25	**790.5 (3)**	796.72	74.91
tight-tard-2-15x2	**905.37**	1800	**905.37 (5)**	905.37	308	**905.37 (1)**	906.35	308	1159.59	**905.37 (5)**	905.37	84.95
tight-tard-1-15x5	1371.29	1800	1371.37 (1)	1380.55	308	1426.97 (1)	1476.65	308	1938.67	**1359.18 (1)**	1371.13	137.08
tight-tard-2-15x5	721.46	1800	691.1 (1)	696.04	308	725.72 (1)	753.55	308	1030.78	**681.27 (1)**	690.75	175.06
tight-tard-1-15x10	815.03	1800	800.87 (1)	824.74	308	832.08 (1)	861.04	308	1228.14	**767.26 (1)**	795.59	158.15
tight-tard-2-15x10	1267.53	1800	1310.04 (1)	1360.76	308	1452.9 (1)	1700.07	308	2620.94	**1224.82 (1)**	1259.04	166.27
loose-tard-1-15x2	**654.84**	1800	**654.84 (5)**	654.84	308	**654.84 (1)**	655.24	308	935.74	**654.84 (5)**	654.84	72.83
loose-tard-2-15x2	293.17	1800	**279.71 (2)**	287.38	308	297.54 (2)	299.02	308	514.81	285.16 (1)	289.74	74.07
loose-tard-1-15x5	1281.87	1800	1320.93 (1)	1334.13	308	1315.47 (1)	1338.49	308	2238.12	**1281 (1)**	1302.70	133.23
loose-tard-2-15x5	335.55	1800	329.47 (1)	340.09	308	335.55 (1)	347.24	308	552.18	**328.26 (1)**	341.45	166.52
loose-tard-1-15x10	**277**	1800	297.19 (1)	313.86	308	296.6 (1)	301.55	308	415.06	**277 (2)**	277.67	102.40
loose-tard-2-15x10	**597.93**	1800	713.97 (1)	779.17	308	664.13 (1)	692.36	308	1344.65	601.24 (1)	646.20	176.80

Numbers in bold indicate the best solution among four methods.
Numbers in the parentheses indicate the number of occurrence of the best solution over five runs for the instance.

Table 4.16 : Computational results for instances with 20 jobs (for experiments CPLEX 12.8.0 was used).

Instance	CPLEX		EA (Dos Santos et al. (2010))			VNS (Wang and Li (2014))			Math_2			
	z	Time (s.)	z_{best}	z_{avg}	Time (s.)	z_{best}	z_{avg}	Time (s.)	z_0	z_{best}	z_{avg}	Time (s.)
tight-equal-1-20x2	1940.06	1800	1937.49 (1)	1941.61	603	1939.11 (1)	1945.5	603	2358.02	**1936.45 (1)**	1937.52	127.92
tight-equal-2-20x2	961.44	1800	**941.69 (1)**	949.65	603	959.65 (1)	970.61	603	1322.46	943.7 (1)	957.61	126.56
tight-equal-1-20x5	3013.79	1800	2866.94 (1)	2900.40	603	2935.95 (1)	2975.49	603	4260.52	**2865.95 (1)**	2914.56	202.42
tight-equal-2-20x5	7233.35	1800	6741.16 (1)	6911.98	603	6964.91 (1)	7020.32	603	8503.53	**6613.62 (1)**	6768.30	240.31
tight-equal-1-20x10	11561.7	1800	10607.2 (1)	11151.10	603	11073.1 (1)	11393	603	16364	**10378.8 (1)**	10738.30	301.42
tight-equal-2-20x10	8501.42	1800	7310.32 (1)	7483.84	603	7317.74 (1)	7441.83	603	10774.1	**6839.3 (1)**	7023.42	360.35
loose-equal-1-20x2	2565.51	1800	**2548.95 (1)**	2553.98	603	2551.22 (1)	2554.22	603	3332.57	**2548.95 (1)**	2553.13	123.41
loose-equal-2-20x2	3105.34	1800	**3069.13 (1)**	3082.75	603	3070.16 (1)	3080.42	603	4080.6	**3069.13 (1)**	3073.32	119.31
loose-equal-1-20x5	8271.34	1800	7545.44 (1)	7715.24	603	7758.02 (1)	7839.11	603	11938.6	**7524.13 (1)**	7631.64	309.63
loose-equal-2-20x5	7793.45	1800	7252.81 (1)	7370.63	603	7302.01 (1)	7473.78	603	9992.2	**7059.38 (1)**	7140.02	225.54
loose-equal-1-20x10	5597.4	1800	5260.48 (1)	5511.91	603	5476.47 (1)	5734.55	603	8564.4	**4651.31 (1)**	4934.16	300.09
loose-equal-2-20x10	1862.41	1800	1987.65 (1)	2156.08	603	1718.08 (1)	1801.39	603	3322.76	**1597.89 (1)**	1639.36	300.69
tight-tard-1-20x2	1675.37	1800	**1671.87 (1)**	1675.10	603	1682.94 (1)	1694.69	603	2321.6	1674.6 (1)	1680.75	188.36
tight-tard-2-20x2	1489.65	1800	**1448.66 (1)**	1451.54	603	1459.57 (1)	1461.33	603	2054.31	1451.61 (1)	1454.81	119.28
tight-tard-1-20x5	3821.29	1800	3672.18 (1)	3704.30	603	3669.09 (1)	3712.62	603	4723.24	**3620.61 (1)**	3651.52	369.08
tight-tard-2-20x5	1881.02	1800	1858.19 (1)	1869.58	603	1976.11 (1)	2004.96	603	2653.29	**1792.78 (1)**	1819.30	276.96
tight-tard-1-20x10	4984.31	1800	4853.81 (1)	4977.21	603	5536.64 (1)	5702.14	603	7973.57	**4445.59 (1)**	4709.70	368.69
tight-tard-2-20x10	4057.01	1800	3358.61 (1)	3485.34	603	3245.9 (1)	3432.98	603	5471.32	**3085.54 (1)**	3153.98	349.36
loose-tard-1-20x2	1265.36	1800	1206.97 (5)	1206.97	603	**1205.59 (1)**	1208.62	603	1979.16	1206.4 (1)	1227.59	139.23
loose-tard-2-20x2	772.89	1800	**769.35 (1)**	771.42	603	790.12 (1)	801.548	603	1317.53	**769.35 (1)**	780.47	136.06
loose-tard-1-20x5	3548.14	1800	2938.24 (1)	3043.90	603	3517.2 (1)	3650.07	603	5391.36	**2931.11 (1)**	3047.98	265.64
loose-tard-2-20x5	4048.77	1800	3722.61 (1)	3771.55	603	3684.67 (1)	3821.16	603	5403.89	**3627.51 (1)**	3737.41	280.08
loose-tard-1-20x10	6494.34	1800	5093.89 (1)	5421.98	603	5478.21 (1)	5640.81	603	9235.84	**4817.32 (1)**	5204.46	421.29
loose-tard-2-20x10	1545.95	1800	1570.64 (1)	1658.06	603	1599.55 (1)	1627.43	603	2464.99	**1401.28 (1)**	1448.16	397.56

Numbers in bold indicate the best solution among four methods.
Numbers in the parentheses indicate the number of occurrence of the best solution over five runs for the instance.

4.6 Conclusion

In this chapter we developed two efficient matheuristic algorithms to tackle the computationally intractable JIT-JSS problem. To generate improved solutions, we implemented both classical and novel neighbourhood structures. We tested the impact of our proposed neigbourhoods and showed their efficacy in obtaining high quality solutions. To evaluate the performance of proposed algorithms, as well as the quality of solutions, we conducted comprehensive computational experiments to treat a set of 72 benchmark instances of JIT-JSS. Furthermore, we compared the performance of our matheuristics and state-of-the-art solution algorithms for JIT-JSS in the extant literature. Overall, our proposed matheuristics produce new best solutions for a large number of tested instances, including new best solutions for about 80% of the instances with 20 jobs, which include up to 200 operations and are among the most difficult instances in the set.

The major contributions of our study lie in providing effective solution methods for JIT-JSS, which are able to produce new best solutions for a large number of benchmark instances, and the relaxation neighbourhoods. To choose relaxed operations we proposed a distribution-based selection mechanism within the neighbourhood structures. We believe the selection of sub-sequences can be further improved by directing the search towards more expensive sub-sequences.

Chapter 5

Conclusion

In this thesis we investigated JIT machine and shop scheduling problems. In particular we focused on classical scheduling problems with applications in the air traffic control and manufacturing systems. Contrary to most studies, we opted a less paved road by grafting matheuristics onto the existing local search procedures. Through extensive computational experiments, it was verified that new relaxation neighbourhoods are able to guide the search to more promising regions in the solution space.

This chapter outlines the merits of the algorithmic framework presented in this thesis. It highlights how the objectives and aims set in Chapter 1 (see Section 1.2) were achieved and limitations of proposed matheuristics. Recommendations and a few potential areas for future research are also given.

5.1 Summary of contributions

In order to address some of the drawbacks of existing solution methodologies we utilized a “Relax-and-Solve” (R&S) framework. R&S consists of two components of ”Relax” and ”Solve”. At each iteration, a subsequence is chosen and its associated operations are relaxed. Then the partially relaxed sequence is solved using a solver (e.g. CPLEX). The “relax” phase allows the operations in the sub-sequence to be able to change their order, whereas the “solve” ensures a feasible (improved) schedule is constructed. By applying R&S:

- We managed to address problems from very different settings with the same algorithmic framework. In spite of dealing with problems from machine and shop scheduling environments in Chapters 3 and 4, we developed heuristics with same building block. In this regard, R&S allowed us to achieve one of the research objectives of this study to derive a problem-independent resolution method to address optimization problems in different settings. In this respect our methodology bears similarities with Large Neighbourhood Search (LNS) (Shaw, 1998; Ropke and Pisinger, 2006) which also tackles challenging combinatorial optimization problems by applying *destroy* and *repair* techniques. We believe general and high-level nature of our proposed framework can fill the void of a unified solution methodology in highly scattered literature of JIT scheduling as noted by Bülbül and Kaminsky (2013);

- We developed effective solution methods which are able to obtain high-quality solutions for large instances in reasonable amounts of time. In particular in Chapter 3 we showed that the presented algorithm solves the largest instances in about one minute, and is therefore suitable for practical settings. In the case of JIT-JSS, the matheuristic algorithms found new best solutions for more than half of the benchmark instances, including new best solutions for 80% of the instances with 20 jobs for the first time. This is in line with one of our research aims to design efficient algorithms that are capable to tackle large instances. Also by exploiting recent advances on mathematical programming techniques (and using an exact solver), we attained another research goal to design robust heuristics;

- One of primary goals of this research was to develop a simple solution methodology which in addition to being capable of competing with state-of-the-art algorithms could be easily tuned. As discussed in Chapter 2, although traditional manipulation techniques often require meticulous parameter tuning and quickly make improvement to initial solutions, they often fail to guide the algorithm towards generating high quality sequences. To overcome these shortcomings we introduced novel relaxation based neighborhoods in which sequencing/allocation decisions are delegated to solvers and as a result the impact of traditional random or myopic moves is minimized. To demonstrate the effectiveness of our novel neighborhoods, we compared proposed matheuristics against state-of-art algorithms in Chapters 3 and 4. Our extensive computational experiments suggest that the proposed neighborhood structures deliver solutions which for many instances are superior to those of existing methods. While the state-of-the-art methods utilize various components within the heuristics and meta-heuristics, leading therefore to implementation of advanced algorithmic techniques and parameters tuning, our proposed R&S's structure is both simple and straightforward which remarkably simplifies parameter tuning.

5.2 Limitations of the study

In this thesis we only tested R&S with CPLEX (ILOG, 2017). Given the obtained results we concluded that our matheuristic outperforms existing solution methods. However, due to delegation of sequencing decisions to the solver, the performance of our framework heavily depends on that of the solver. As such, although it is expected that similar (or superior) results can be achieved using other commercial solvers such as GUROBI (Gurobi Optimization, 2018) or XPRESS (Xpress, 2020), integrating R&S with open source optimization packages like GPLK (Makhorin, 2019) or LP-SOLVE (Berkelaar and Notebaert, n.d.) may not lead to the same performance particularly for large sized instances.

Although R&S managed to obtain a large number of best known solutions for ALP and ASP, also significantly improved the upper bounds for JIT-JSS, its architecture restricts estimation of lower bounds to further assess its performance. Nevertheless, we believe our upper bounds can indirectly contribute to obtaining better lower bounds. For instance in case of JIT-JSS, best solutions found by R&S can be utilized to speed up the convergence of Lagrangian Relaxations of Baptiste et al. (2008) and Tanaka et al. (2015).

5.3 Future research directions

The following areas for possible future research are recommended:

- In this research a distribution-based selection mechanism was used to guide the neighbourhoods. In this regard, the selection of sub-sequences can be further improved by directing the search towards more expensive sub-sequences. For instance the information associated with the current solution such as the completion time, waiting time or reduced costs of each operation in the current solution can be used to select the next set of operations to be relaxed.

- Future research can also be directed towards designing more advanced relaxation-based neighborhoods that are able to extract problem-specific information to guide the search. For instance deducing dominance rules can reduce the number of moves to be explored by the solver for relaxed set. Moreover while we only used MIP solver (i.e. CPLEX) to re-optimize the sub-problems, one can also solve this using CP solvers. Indeed using hybrid methods leveraging CP/MIP in which the information between solvers is exchanged is another interesting research path.

- Similar to classical manipulation techniques, our novel neighbourhood structures can also be applied to deal with a myriad of optimization problems. In this study we adapted them for sequencing decisions (resource constraints) which are the main source for the complexity of scheduling problems. We believe their applications can be explored for other optimization problems in different realms by partially destructing and re-optimizing complicating constraints.

- The Covid 19 pandemic exposed vulnerabilities in global supply chains causing many organizations to rethink their supply chain strategies. On the other hand excessive insistence on JIT principles like waste elimination often in the form of cutting corners has left many firms with very little margin for error. Removing nearly all the redundancies and imposing inventory costs on supplier side may render systems which are susceptible to disruptions such as the case with Coronavirus pandemic. As noted by Zhu et al. (2020), JIT is a powerful tool in times of normalcy but also a strong cause of the shortages in the event of a crisis. This necessitates the need to introduce resilience into JIT systems. To the best of our knowledge, resilient JIT models are almost non-existent in the literature. Thus constructing baseline schedules which can be recovered within a reasonable time can be another interesting research avenue.

Appendix A

Comparison of Relax_1 and Relax_2 for ALP and ASP

Following one of examiner's suggestion, we conducted new experiments comparing Relax_1 and Relax_2 for ALP* (see Table A.1) and ASP (Table A.2) instances. However, since we did not have access to our original UTS computer, we performed experiments on a different personal computer. We ran each algorithm five times. In Tables A.1 and A.2, the first column shows the names of the instances. Columns two to four show the outcomes of Relax_1, namely the best objective function value ("z_{best}"), the average objective function value over five runs ("z_{avg}") and the average computational time in seconds ("Time (s.)"), and columns five to seven denote those of Relax_2. The value of time is averaged over five runs.

*Since the initial solution generator for Relax_2 is SiPSi (which is a single machine solver), we only ran experiments for single runway case.

Table A.1 : The comparison of Relax_1 and Relax_2 for ALP instances ($m = 1$) (for experiments CPLEX 12.6.0 was used).

Instance	Relax_1			Relax_2		
	z_{best}	z_{avg}	Time (s.)	z_{best}	z_{avg}	Time (s.)
Airland1	700	700	0.675	700	700	0.7028
Airland2	1480	1480	1.6526	1480	1480	1.4832
Airland3	820	820	2.0068	820	820	1.4046
Airland4	2520	2520	1.7844	2520	2520	1.9712
Airland5	3100	3100	3.77	3100	3100	3.549
Airland6	24442	24442	1.4834	24442	24442	1.0298
Airland7	1550	1550	2.177	1550	1550	2.5986
Airland8	1950	1950	9.842	1950	1950	9.9304
Airland9	5611.7	5611.7	14.6016	5611.7	5611.7	15.4692
Airland10	12379.2	12430.9	54.3588	12292.2	12361.8	54.9938
Airland11	12418.3	12418.3	25.3804	12418.3	12418.3	27.0212
Airland12	16176.7	16239.9	56.5442	16176.7	16176.7	58.188
Airland13	37433	37450.5	143.689	37433	37437.4	162.907

Table A.2 : The comparison of Relax_1 and Relax_2 for ASP instances (for experiments CPLEX 12.6.0 was used).

Instance	Relax_1			Relax_2		
	z_{best}	z_{avg}	Time (s.)	z_{best}	z_{avg}	Time (s.)
FPT01	265	265	13.0102	265	265	8.2308
FPT02	293	293	14.8266	293	293	13.8058
FPT03	255	255	11.9394	255	255	11.3242
FPT04	268	268	13.5212	268	268	10.0774
FPT05	249	249	14.034	249	249	8.954
FPT06	167	167	12.204	167	167	8.0294
FPT07	201	201	14.211	198	198	12.3236
FPT08	167	167	16.5864	167	167	10.8702
FPT09	186	186	12.288	183	183	11.583
FPT10	214	214	10.9338	211	211	9.9458
FPT11	229	229	15.5752	229	229	9.0542
FPT12	207	207	11.3706	207	207	7.9364
FPT13	604	608.8	52.1458	604	604	25.4552
FPT14	2053	2132.2	72.6906	1994	1998	40.216
FPT15	798	800.8	48.0672	796	796	20.0854
FPT16	1324	1359.6	69.107	1316	1316	46.2382
FPT17	2506	2679	78.6326	2372	2385.6	43.5766
FPT18	1563	1597	72.6354	1512	1512	40.9234
FPT19	2167	2355.8	82.4556	2115	2115	47.9506
FPT20	3459	4026.2	83.1324	3063	3063	51.6024
FPT21	4461	4873.8	83.4232	3597	3620.6	70.2284
FPT22	3711	4041.8	96.3036	2920	2920	63.1934
FPT23	4865	5423.4	95.9728	3649	3649	63.7716
FPT24	5057	5623.8	94.2338	3695	3698.6	67.5946
FPT25	4982	5274.8	108.088	3786	3786	87.7462
FPT26	5598	5992.4	111.753	4155	4155	101.383
FPT27	5741	6071.4	109.607	4173	4176.2	112.722

Appendix B

Comparison of Math_1 and Math_2 for JIT-JSS

Following one of examiner's suggestion, we conducted new experiments comparing Math_1 and Math_2 for JIT-JSS instances (see Table B.1). However, since we did not have access to our original UTS computer, we performed experiments on a different personal computer. We ran each algorithm five times. In Table B.1, the first column shows the names of the instances. Columns two to four show the outcomes of Math_1, namely the best objective function value ("z_{best}"), the average objective function value over five runs ("z_{avg}") and the average computational time in seconds ("Time (s.)"), and columns five to seven denote those of Math_2. The value of time is averaged over five runs.

Table B.1 : The comparison of Math_1 and Math_2 for JIT-JSS instances (for experiments CPLEX 12.6.0 was used).

Instance	Math_1			Math_2		
	z_{best}	z_{avg}	Time (s.)	z_{best}	z_{avg}	Time (s.)
tight-equal-test1-10x2	461.96	470.596	6.0634	461.96	461.96	55.7598
tight-equal-test2-10x2	448.32	448.32	6.9266	448.32	448.32	44.2392
tight-equal-test1-10x5	689.11	698.394	11.2766	689.11	715.91	62.987
tight-equal-test2-10x5	763.24	768.964	9.6284	763.24	769.3	76.5788
tight-equal-test1-10x10	1276.23	1289.75	21.0958	1276.23	1277.32	74.9544
tight-equal-test2-10x10	1866.92	2081.12	21.7508	1866.92	1872.98	119.549
loose-equal-test1-10x2	224.84	234.924	7.4714	224.84	224.84	48.2056
loose-equal-test2-10x2	324.43	346.73	4.1788	319.37	325.886	48.0938
loose-equal-test1-10x5	1767.07	1827.61	15.89	1719.63	1746.87	71.5264
loose-equal-test2-10x5	971.96	1017.79	8.013	971.96	978.796	63.2058
loose-equal-test1-10x10	360.74	369.078	17.7974	364.39	365.72	78.8674
loose-equal-test2-10x10	249.85	254.154	24.6322	249.85	249.85	85.0184
tight-tard-test1-10x2	179.46	183.644	6.8594	179.46	179.46	41.0294
tight-tard-test2-10x2	145.37	146.586	5.4492	145.37	147.802	61.7218
tight-tard-test1-10x5	387.3	393.464	12.7918	386.57	389.318	69.4222
tight-tard-test2-10x5	635.93	637.222	9.7178	627.45	629.146	65.2264
tight-tard-test1-10x10	687.65	719.662	23.4762	668.14	708.036	102.142
tight-tard-test2-10x10	777.85	787.872	18.3792	777.85	779.472	100.656
loose-tard-test1-10x2	416.44	424.542	3.9148	416.44	416.44	23.2934
loose-tard-test2-10x2	137.94	158.22	7.8016	137.94	137.94	29.2476
loose-tard-test1-10x5	175.08	179.59	13.296	175.08	175.08	63.0514
loose-tard-test2-10x5	504.36	517.486	12.61	485.06	503.168	63.085
loose-tard-test1-10x10	375.71	376.906	19.271	374.2	377.244	76.8022
loose-tard-test2-10x10	144.94	144.94	12.2222	144.94	144.94	66.0602
tight-equal-test1-15x2	3344.54	3344.54	55.1374	3344.54	3345.62	90.2058
tight-equal-test2-15x2	1479.76	1479.76	35.364	1479.76	1481.31	91.5046
tight-equal-test1-15x5	1318.68	1350.33	72.8858	1318.68	1336.2	167.592
tight-equal-test2-15x5	2897.51	2940.67	67.221	2632.22	2680.67	157.065
tight-equal-test1-15x10	6950.03	7257.76	102.186	6952.99	7403.77	217.464
tight-equal-test2-15x10	4750.25	5044.12	119.916	4849.68	5238.89	211.028
loose-equal-test1-15x2	1041.33	1054.47	25.6126	1041.33	1041.4	91.252
loose-equal-test2-15x2	505.16	516.932	22.1472	505.16	516.72	84.0696
loose-equal-test1-15x5	3207.45	3287.7	100.355	3196.85	3311.73	177.947
loose-equal-test2-15x5	3276.3	3335.37	81.1986	3283.43	3353.25	193.254
loose-equal-test1-15x10	947.52	963.446	101.998	924.82	947.518	205.316
loose-equal-test2-15x10	1522.04	1543.09	124.447	1490.27	1564.8	183.417
tight-tard-test1-15x2	790.5	798.182	72.2042	790.66	794.532	112.056
tight-tard-test2-15x2	905.37	908.336	34.3016	905.37	905.77	112.353
tight-tard-test1-15x5	1359.18	1378.59	95.0292	1370.55	1374.02	152.229
tight-tard-test2-15x5	679.45	691.54	78.278	689.76	697.162	194.178
tight-tard-test1-15x10	776.39	787.498	123.625	797.3	818.072	206.422
tight-tard-test2-15x10	1232.37	1290.13	125.155	1231.14	1291.7	199.894
loose-tard-test1-15x2	654.84	654.84	51.085	654.84	654.84	93.7398
loose-tard-test2-15x2	279.71	310.452	34.2182	290.88	296.052	85.0284
loose-tard-test1-15x5	1281.26	1316.52	102.368	1281.41	1290.68	159.471
loose-tard-test2-15x5	341.03	381.998	136.254	329.34	343.094	154.921
loose-tard-test1-15x10	277.24	278.168	106.316	280.54	281.454	194.864
loose-tard-test2-15x10	587.9	633.276	110.098	630.99	654.228	223.13
tight-equal-test1-20x2	1933.72	1936.45	111.055	1937.49	1943.91	172.688
tight-equal-test2-20x2	951.28	963.138	96.5628	951.11	960.126	137.473
tight-equal-test1-20x5	2869.97	2914.79	180.706	2888.47	2908.02	291.43
tight-equal-test2-20x5	6643.07	6786.06	120.731	6724.53	6774.83	244.783
tight-equal-test1-20x10	10426.5	10666.2	133.158	10572.4	10977.8	449.783
tight-equal-test2-20x10	7303.93	7358.98	169.124	7041.96	7378.66	432.021
loose-equal-test1-20x2	2547.68	2550.3	75.3654	2556.97	2562.64	235.629
loose-equal-test2-20x2	3069.13	3096.81	88.1052	3074.73	3084.05	185.607
loose-equal-test1-20x5	7496.27	7722.58	179.542	7581.3	7667.14	281.33
loose-equal-test2-20x5	7053.66	7192.19	182.892	7100.13	7180.14	343.8
loose-equal-test1-20x10	4920.46	5048.79	150.235	5132.16	5228.04	509.341
loose-equal-test2-20x10	1626.53	1690.3	166.584	1624.55	1763.31	375.113
tight-tard-test1-20x2	1671.87	1680.69	132.157	1683.6	1696.89	176.225
tight-tard-test2-20x2	1452.05	1457.81	104.447	1449.93	1458.84	223.903
tight-tard-test1-20x5	3612.93	3662.22	173.979	3640.84	3674.64	357.169
tight-tard-test2-20x5	1809.02	1857.85	172.753	1807.29	1835.58	390.2
tight-tard-test1-20x10	4397.66	5227.82	107.833	4574	4782.67	457.433
tight-tard-test2-20x10	3198.11	3363.48	159.332	3253.29	3328.1	518.791
loose-tard-test1-20x2	1204.43	1231.65	109.736	1210.25	1225.83	183.437
loose-tard-test2-20x2	782.39	797.904	75.059	771.06	784.656	191.127
loose-tard-test1-20x5	2968.86	3065.98	164.473	3029.64	3041.16	337.323
loose-tard-test2-20x5	3609.4	3759.88	142.931	3717.15	3785.82	365.16
loose-tard-test1-20x10	5039.34	5421.44	164.783	5438.71	5697.67	598.35
loose-tard-test2-20x10	1440.72	1551.87	167.896	1498.89	1549.25	553.065

References

Abdul-Razaq, T. and Potts, C. (1988). "Dynamic programming state-space relaxation for single-machine scheduling". In: *Journal of the Operational Research Society* 39(2), pp. 141–152.

Abela, J, Abramson, D, Krishnamoorthy, M, De Silva, A, and Mills, G. (1993). "Computing optimal schedules for landing aircraft". In: *proceedings of the 12th national conference of the Australian Society for Operations Research, Adelaide*, pp. 71–90.

Adams, J., Balas, E., and Zawack, D. (1988). "The shifting bottleneck procedure for job shop scheduling". In: *Management Science* 34(3), pp. 391–401.

Ahuja, R. K., Hochbaum, D. S., and Orlin, J. B. (2003). "Solving the convex cost integer dual network flow problem". In: *Management Science* 49(7), pp. 950–964.

Al-Salem, M., Bedoya-Valencia, L., and Rabadi, G. (2016). "Heuristic and Exact Algorithms for the Two-Machine Just in Time Job Shop Scheduling Problem". In: *Mathematical Problems in Engineering* 2016.

Applegate, D. and Cook, W. (1991). "A computational study of the job-shop scheduling problem". In: *ORSA Journal on Computing* 3(2), pp. 149–156.

Arabameri, S. and Salmasi, N. (2013). "Minimization of weighted earliness and tardiness for no-wait sequence-dependent setup times flowshop scheduling problem". In: *Computers & Industrial Engineering* 64(4), pp. 902–916.

Araujo, R. P., Santos, A. G. dos, and Arroyo, J. E. C. (2009). "Genetic algorithm and local search for just-in-time job-shop scheduling". In: *Evolutionary Computation, 2009. CEC'09. IEEE Congress on.* IEEE, pp. 955–961.

Avella, P., Boccia, M., Mannino, C., and Vasilyev, I. (2017). "Time-indexed formulations for the runway scheduling problem". In: *Transportation Science* 51(4), pp. 1196–1209.

Awasthi, A., Kramer, O., and Lässig, J. (2013). "Aircraft Landing Problem: An Efficient Algorithm for a Given Landing Sequence". In: *2013 IEEE 16th International Conference on Computational Science and Engineering*, pp. 20–27.

Azizoglu, M. and Webster, S. (1997). "Scheduling about an unrestricted common due window with arbitrary earliness/tardiness penalty rates". In: *IIE Transactions* 29(11), pp. 1001–1006.

Baker, K. R. and Scudder, G. D. (1990). "Sequencing with Earliness and Tardiness Penalties: A Review". In: *Operations Research* 38(1), pp. 22–36.

Baker, K. R. and Trietsch, D. (2013). *Principles of sequencing and scheduling*. John Wiley & Sons.

Balakrishnan, H. and Chandran, B. G. (2010). "Algorithms for Scheduling Runway Operations Under Constrained Position Shifting". In: *Operations Research* 58(6), pp. 1650–1665. ISSN: 0030364X, 15265463.

Balakrishnan, N., Kanet, J. J., and Sridharan, V (1999). "Early/tardy scheduling with sequence dependent setups on uniform parallel machines". In: *Computers & Operations Research* 26(2), pp. 127–141.

Baptiste, P., Flamini, M., and Sourd, F. (2008). "Lagrangian bounds for just-in-time job-shop scheduling". In: *Computers & Operations Research* 35(3). Part Special Issue: New Trends in Locational Analysis, pp. 906 –915. ISSN: 0305-0548.

Beasley, J. E., Krishnamoorthy, M., Sharaiha, Y. M., and Abramson, D. (2000). "Scheduling aircraft landings—the static case". In: *Transportation science* 34(2), pp. 180–197.

Beck, C. and Refalo, P. (2002). "Combining local search and linear programming to solve earliness/tardiness scheduling problems". In: *In Fourth International Workshop on Integration of AI and OR Techniques (CP-AI-OR'02)*, pp. 221–235.

Beck, J. C. and Refalo, P. (2001). "A hybrid approach to scheduling with earliness and tardiness costs". In: *In Proceedings of the Third International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR'01)*, pp. 175–188.

Beck, J. C. and Refalo, P. (2003). "A hybrid approach to scheduling with earliness and tardiness costs". In: *Annals of Operations Research* 118(1), pp. 49–71.

Behnamian, J, Fatemi Ghomi, S., and Zandieh, M (2010a). "Development of a hybrid metaheuristic to minimise earliness and tardiness in a hybrid flowshop with sequence-dependent setup times". In: *International Journal of Production Research* 48(5), pp. 1415–1438.

Behnamian, J and Zandieh, M (2013). "Earliness and tardiness minimizing on a realistic hybrid flowshop scheduling with learning effect by advanced metaheuristic". In: *Arabian Journal for Science and Engineering* 38(5), pp. 1229–1242.

Behnamian, J, Zandieh, M, and Ghomi, S. F. (2010b). "Due windows group scheduling using an effective hybrid optimization approach". In: *The International Journal of Advanced Manufacturing Technology* 46(5-8), pp. 721–735.

Bennell, J. A., Mesgarpour, M., and Potts, C. N. (2011). "Airport runway scheduling". In: *4OR* 9(2), p. 115. ISSN: 1614-2411.

Berkelaar, K. M. and Notebaert, P (n.d.). *Open source (mixed-integer) linear programming system. Sourceforge.*

Bianco, L., Dell'Olmo, P., and Giordani, S. (1999). "Minimizing total completion time subject to release dates and sequence-dependentprocessing times". In: *Annals of Operations Research* 86(0), pp. 393–415. ISSN: 1572-9338.

Biskup, D. and Cheng, T. E. (1999). "Multiple-machine scheduling with earliness, tardiness and completion time penalties". In: *Computers & Operations Research* 26(1), pp. 45 –57.

Biskup, D. and Feldmann, M. (2005). "On scheduling around large restrictive common due windows". In: *European Journal of Operational Research* 162(3). Decision-Aid to Improve Organisational Performance, pp. 740 –761.

Boschetti, M. A., Maniezzo, V., Roffilli, M., and Bolufé Röhler, A. (2009). "Matheuristics: Optimization, Simulation and Control". In: *Hybrid Metaheuristics*. Ed. by M. J. Blesa, C. Blum, L. Di Gaspero, A. Roli, M. Sampels, and A. Schaerf. Springer Berlin Heidelberg: Berlin, Heidelberg, pp. 171–177. ISBN: 978-3-642-04918-7.

Brucker, P., Hilbig, T., and Hurink, J. (1999). "A branch and bound algorithm for a single-machine scheduling problem with positive and negative time-lags". In: *Discrete applied mathematics* 94(1-3), pp. 77–99.

Bülbül, K. and Kaminsky, P. (2013). "A linear programming-based method for job shop scheduling". In: *Journal of Scheduling* 16(2), pp. 161–183.

Bulhoes, T., Sadykov, R., Subramanian, A., and Uchoa, E. (2018). "On the exact solution of a large class of parallel machine scheduling problems". In.

Bürgy, R. and Bülbül, K. (2018). "The job shop scheduling problem with convex costs". In: *European Journal of Operational Research* 268(1), pp. 82–100.

Canel, C., Rosen, D., and Anderson, E. A. (2000). "Just-in-time is not just for manufacturing: a service perspective". In: *Industrial Management & Data Systems*.

Carchrae, T. and Beck, J. C. (2009). "Principles for the design of large neighborhood search". In: *Journal of Mathematical Modelling and Algorithms* 8(3), pp. 245–270.

Chandra, P., Mehta, P., and Tirupati, D. (2009). "Permutation flow shop scheduling with earliness and tardiness penalties". In: *International Journal of Production Research* 47(20), pp. 5591–5610.

Chang, P. C. (1999). "A branch and bound approach for single machine scheduling with earliness and tardiness penalties". In: *Computers & Mathematics with Applications* 37(10), pp. 133–144.

Chen, B., Potts, C. N., and Woeginger, G. J. (1998a). "A review of machine scheduling: Complexity, algorithms and approximability". In: *Handbook of combinatorial optimization*, pp. 1493–1641.

Chen, H., Chu, C., and Proth, J.-M. (1998b). "An improvement of the Lagrangean relaxation approach for job shop scheduling: a dynamic programming method". In: *IEEE transactions on robotics and automation* 14(5), pp. 786–795.

Chen, Z.-L. and Lee, C.-Y. (2002). "Parallel machine scheduling with a common due window". In: *European Journal of Operational Research* 136(3), pp. 512 –527.

Danna, E. and Perron, L. (2003). "Structured vs. unstructured large neighborhood search: A case study on job-shop scheduling problems with earliness and tardiness costs". In: *International Conference on Principles and Practice of Constraint Programming*. Springer, pp. 817–821.

Danna, E., Rothberg, E., and Le Pape, C. (2003). "Integrating mixed integer programming and local search: A case study on job-shop scheduling problems". In: *Fifth International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR'2003)*, pp. 65–79.

D'Ariano, A., Pacciarelli, D., Pistelli, M., and Pranzo, M. (2015). "Real-time scheduling of aircraft arrivals and departures in a terminal maneuvering area". In: *Networks* 65(3), pp. 212–227.

Doi, T., Nishi, T., and Voß, S. (2018). "Two-level decomposition-based matheuristic for airline crew rostering problems with fair working time". In: *European Journal of Operational Research* 267(2), pp. 428 –438. ISSN: 0377-2217.

Dos Santos, A. G., Araujo, R. P., and Arroyo, J. E. (2010). "A combination of evolutionary algorithm, mathematical programming, and a new local search procedure for the just-in-time job-shop scheduling problem". In: *International Conference on Learning and Intelligent Optimization*. Springer, pp. 10–24.

Doulabi, S. H. H., Avazbeigi, M., Arab, S., and Davoudpour, H. (2012). "An effective hybrid simulated annealing and two mixed integer linear formulations for just-in-time open shop scheduling problem". In: *The International Journal of Advanced Manufacturing Technology* 59(9-12), pp. 1143–1155.

Dumas, Y., Soumis, F., and Desrosiers, J. (1990). "Optimizing the schedule for a fixed vehicle path with convex inconvenience costs". In: *Transportation Science* 24(2), pp. 145–152.

Easton, F. F. and Moodie, D. R. (1999). "Pricing and lead time decisions for make-to-order firms with contingent orders". In: *European Journal of operational research* 116(2), pp. 305–318.

Eilon, S. and Chowdhury, I. (1977). "Minimising waiting time variance in the single machine problem". In: *Management Science* 23(6), pp. 567–575.

Erel, E. and Ghosh, J. B. (2008). "FPTAS for half-products minimization with scheduling applications". In: *Discrete Applied Mathematics* 156(15), pp. 3046–3056.

Ernst, A. T., Krishnamoorthy, M., and Storer, R. H. (1999). "Heuristic and exact algorithms for scheduling aircraft landings". In: *Networks* 34(3), pp. 229–241. ISSN: 1097-0037.

Fakhrzad, M. and Heydari, M. (2008). "A heuristic algorithm for hybrid flow-shop production scheduling to minimize the sum of the earliness andf tardiness costs". In: *Journal of the Chinese Institute of Industrial Engineers* 25(2), pp. 105–115.

Faye, A. (2015). "Solving the Aircraft Landing Problem with time discretization approach". In: *European Journal of Operational Research* 242(3), pp. 1028 – 1038. ISSN: 0377-2217.

Fernandez-Viagas, V., Dios, M., and Framinan, J. M. (2016). "Efficient constructive and composite heuristics for the permutation flowshop to minimise total earliness and tardiness". In: *Computers & Operations Research* 75, pp. 38–48.

Fernandez-Viagas, V. and Framinan, J. M. (2015). "NEH-based heuristics for the permutation flowshop scheduling problem to minimise total tardiness". In: *Computers & Operations Research* 60, pp. 27–36.

Flamini, M. and Pacciarelli, D. (2008). "Real time management of a metro rail terminus". In: *European Journal of Operational Research* 189(3), pp. 746–761.

Fox, M. S. and Smith, S. F. (1984). "ISIS—a knowledge-based system for factory scheduling". In: *Expert Systems* 1(1), pp. 25–49.

French, S. (1982). *Sequencing and scheduling: an introduction to the mathematics of the job-shop*. Ellis Horwood Ltd, Publisher.

Fuentes, M., Cadarso, L., and Marín, Ángel (2018). "A Fix & Relax matheuristic for the Crew Scheduling Problem". In: *Transportation Research Procedia* 33. XIII Conference on Transport Engineering, CIT2018, pp. 307 –314. ISSN: 2352-1465.

Furini, F., Kidd, M. P., Persiani, C. A., and Toth, P. (2015). "Improved rolling horizon approaches to the aircraft sequencing problem". In: *Journal of Scheduling* 18(5), pp. 435–447. ISSN: 1099-1425.

Furini, F., Persiani, C. A., and Toth, P. (2012). "Aircraft sequencing problems via a rolling horizon algorithm". In: *International Symposium on Combinatorial Optimization*. Springer, pp. 273–284.

Gao, K. Z., Suganthan, P. N., Tasgetiren, M. F., Pan, Q. K., and Sun, Q. Q. (2015). "Effective ensembles of heuristics for scheduling flexible job shop problem with new job insertion". In: *Computers & Industrial Engineering* 90, pp. 107–117.

Gao, X. (2018). "Corporate cash hoarding: The role of just-in-time adoption". In: *Management Science* 64(10), pp. 4858–4876.

Garey, M. R., Johnson, D. S., and Sethi, R. (1976). "The complexity of flowshop and jobshop scheduling". In: *Mathematics of Operations Research* 1(2), pp. 117–129.

Garey, M. R., Tarjan, R. E., and Wilfong, G. T. (1988). "One-processor scheduling with symmetric earliness and tardiness penalties". In: 13(2), pp. 330–348.

Gélinas, S. and Soumis, F. (2005). "Dantzig-Wolfe decomposition for job shop scheduling". In: *Column generation*. Springer, pp. 271–302.

Gen, M., Tsujimura, Y., and Kubota, E. (1994). "Solving job-shop scheduling problems by genetic algorithm". In: *Systems, Man, and Cybernetics, 1994. Humans, Information and Technology., 1994 IEEE International Conference on*. Vol. 2. IEEE, pp. 1577–1582.

Ghoniem, A. and Farhadi, F. (2015). "A column generation approach for aircraft sequencing problems: a computational study". In: *Journal of the Operational Research Society* 66(10), pp. 1717–1729. ISSN: 1476-9360.

Ghoniem, A., Farhadi, F., and Reihaneh, M. (2015). "An accelerated branch-and-price algorithm for multiple-runway aircraft sequencing problems". In: *European Journal of Operational Research* 246(1), pp. 34 –43. ISSN: 0377-2217.

Giffler, B. and Thompson, G. L. (1960). "Algorithms for Solving Production-Scheduling Problems". In: *Operations Research* 8(4), pp. 487–503.

Girish, B. (2016). "An efficient hybrid particle swarm optimization algorithm in a rolling horizon framework for the aircraft landing problem". In: *Applied Soft Computing* 44, pp. 200 –221. ISSN: 1568-4946.

Gomes, M. C., Barbosa-Póvoa, A. P., and Novais, A. Q. (2013). "Reactive scheduling in a make-to-order flexible job shop with re-entrant process and assembly: a mathematical programming approach". In: *International Journal of Production Research* 51(17), pp. 5120–5141.

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Kan, A. R. (1979). "Optimization and approximation in deterministic sequencing and scheduling: a survey". In: *Annals of discrete mathematics*. Vol. 5. Elsevier, pp. 287–326.

Gurobi Optimization, L. (2018). *Gurobi optimizer reference manual*.

Hall, N. G., Kubiak, W., and Sethi, S. P. (1991). "Earliness-Tardiness Scheduling Problems, II: Deviation of Completion Times About a Restrictive Common Due Date". In: *Operations Research* 39(5), pp. 847–856.

Han, Z., Zhu, Y., and Lin, S. (2015). "A dynamic co-evolution compact genetic algorithm for E/T problem". In: *IFAC-PapersOnLine* 48(28), pp. 1439–1443.

Hansen, J. V. (2004). "Genetic search methods in air traffic control". In: *Computers & Operations Research* 31(3), pp. 445–459.

Heizer, J. and Render, B. (2013). *Operations management: sustainability and supply chain management*. Pearson Higher Ed.

Helber, S. and Sahling, F. (2010). "A fix-and-optimize approach for the multi-level capacitated lot sizing problem". In: *International Journal of Production Economics* 123(2), pp. 247 –256. ISSN: 0925-5273.

Hu, X.-B. and Chen, W.-H. (2005). "Genetic algorithm based on receding horizon control for arrival sequencing and scheduling". In: *Engineering Applications of Artificial Intelligence* 18(5), pp. 633–642.

Huang, R.-H., Yang, C.-L., and Cheng, W.-C. (2013). "Flexible job shop scheduling with due window—a two-pheromone ant colony approach". In: *International Journal of Production Economics* 141(2), pp. 685–697.

ILOG, I. (Jan. 2017). *IBM ILOG CPLEX V12.8.0: User's manual for CPLEX*.

Janiak, A., Janiak, W., Kovalyov, M. Y., Kozan, E., and Pesch, E. (2013). "Parallel machine scheduling and common due window assignment with job independent earliness and tardiness costs". In: *Information Sciences* 224, pp. 109 –117.

Janiak, A., Kovalyov, M. Y., and Marek, M. (2007a). "Soft due window assignment and scheduling on parallel machines". In: *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 37(5), pp. 614–620.

Janiak, A., Kozan, E., Lichtenstein, M., and Oğuz, C. (2007b). "Metaheuristic approaches to the hybrid flow shop scheduling problem with a cost-related criterion". In: *International journal of production economics* 105(2), pp. 407–424.

Johnson, S. M. (1954). "Optimal two-and three-stage production schedules with setup times included". In: *Naval research logistics quarterly* 1(1), pp. 61–68.

Jolai, F., Sheikh, S., Rabbani, M., and Karimi, B. (2009). "A genetic algorithm for solving no-wait flexible flow lines with due window and job rejection". In: *The International Journal of Advanced Manufacturing Technology* 42(5-6), p. 523.

Kedad-Sidhoum, S., Solis, Y. R., and Sourd, F. (2008). "Lower bounds for the earliness–tardiness scheduling problem on parallel machines with distinct due dates". In: *European Journal of Operational Research* 189(3), pp. 1305–1316.

Kelbel, J. and Hanzálek, Z. (2007). "Constraint programming search procedure for earliness/tardiness job shop scheduling problem". In: *Proc. of the 26th Workshop of the UK Planning and Scheduling Special Interest Group*, pp. 67–70.

Kelbel, J. and Hanzálek, Z. (2011). "Solving production scheduling with earliness/tardiness penalties by constraint programming". In: *Journal of Intelligent Manufacturing* 22(4), pp. 553–562.

Kellerer, H., Rustogi, K., and Strusevich, V. A. (2018). "A fast FPTAS for single machine scheduling problem of minimizing total weighted earliness and tardiness about a large common due date". In: *Omega*.

Khare, A. and Agrawal, S. (2019). "Scheduling hybrid flowshop with sequence-dependent setup times and due windows to minimize total weighted earliness and tardiness". In: *Computers & Industrial Engineering* 135, pp. 780–792.

Kovalyov, M. Y. and Kubiak, W. (1999). "A fully polynomial approximation scheme for the weighted earliness–tardiness problem". In: *Operations Research* 47(5), pp. 757–761.

Kramer, A. and Subramanian, A. (2017). "A unified heuristic and an annotated bibliography for a large class of earliness–tardiness scheduling problems". In: *Journal of Scheduling*, pp. 1–37.

Kubiak, W., Lou, S., and Sethi, S. (1990). "Equivalence of mean flow time problems and mean absolute deviation problems". In: *Operations Research Letters* 9(6), pp. 371–374.

Laborie, P. and Godard, D. (2007). "Self-adapting large neighborhood search: Application to single-mode scheduling problems". In: *Proceedings MISTA-07, Paris* 8.

Laborie, P. and Rogerie, J. (2016). "Temporal linear relaxation in IBM ILOG CP Optimizer". In: *Journal of Scheduling* 19(4), pp. 391–400.

Lancia, G., Rinaldi, F., and Serafini, P. (2011). "A time-indexed LP-based approach for min-sum job-shop problems". In: *Annals of Operations Research* 186(1), pp. 175–198.

Lauff, V. and Werner, F. (2004). "On the complexity and some properties of multi-stage scheduling problems with earliness and tardiness penalties". In: *Computers & Operations Research* 31(3), pp. 317–345.

Lee, C.-Y. and Choi, J. Y. (1995). "A genetic algorithm for job sequencing problems with distinct due dates and general early-tardy penalty weights". In: *Computers & Operations Research* 22(8), pp. 857–869.

Li, G. (1997). “Single machine earliness and tardiness scheduling”. In: *European Journal of Operational Research* 96(3), pp. 546–558.

Liaw, C.-F. (1999). “A branch-and-bound algorithm for the single machine earliness and tardiness scheduling problem”. In: *Computers & Operations Research* 26(7), pp. 679–693.

Lin, H.-F. (1998). “A heuristic solution to the total tardiness and earliness penalties of an m-machine nonpreemptive open-shop schedule”. In: *Journal of the Chinese Institute Of Industrial Engineers* 15(2), pp. 159–167.

Liu, S. Q. and Kozan, E. (2011). “Scheduling Trains with Priorities: A No-Wait Blocking Parallel-Machine Job-Shop Scheduling Model”. In: *Transportation Science* 45(2), pp. 175–198.

Madhushini, N, Rajendran, C., and Deepa, Y (2009). “Branch-and-bound algorithms for scheduling in permutation flowshops to minimize the sum of weighted flowtime/sum of weighted tardiness/sum of weighted flowtime and weighted tardiness/sum of weighted flowtime, weighted tardiness and weighted earliness of jobs”. In: *Journal of the Operational Research Society* 60(7), pp. 991–1004.

Makhorin, A (2019). *The GNU Linear Programming Kit (GLPK). GNU Software Foundation, 2000.*

Maniezzo, V., Sttzle, T., and Vo, S. (2009). *Matheuristics: Hybridizing Metaheuristics and Mathematical Programming.* 1st. Springer Publishing Company, Incorporated. ISBN: 144191305X, 9781441913050.

Mason, S. J., Fowler, J. W., and Matthew Carlyle, W (2002). “A modified shifting bottleneck heuristic for minimizing total weighted tardiness in complex job shops”. In: *Journal of Scheduling* 5(3), pp. 247–262.

Mason, S. J., Jin, S., and Jampani, J. (2009). “A moving block heuristic to minimise earliness and tardiness costs on parallel machines”. In: *International Journal of Production Research* 47(19), pp. 5377–5390.

McLachlin, R. (1997). “Management initiatives and just-in-time manufacturing”. In: *Journal of Operations management* 15(4), pp. 271–292.

Mladenović, N. and Hansen, P. (1997). “Variable neighborhood search”. In: *Computers & Operations Research* 24(11), pp. 1097–1100.

Mönch, L. and Drießel, R. (2005). “A distributed shifting bottleneck heuristic for complex job shops”. In: *Computers & Industrial Engineering* 49(3), pp. 363–380.

Mönch, L., Schabacker, R., Pabst, D., and Fowler, J. W. (2007). "Genetic algorithm-based subproblem solution procedures for a modified shifting bottleneck heuristic for complex job shops". In: *European Journal of Operational Research* 177(3), pp. 2100–2118.

Monden, Y. (2011). *Toyota production system.* CRC Press.

Monette, J.-N., Deville, Y., and Van Hentenryck, P. (2009). "Just-In-Time Scheduling with Constraint Programming." In: *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS).*

Mosheiov, G. (2003). "Scheduling unit processing time jobs on an m-machine flow-shop". In: *Journal of the Operational Research Society* 54(4), pp. 437–441.

M'Hallah, R (2014a). "An iterated local search variable neighborhood descent hybrid heuristic for the total earliness tardiness permutation flow shop". In: *International Journal of Production Research* 52(13), pp. 3802–3819.

M'Hallah, R. (2007). "Minimizing total earliness and tardiness on a single machine using a hybrid heuristic". In: *Computers & Operations Research* 34(10), pp. 3126–3142.

M'Hallah, R. (2014b). "Minimizing total earliness and tardiness on a permutation flow shop using VNS and MIP". In: *Computers & Industrial Engineering* 75, pp. 142–156.

Ng, K., Lee, C., Chan, F. T., and Lv, Y. (2018). "Review on meta-heuristics approaches for airside operation research". In: *Applied Soft Computing.*

Ng, K., Lee, C., Chan, F. T., and Qin, Y. (2017). "Robust aircraft sequencing and scheduling problem with arrival/departure delay using the min-max regret approach". In: *Transportation Research Part E: Logistics and Transportation Review* 106, pp. 115 –136. ISSN: 1366-5545.

Oliver, N. (1991). "The dynamics of just-in-time". In: *New Technology, Work and Employment* 6(1), pp. 19–27.

Ow, P. S. and Morton, T. E. (1989). "The Single Machine Early/Tardy Problem". In: *Management Science* 35(2), pp. 177–191.

Pan, Q.-K., Ruiz, R., and Alfaro-Fernández, P. (2017). "Iterated search methods for earliness and tardiness minimization in hybrid flowshops with due windows". In: *Computers & Operations Research* 80, pp. 50–60.

Persona, A., Battini, D., and Rafele, C. (2008). "Hospital efficiency management: the just-in-time and Kanban technique". In: *International Journal of Healthcare Technology and Management* 9(4), pp. 373–391.

Pham, D.-N. and Klinkert, A. (2008). "Surgical case scheduling as a generalized job shop scheduling problem". In: *European Journal of Operational Research* 185(3), pp. 1011 –1025. ISSN: 0377-2217.

Pinedo, M. (2008). *Scheduling: Theory, Algorithms, and Systems.* Prentice Hall international series in industrial and systems engineering. Springer. ISBN: 9780387789347.

Pinol, H. and Beasley, J. (2006). "Scatter Search and Bionomic Algorithms for the aircraft landing problem". In: *European Journal of Operational Research* 171(2), pp. 439 –462. ISSN: 0377-2217.

Raidl, G. R. and Puchinger, J. (2008). "Combining (integer) linear programming techniques and metaheuristics for combinatorial optimization". In: *Hybrid metaheuristics.* Springer, pp. 31–62.

Ropke, S. and Pisinger, D. (2006). "An adaptive large neighborhood search heuristic for the pickup and delivery problem with Time windows". In: *Transportation science* 40(4), pp. 455–472.

Ruiz, R. and Maroto, C. (2005). "A comprehensive review and evaluation of permutation flowshop heuristics". In: *European journal of operational research* 165(2), pp. 479–494.

Ruiz, R. and Vázquez-Rodríguez, J. A. (2010). "The hybrid flow shop scheduling problem". In: *European journal of operational research* 205(1), pp. 1–18.

Sabar, N. R. and Kendall, G. (2015). "An iterated local search with multiple perturbation operators and time varying perturbation strength for the aircraft landing problem". In: *Omega* 56, pp. 88 –98. ISSN: 0305-0483.

Sabuncuoglu, I. and Lejmi, T. (1999). "Scheduling for non regular performance measure under the due window approach". In: *Omega* 27(5), pp. 555–568.

Sadeh, N. and Fox, M. S. (1990). "Variable and value ordering heuristics for activity-based job-shop scheduling". In: *Proceedings of the Fourth International Conference on Expert Systems in Production and Operations Management, Hilton Head Island, SC*, pp. 134–144.

Salehipour, A. (2019). "An algorithm for single- and multiple-runway aircraft landing problem". In: *Mathematics and Computers in Simulation.*

Salehipour, A. and Ahmadian, M. M. (2017). "A heuristic algorithm for the Aircraft Landing Problem". In: *The 22nd International Congress on Modelling and Simulation (MODSIM2017).*

Salehipour, A., Ahmadian, M. M., and Oron, D. (2018). "Efficient and simple heuristics for the Aircraft Landing Problem". In: *Matheuristic 2018 workshop (MH2018).*

Salehipour, A., Modarres, M., and Naeni, L. M. (2013). "An efficient hybrid metaheuristic for aircraft landing problem". In: *Computers & Operations Research* 40(1), pp. 207–213.

Salehipour, A., Naeni, L. M., and Kazemipoor, H. (2009). "Scheduling aircraft landings by applying a variable neighborhood descent algorithm: Runway-dependent landing time case". In: *Journal of Applied Operational Research* 1(1), pp. 39 –49.

Samá, M., D'Ariano, A., Corman, F., and Pacciarelli, D. (2018). "Coordination of scheduling decisions in the management of airport airspace and taxiway operations". In: *Transportation Research Part A: Policy and Practice* 114, pp. 398–411.

Samá, M., D'Ariano, A., D'Ariano, P., and Pacciarelli, D. (2017). "Scheduling models for optimal aircraft traffic control at busy airports: tardiness, priorities, equity and violations considerations". In: *Omega* 67, pp. 81–98.

Samá, M., D'Ariano, A., D'Ariano, P., and Pacciarelli, D. (2017). "Scheduling models for optimal aircraft traffic control at busy airports: Tardiness, priorities, equity and violations considerations". In: *Omega* 67, pp. 81 –98. ISSN: 0305-0483.

Sarin, S and Lefoka, M (1993). "Scheduling heuristic for the n-jobm-machine flow shop". In: *Omega* 21(2), pp. 229–234.

Sarper, H. (1995). "Minimizing the sum of absolute deviations about a common due date for the two-machine flow shop problem". In: *Applied mathematical modelling* 19(3), pp. 153–161.

Schaller, J and Valente, J. M. (2013a). "An evaluation of heuristics for scheduling a non-delay permutation flow shop with family setups to minimize total earliness and tardiness". In: *Journal of the Operational Research Society* 64(6), pp. 805–816.

Schaller, J. (2007). "A comparison of lower bounds for the single-machine early/tardy problem". In: *Computers & operations research* 34(8), pp. 2279–2292.

Schaller, J. and Valente, J. (2019a). "Branch-and-bound algorithms for minimizing total earliness and tardiness in a two-machine permutation flow shop with unforced idle allowed". In: *Computers & Operations Research* 109, pp. 1–11.

Schaller, J. and Valente, J. M. (2013b). "A comparison of metaheuristic procedures to schedule jobs in a permutation flow shop to minimise total earliness and tardiness". In: *International Journal of Production Research* 51(3), pp. 772–779.

Schaller, J. and Valente, J. M. (2019b). "Heuristics for scheduling jobs in a permutation flow shop to minimize total earliness and tardiness with unforced idle time allowed". In: *Expert Systems with Applications* 119, pp. 376–386.

Şen, H. and Bülbül, K. (2015). "A strong preemptive relaxation for weighted tardiness and earliness/tardiness problems on unrelated parallel machines". In: *INFORMS Journal on Computing* 27(1), pp. 135–150.

Shaw, P. (1998). In: *International conference on principles and practice of constraint programming*. Springer, pp. 417–431.

Sidney, J. B. (1977). "Optimal single-machine scheduling with earliness and tardiness penalties". In: *Operations Research* 25(1), pp. 62–69.

Sourd, F. and Kedad-Sidhoum, S. (2003). "The one-machine problem with earliness and tardiness penalties". In: *Journal of scheduling* 6(6), pp. 533–549.

Stank, T. P. and Crum, M. R. (1997). "Just-in-time management and transportation service performance in a cross-border setting". In: *Transportation journal*, pp. 31–42.

Sung, C. S. and Min, J. (2001). "Scheduling in a two-machine flowshop with batch processing machine (s) for earliness/tardiness measure under a common due date". In: *European Journal of Operational Research* 131(1), pp. 95–106.

Tanaka, S. and Araki, M. (2013). "An exact algorithm for the single-machine total weighted tardiness problem with sequence-dependent setup times". In: *Computers & Operations Research* 40(1), pp. 344–352.

Tanaka, S. and Fujikuma, S. (2012). "A dynamic-programming-based exact algorithm for general single-machine scheduling with machine idle time". In: *Journal of Scheduling* 15(3), pp. 347–361.

Tanaka, S., Fujikuma, S., and Araki, M. (2009). "An exact algorithm for single-machine scheduling without machine idle time". In: *Journal of Scheduling* 12(6), pp. 575–593.

Tanaka, S., Sadykov, R., and Detienne, B. (2015). “A new Lagrangian bound for the min-sum job-shop scheduling”. In: *International Symposium on Scheduling ISS’2015*.

Tasgetiren, M. F., Sevkli, M., Liang, Y.-C., and Gençyilmaz, G. (2004). “Particle swarm optimization algorithm for single machine total weighted tardiness problem”. In: *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753)*. Vol. 2. IEEE, pp. 1412–1419.

Vadlamani, S. and Hosseini, S. (2014). “A novel heuristic approach for solving aircraft landing problem with single runway”. In: *Journal of Air Transport Management* 40, pp. 144 –148. ISSN: 0969-6997.

Valente, J. M. and Alves, R. A. (2005). “An exact approach to early/tardy scheduling with release dates”. In: *Computers & Operations Research* 32(11), pp. 2905 – 2917.

Vallada, E. and Ruiz, R. (2012). “Scheduling unrelated parallel machines with sequence dependent setup times and weighted earliness–tardiness minimization”. In: *Just-in-Time Systems*. Springer, pp. 67–90.

Wan, G. and Yen, B. P.-C. (2002). “Tabu search for single machine scheduling with distinct due windows and weighted earliness/tardiness penalties”. In: *European Journal of Operational Research* 142(2), pp. 271 –281.

Wang, S. and Li, Y. (2014). “Variable neighbourhood search and mathematical programming for just-in-time job-shop scheduling problem”. In: *Mathematical Problems in Engineering* 2014.

Wen, M., Larsen, J., and Clausen, J. (2005). “An exact algorithm for aircraft landing problem”. In.

Weng, M. X. and Ventura, J. A. (1996). “A note on “common due window scheduling””. In: *Production and Operations Management* 5(2), pp. 194–200.

Woo, Y.-B. and Kim, B. S. (2018). “Matheuristic approaches for parallel machine scheduling problem with time-dependent deterioration and multiple rate-modifying activities”. In: *Computers & Operations Research* 95, pp. 97 –112. ISSN: 0305-0548.

Xiangwei, M., Ping, Z., and Chunjin, L. (2011). “Sliding window algorithm for aircraft landing problem”. In: *Control and Decision Conference (CCDC), 2011 Chinese*. IEEE, pp. 874–879.

Xpress (2020). *Optimizer Reference Manual.* Available at `http://www.fico.com/xpress`.

Yan, H.-S., Wan, X.-Q., and Xiong, F.-L. (2014). "A hybrid electromagnetism-like algorithm for two-stage assembly flow shop scheduling problem". In: *International Journal of Production Research* 52(19), pp. 5626–5639.

Yang, H. A., Li, J. Y., and Qi, L. L. (2012a). "An improved genetic algorithm for just-in-time job-shop scheduling problem". In: *Advanced Materials Research.* Vol. 472. Trans Tech Publ, pp. 2462–2467.

Yang, H.-a., Sun, Q.-f., Saygin, C., and Sun, S.-d. (2012b). "Job shop scheduling based on earliness and tardiness penalties with due dates and deadlines: an enhanced genetic algorithm". In: *The International Journal of Advanced Manufacturing Technology* 61(5-8), pp. 657–666.

Yeung, W., O´guz, C., and Cheng, T. E. (2004). "Two-stage flowshop earliness and tardiness machine scheduling involving a common due window". In: *International Journal of Production Economics* 90(3). Production Control and Scheduling, pp. 421 –434.

Yeung, W., O´guz, C., and Cheng, T. (2001). "Single-machine scheduling with a common due window". In: *Computers & Operations Research* 28(2), pp. 157 – 175.

Yoon, S.-H. and Ventura, J. A. (2002). "Minimizing the mean weighted absolute deviation from due dates in lot-streaming flow shop scheduling". In: *Computers & Operations Research* 29(10), pp. 1301–1315.

Yu, S.-P., Cao, X.-B., and Zhang, J. (2011). "A real-time schedule method for Aircraft Landing Scheduling problem based on Cellular Automation". In: *Applied Soft Computing* 11(4), pp. 3485–3493.

Zegordi, S., Itoh, K, and Enkawa, T (1995). "A knowledgeable simulated annealing scheme for the early/tardy flow shop scheduling problem". In: *THE INTERNATIONAL JOURNAL OF PRODUCTION RESEARCH* 33(5), pp. 1449–1466.

Zhan, Z.-h., Zhang, J., and Gong, Y.-j. (2009). "Ant colony system based on receding horizon control for aircraft arrival sequencing and scheduling". In: *Proceedings of the 11th Annual conference on Genetic and evolutionary computation.* ACM, pp. 1765–1766.

Zhang, D. and Klabjan, D. (2017). “Optimization for gate re-assignment”. In: *Transportation Research Part B: Methodological* 95, pp. 260 –284. ISSN: 0191-2615.

Zhu, G., Chou, M. C., and Tsai, C. W. (2020). “Lessons learned from the COVID-19 pandemic exposing the shortcomings of current supply chain operations: A long-term prescriptive offering”. In: *Sustainability* 12(14), p. 5858.