Scheduling under preventive maintenance and sequence-dependent setup-times constraints to minimize job rejection costs or weighted sum of completion times

Hanane Krim

N° Ordre : 19/17

Thèse de doctorat

Pour obtenir le grade de

Docteur de l'Université Polytechnique Hauts-de-France (UPHF)

Discipline : **Informatique**

Présentée et soutenue par : Hanane KRIM

Le 03/07/2019, à Valenciennes

École doctorale : Sciences Pour l'Ingénieur (SPI) – 072
Laboratoire : Laboratoire d'Automatique, de Mécanique et d'Informatique Industrielles et Humaines (LAMIH) UMR CNRS 8201

Scheduling under preventive maintenance and sequence-dependent setup-times constraints to minimize job rejection costs or weighted sum of completion times

---

Président de jury

- DHAENENS, Clarisse. Professeur des Universités, Université de Lille

Rapporteurs

- MOUKRIM, Aziz. Professeur des Universités, Université de Technologie de Compiègne
- YALAOUI, Alice. Maître de Conférences HDR, Université de Technologie de Troyes

Examinateurs

- KACEM, Imed. Professeur des Universités, Université de Lorraine, Metz
- ZUFFEREY, Nicolas. Professeur des Universités, Université de Genève, Suisse

Invités

- POTVIN, Jean-Yves. Professeur des Universités, Université de Montréal, Québec

Directeur de thèse

- DUVIVIER, David. Professeur des Universités, UPHF, Valenciennes

Encadrant de thèse

- BENMANSOUR, Rachid. Professeur assistant, INSEA, Rabat, Maroc

N° Ordre : 19/17

Thèse de doctorat

Pour obtenir le grade de

Docteur de l'Université Polytechnique Hauts-de-France (UPHF)

Discipline : **Informatique**

Présentée et soutenue par : Hanane KRIM

Le 03/07/2019, à Valenciennes

École doctorale : Sciences Pour l'Ingénieur (SPI) – 072
Laboratoire : Laboratoire d'Automatique, de Mécanique et d'Informatique Industrielles et Humaines (LAMIH) UMR CNRS 8201

Ordonnancement sous contraintes de maintenance préventive et temps de préparation dépendants de la séquence pour minimiser les coûts de rejet ou la somme pondérée des dates de fin

---

Président de jury

- DHAENENS, Clarisse. Professeur des Universités, Université de Lille

Rapporteurs

- MOUKRIM, Aziz. Professeur des Universités, Université de Technologie de Compiègne
- YALAOUI, Alice. Maître de Conférences HDR, Université de Technologie de Troyes

Examinateurs

- KACEM, Imed. Professeur des Universités, Université de Lorraine, Metz
- ZUFFEREY, Nicolas. Professeur des Universités, Université de Genève, Suisse

Invités

- POTVIN, Jean-Yves. Professeur des Universités, Université de Montréal, Québec

Directeur de thèse

- DUVIVIER, David. Professeur des Universités, UPHF, Valenciennes

Encadrant de thèse

- BENMANSOUR, Rachid. Professeur assistant, INSEA, Rabat, Maroc

Abstract

Production scheduling is considered as one of the most important tasks carried out in manufacturing systems. It allows available resources to perform a number of tasks, over a given period of time, while optimizing one or more objectives such as reducing production delays or costs associated with storage.

In France, these industries contribute significantly to the regional and national economy, making the Hauts-de-France region the fourth French economic region. In order to remain competitive, these companies must be based, on one side, on a reliable production system that is available at any time, and on the other side, on powerful Decision Support Systems to react quickly to any unpredictable situation such as failures, late deliveries of raw material or orders cancellation, etc.

Furthermore, maintenance is another aspect closely connected to production scheduling in real manufacturing settings. One of the most common assumptions in the scheduling literature is that the machines or resources are always available but, in practice, they may have to be stopped due to failures or to preventive maintenance. Taking into account that machines are an essential part of the production process and maintenance costs represent a large percentage of the total budget of operations, it is desirable to coordinate efficiently the maintenance planning and production scheduling.

This thesis addresses exactly this problem, while considering other constraints such as sequence-dependent setup times between tasks. The main objective of this work is to design and develop optimization methods for decision support systems applied to scheduling problems with unavailability constraints due to preventive maintenance. The proposed models and tools have been validated through academic problems and on the basis of simplified industrial problems. Therefore, the output of the thesis is to develop new algorithms and models based on integer linear programming, heuristics and metaheuristics to solve production scheduling problems.

Keywords: Scheduling - Maintenance - Setup-time - Job rejection - Optimization

Résumé

L'ordonnancement est considéré comme l'une des tâches les plus importantes en industrie, notamment dans les ateliers de production. Son but principal est d'allouer les ressources disponibles aux tâches sur une période donnée, tout en optimisant un ou plusieurs objectifs tels que la minimisation des délais de production et les coûts de stockage.

En France, ces industries contribuent de manière significative à l'économie régionale et nationale, faisant de la région Hauts-de-France la quatrième région économique française. Pour rester compétitives, ces sociétés doivent reposer, d'une part, sur un système de production fiable et disponible à tout moment, et d'autre part, sur de puissants outils d'aide à la décision permettant de réagir rapidement à toute situation imprévue telle qu'une panne ou un retard de livraison de matières premières, des annulation de commande, etc.

Par ailleurs, la maintenance est un autre aspect étroitement lié à l'ordonnancement de la production. L'une des hypothèses les plus courantes dans la littérature est que les machines ou les ressources sont toujours disponibles à tout moment, or, en pratique, il peut être nécessaire de les arrêter en raison de pannes ou de maintenance préventive. Compte tenu du fait que les machines sont un élément essentiel du processus de production et que les coûts de maintenance représentent un grand pourcentage du budget total des opérations, il est souhaitable de bien coordonner la planification de la maintenance et l'ordonnancement de la production.

Cette thèse aborde exactement ce problème, tout en considérant d'autres contraintes comme les temps de préparation dépendant de la sequence. L'objectif principal de ce travail est de concevoir et de développer des méthodes d'optimisation pour l'aide à la décision appliquées aux problèmes d'ordonnancement avec contrainte d'indisponibilité dûe à la maintenance préventive. Ces outils sont validés à travers des problèmes académiques et industriels simplifiés. Par conséquent, cette thèse a conduit au développement de nouveaux algorithmes et modèles basés sur la programmation linéaire en nombres entiers, des heuristiques et des métaheuristiques pour résoudre des problèmes d'ordonnancement de la production.

Mots-clefs : Ordonnancement - Maintenance - Temps de préparation - Rejet de tâches - Optimisation

Remerciements

Je tiens à exprimer toute ma gratitude à plusieurs personnes que j'ai eu la chance d'avoir à mes côtés. Sans leur aide et leur soutien, cette thèse n'aurait jamais vu le jour.

Mes remerciements et pas des moindres vont particulièrement à mon directeur de thése, le Professeur David Duvivier qui m'a mise à l'aise dès le premier jour où j'ai mis les pieds à l'UPHF ex-UVHC. Je le remercie infiniment pour sa bonne humeur, sa disponibilité, ses conseils judicieux et corrections scrupuleuses, ses n^{imes} relectures de mes cv et lettres de motivations, mais également pour son enthousiasme et ses encouragements à chaque fois que je voulais partir en mobilité que ce soit en France ou à l'étranger.

Je remercie par la même occasion, mon encadrant, le Docteur Rachid Benmansour, pour tout les conseils judicieux qui l'a pu me procurer, pour son aide précieuse à l'élaboration de différents modèles mathématiques et pour sa grande disponibilité.

De la même manière, je remercie bien évidement le Professeur Aziz Moukrim et le Docteur HDR Alice Yalaoui d'avoir accepté d'être rapporteurs de cette thèse et de m'avoir permis la soutenance de celle-ci.

Je remercie aussi tous les autres membres du jury, les Professeurs Imad Kacem, Nicolas Zufferey et Clarisse Dhaenens pour tout leurs conseils remarques pertinentes.

Je tiens à doublement remercier le Professeur Nicolas Zufferey avec qui j'ai eu la grande opportunité de collaborer. Je le remercie de m'avoir accueilli deux fois de suite, à Montréal en Juillet 2017 et à Genève en Février 2018. J'ai appris énormément de choses du Professeur Zufferey, sa rigueur, sa bonté et sa bonne humeur m'ont fortement marqué.

Je n'oublie pas de remercier le Professeur Jean-Yves Potvin qui m'a donné l'opportunité d'aller au CIRRELT à Montréal, pour ses grandes disponibilités, pour son aide, ses corrections et tout les échanges fructueux qu'on a pu avoir surplace et par emails/skype.

Je tiens aussi à dire un grand merci à mes co-auteurs pour toutes leurs remarques et corrections pertinentes, notamment le Professeur Daoud-Aït-Kadi, qui m'a chaleureusement accueilli lors de mon déplacement à l'université Laval (Québec, Canada), ainsi qu'au deux Professeurs Said Hanafi, chef de département informatique du LAMIH et Abdelhakim Artiba, président de l'UPHF.

Enfin, tous les organismes qui ont financé plusieurs de mes déplacements, je cite : EURO, le GDR RO, la ROADEF, collège doctoral Lille nord de France et la région Haut de France au travers de la bourse MERMOZ.

A tous les membres du département informatique permanents et contractuels ainsi que le pôle administratif (Marlène, Corine, Maureen, Isabelle,...). À tous mes amis et collègues (Lydia, Zeineb, Marko, Yunfei, Aymen, Ayoub et sa femme Amina, Yazid, Valérie) et tous les autres.

J'ai pu aussi compter sur le soutien sans faille de mon très cher mari Farid, mes très chers parents Nadia et Mohammed-Tahar ainsi qu'à toute ma belle-famille. Merci aussi à celle qui m'a donné la force et le courage d'avancer au travers de ses beaux sourires, ma petite princèsse, ma petite cerise sur le gâteau, ma fille Anaïs.

The present research work has been carried out in the ELSAT 2020 project supported by the European Union with the European Regional Development Fund, the French State, and the Hauts-de-France Region Council. I gratefully acknowledge the support of these institutions.

The present research work has also been carried out in the context of the LIA (International Associate Laboratory) ROI-TML (Operational Research and Computer Science in Transportation, Mobility and Logistics) between the LAMIH UMR 8201 (France) and the CIRRELT (Canada). I gratefully acknowledge the support of these laboratories and relating institutions.

À mes parents Nadia et Mohammed-Tahar,

À mon mari Farid,

À ma fille Anaïs

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we give the context of this thesis is defined, by presenting at first the research project in which this thesis is included. In the second part of this chapter, we present the working hypotheses on which our work is carried out given that the considered problem areas are theoretical and based mainly on simplified industrial problems. The chapter ends with the main contributions and the output of this thesis.

Contents

1.1 General context: ELSAT2020 project

This dissertation has been carried out in the context of the ELSAT[1] (Ecomobility, Logistics, Security and Adaptability in Transport) Horizon 2020 project. This project is co-financed by the European Union with the European Regional Development Fund, the French State, and the Hauts-de-France Region Council.

The ELSAT2020 project aims to address significant societal challenges for Transport and Ecomobility, which are safer, more intelligent, more environmentally respectful, more integrated, customized, acceptable and economically efficient. It is built around major strategic objectives and is divided into 22 sub-projects. The research carried out in the ELSAT2020 project is mainly aimed at the economic sectors of the automobile, rail and logistics transportation and new mobility.

This thesis is considered as one of the contributions in the Project 2 (P2) defined in the Strategical Objective number 2 (named OS2) of the ELSAT 2020 project.

More specifically, OS2 deals with the optimization of mobility systems and logistics and the second project (P2) in which this thesis is included concerns the Optimization of Operations in Logistics and Maintenance of Transportation Systems (OLOGMAESTRO).

As mentioned in the objective of the OS2/PS project, current logistics chains are based on different modes of transportation interacting with each other. Thus their good functioning and optimization depends on the reliability of the means of transportation systems used. As a result, the maintenance policy plays an important role to ensure both the fluidity of trade of goods and the reduction of operating costs due to breakdowns for example. In addition, manufacturers are constantly developing their transportation systems to meet a variable and personalized demand while offering a good quality of service in terms of cost, quality and time. These manufacturers are also concerned by the environmental, social and social impact of these transportation systems. Therefore, they must ensure that the availability of their transportation systems and the infrastructure used for transport are improved. The objective of this project is to achieve these goals. To do this, our approach is to provide generic decision support systems for the management of logistics operations and maintenance. These tools must take into account the different types of constraints existing in logistics in general and in transportation systems in particular.

In addition, part of the present research work has been carried out in the context of the LIA (International Associate Laboratory) ROI-TML (Operational Research and Computer Science in Transportation, Mobility and Logistics) between the LAMIH UMR CNRS 8201 (France) and the CIRRELT[2] (Canada).

1.2 Problem statement and objectives of the thesis

Nowadays, the majority of production is transported by national or international supply chains. The sites of production, processing, assembly, storage, repackaging, distribution, etc. can be scattered all over the world. The routing of goods between these sites is carried out through different modes of transport in interaction. This is especially true as the current transport systems are more and more complex and therefore require technically

[1] http://www.elsat2020.org/en
[2] https://www.cirrelt.ca/

more maintenance. Consequently, the encountered problems in practical and industrial cases related to the transport systems mentioned above can be modeled using scheduling theory.

On the other hand, an effective maintenance policy plays an important role in ensuring both the fluidity of trade (goods, people) and the reduction of operating costs. For example, rail transport, considered for a long time as important by the European Union and also by the Hauts-de-France region, is particularly characterized by its ability to reduce congestion and its low impact on traffic for the environment in relation with roads or air transport.

In the above-mentioned context, this thesis focuses on solving scheduling and maintenance problems. Indeed, *scheduling* is a decision making problem that is used on a regular basis in many manufacturing and service industries as well as in most information processing environments. It plays an important role on allowing available resources to perform a number of tasks over a given period of time. More generally we say that scheduling problems involve jobs and machines, with the aim to find the optimal ordering of the jobs over the machines in such a way to optimize one or more objective functions. Obviously there can be some constraints on the possible orderings, so the objective is to find a schedule specifying when and on which machine every job must be processed. Thereby, one of the most important constraints considered throughout this dissertation is the unavailability of one or more machines of the workshop due to a preventive maintenance policy.

Indeed, the main objective of this thesis is to design and develop optimization decision support systems for scheduling problems with enrolled maintenance policy due to failures and machine breakdowns. The proposed models and tools have been validated on through academic problems, on the basis of simplified industrial problems.

Furthermore, the output of the thesis is, ultimately, to develop new algorithms and models based on mathematical models, heuristics and metaheuristics to solve that kind of scheduling problems. Testing and validation infrastructure are based on predefined designs of experiments, consistent with the above-mentioned aspects.

1.3 Work hypothesis

In this section we are mainly interested in the different hypotheses considered in this thesis.

Firstly, in scheduling theory, the type and number of considered machines differs from one problem to another. Here, we consider two cases. Scheduling with single machine and scheduling on multiple machines. Sections 1.3.1 and 1.3.2 explain in general the industrial interest of dealing with this two kind of problems. Secondly, when the number of machines in the workshop is fixed, we are usually interested in the production environment. Indeed, two other hypotheses are then introduced, scheduling with considering maintenance policy in section 1.3.3 and sequence dependent setup times between each two tasks that should be performed in section 1.3.4. Finally, There are several performance criteria to measure the quality of a schedule. In this thesis we are interested mainly by two criteria: The weighted sum of completion times and jobs rejection cost criteria. The interest of considering these two criteria are discussed respectively in sections 1.3.5 and 1.3.6.

1.3.1 Critical machine based scheduling

In scheduling theory, a single-machine scheduling or single-resource scheduling is defined as the process which assigns a group of tasks (jobs) to a single machine or resource in the shop. Generally, this machine (resource) is defined as the critical machine (resource) in the shop floor in which the workload is accentuated. Is is also the most crucial or vital machine in a shop floor where jobs are ranked so that one or many performance measures (objective functions) should be optimized.

Figures 1.1 and 1.2 describe two critical machine cases in a workshop. Indeed, when a machine is in the middle of the production chain or at the very beginning of this one (other layouts are also possible), it is considered as critical because if it breaks down the whole workshop production will be delayed or blocked.

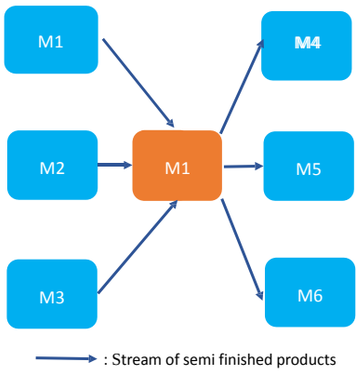

Figure 1.1: First illustration of a critical machine in a workshop

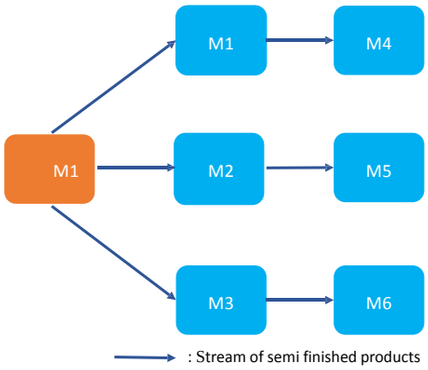

Figure 1.2: Second illustration of a critical machine in a workshop

Some examples for critical machines in transport systems

Critical machines based scheduling has raised a special interest for several years. The transportation system machine scheduling in one of the examples of critical machine based scheduling. Figure 1.3 shows a container loading and unloading crane and Figure 1.4 a rotating hub in railways systems. These two resources are most

Figure 1.3: Loading and unloading crane

Figure 1.4: Rotating hub in railways systems

of the time considered as critical machines because of their vital and crucial role in the whole system process. In this perspective, many researchers investigated this area. (Luo et al., 2016) investigated the integration of the two problems focusing on the unloading process in an automated container terminal, where all or part of the equipment are automated. Further details concerning the critical machine in the shop floor are available in (Vivek et al., 2017). Furthermore, as mentioned by Pinedo in his book (Pinedo, 2016), in theory, single machine models are important for various reasons. The single machine environment is very simple and a special case of all other environments. Single machine models often have properties that neither machines in parallel nor machines in series have. Furthermore the results that can be obtained for single machine models not only provide insights into the single machine environment, they also provide a basis for heuristics that are applicable to more complicated machine environments. In practice, scheduling problems in more complicated machine environments are often decomposed into subproblems that deal with single machines. For example, a complicated machine environment with a single bottleneck may give rise to a single machine model.

1.3.2 Parallel machines based scheduling

The parallel machine scheduling is a natural extension of the single-machine structure and also the basic unit of the flexible job shop structure. It has been extensively studied due to its practical applications in various manufacturing systems (Cheng and Sin, 1990); (Baykasoğlu and Ozsoydan, 2018), such as printed circuit board manufacturing (Pearn et al., 2007), group technology cells, semiconductor manufacturing, painting and plastic industries, injection molding processes, tracking and data relay Satellite System (Rojanasoonthon et al., 2003), and recently on synchronizing loads in hub terminals (Guo et al., 2018), etc. Furthermore, it appears as a relaxation of more complex problems like the hybrid flow shop scheduling problem or the RCPSP (Resource-Constrained Project Scheduling Problem) (Chen and Powell, 1999); several methods are used to solve this problem as column generation strategies (for more details about this methods see the references (Desrosiers and Lübbecke, 2005); (Barnhart et al., 1998); (Desaulniers et al., 2005)), heuristics, linear programs, etc.

1.3.3 Maintenance policy in an industrial workshop

Definition

Maintenance is a set of operations or techniques that allow to maintain or restore an equipment to a specific state and guarantee a given service. According to the British Standard Institute (BSI) (Standard, 1984), the latter is defined as a combination of actions to restore an item to an acceptable condition. Applications and examples which reflect the growing importance of maintenance are provided by (Ben Daya et al., 2009).

Maintenance actions can usually be classified into two major categories: Planned maintenance (proactive) and unplanned maintenance (reactive) as illustrated in Figure 1.5. In this dissertation, we focused on the first category of maintenance. Indeed, Proactive maintenance policy is organized and carried out with forethought, control and the use of records to a predetermined plan while reactive maintenance refers to repairs that are done when equipment has already broken down, in order to restore the equipment to its operating condition.

However, the goal of this subsection is not to present a state of the art of maintenance policies, but only to give sufficient information for an application in the context of this thesis.

Predictive Maintenance: PrM

It is a set of activities that detect changes in the physical condition of equipment (signs of failure) in order to carry out the appropriate maintenance work for maximizing the service life of equipment without increasing the risk of failure.

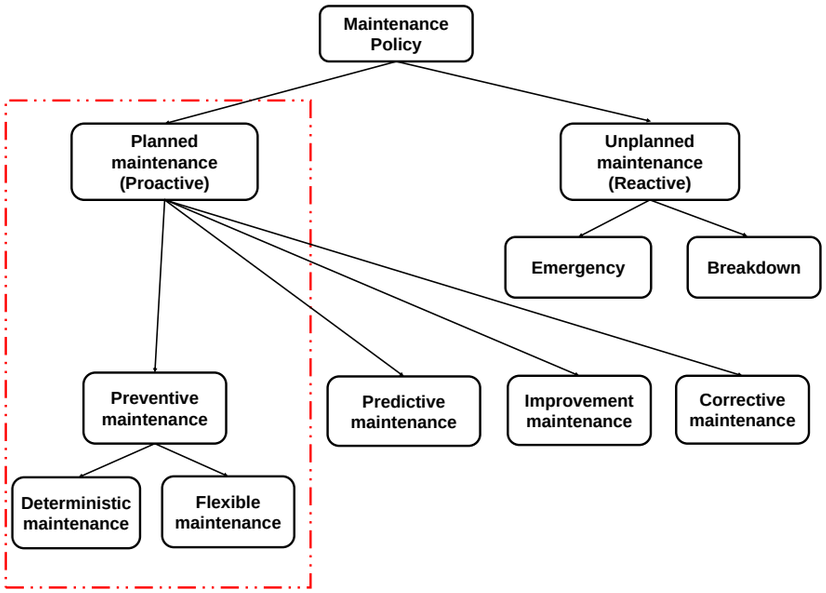

Figure 1.5: Maintenance policy

Corrective maintenance: CM

Corrective maintenance is those actions carried out when the failure has already happened. It consists of detecting, isolating and correcting a fault in order to allow the equipment, the machine or the system to return to its normal operating state. Further details of CM are available in (Banks, 1998) and (Adams, 2008).

Preventive maintenance: PM

Preventive maintenance actions are defined as systematic inspection, detection, correction, and prevention of incipient failures, before they become actual or major failures. In other words, PM is commonly conducted in order to keep a system at the desired level of operation and is performed when the system is idle at fixed time intervals.

Variety of models have been developed to determine optimal PM period, such as in (McCall, 1965) and (Pierskalla and Voelker, 1976). Therefore, several PM policies should be defined, with the aim of determining when

it is necessary to carry out PM operations on the machines according to different criteria. According to Ruiz et al. (2007), three possible policies are commonly defined:

- **Policy I**: preventive maintenance at fixed predefined time intervals,
- **Policy II**: optimum period model for the preventive maintenance maximizing the machines availability,
- **Policy III**: maintaining a minimum reliability threshold for a given production period t.

According to Ruiz et al. (2007), the widely used policy in industry is Policy I. In this latter, PM operations are planned to be performed at predefined time intervals without considering probabilistic models for the time to failure and making the best use of scheduled stops after weekly, monthly, or even annual cyclical production periods. Policies II and III utilize probabilistic models to find an optimal preventive maintenance interval. For example, Singh and Arvinderjit (2013) demonstrate that maintenance planning using probabilistic models helps to reduce downtime and increase the profits of companies.

More details about each policy II and III could be found in (Ruiz et al., 2007).

Hence, the first mentioned maintenance policies certainly influence the machine availability and the machine utilization ratio. Thereby, we can find in the literature two types of preventive maintenance as reported by Yang et al. (2011):

Deterministic maintenance: It means that maintenance periods are determined before jobs are scheduled, i.e., the starting times of the maintenance activities are fixed in advance. Scheduling dealing with this type of maintenance is often referred in literature as *scheduling with availability constraints*, *scheduling with limited machine availability* or *scheduling with fixed non-availability intervals* because during maintenance periods the machine is unavailable for processing any job.

Flexible maintenance: It means that the schedule of maintenance periods are determined jointly with the schedule of jobs, i.e., the starting times of the maintenance activities are allowed to be flexible and are determined in the scheduling process. This problem is known as *scheduling with flexible maintenance*, *scheduling with unfixed availability constraints*, *integrating preventive maintenance planning and production scheduling*, or (simultaneously/jointly) *scheduling jobs and maintenance activities*.

However, in both cases of flexible maintenance and deterministic maintenance policy, in many production systems, periodic inspection, periodic repair and preventive maintenance are usually conducted in the shops. These maintenance works are usually scheduled periodically in the production systems, we then speak about "periodic maintenance". With proper planning of periodic maintenance, the shop can concentrate on production efficiency and safety, resulting in increasing productivity and achieving high safety awareness as stated in (Arts et al., 1998) and (Liao and Chen, 2003).

To the best of our knowledge, Liao and Chen (2003) are the first group of researchers that take periodic maintenance into consideration in the scheduling community.

Furthermore, we point out that the objective in this thesis is not to find the best time (period to perform a preventive maintenance as defined by policy II and III) but to find the best schedule of jobs taking into

consideration the preventive maintenance policy. In other words, PM is considered as an unavailability constraint for the machines or equipment during the scheduling horizon that we should take into consideration. This matches with Policy I. Indeed, throughout this dissertation, we consider both the deterministic maintenance and flexible maintenance with periodic aspect in Chapters 3 and 4 respectively Chapter 5.

1.3.4 Sequence dependent setup-times constraint

Setup-times definition:

A setup is a set of operations that should be performed before or after processing a job on a machine to prepare it for processing the next job. It is also defined as a non-productive periods needed at machines in order to make cleanings, configurations or operations for the production between jobs. The interest in scheduling problems where setup times are explicitly considered began in the mid-1960s. The importance of considering the setup times in production scheduling can not be underestimated as pointed out by Allahverdi (2015). Hence, ignoring setup times may be valid for some applications, but it adversely affects the solution quality of some other applications of scheduling. This is because setup process is not a value added factor, and hence, setup times need to be explicitly considered while scheduling decisions are made in order to increase productivity, eliminate waste, and improve resource utilization.

There exists three major variants of setup-times in the related literature:

- **Independent setup-times** which means that the setup time doesn't depend on any sequence and appears the same between all the scheduled jobs.
- **Sequence dependent setup-times** which means that the setup time depends on the job that is processed immediately before. This is frequently observed in real industry settings as in the textile industry, manufacturing printed circuit boards and chemical industry (Chen, 2009; Gravel et al., 2000).
- **Job families setup times:** According to Pinedo (2016), the jobs belong in this case to F different job families. Jobs from the same family may have different processing times, but they can be processed on a machine one after another without requiring any setup in between. However, if the machine switches over from one family to another then a setup is required.

However, in this dissertation we consider the case of scheduling with sequence dependent setup-times. Due to the great savings obtained, last decades have shown an increasingly interest in studying sequencing problems where the setup-times are explicitly incorporated in scheduling decisions. A recent review is provided in (Allahverdi, 2015).

Many works have assumed that there are no setup-times or they are independent of job sequence, however, this situation may not always be true in practice. In various real world industrial/service environments such as chemical, printing, pharmaceutical, and automobile manufacturing, the setup operations, such as cleaning up or changing tools, are not only often required between jobs but they are also strongly dependent on the immediately preceding process on the same machine (Lee and Pinedo, 1997). The motivation behind this assumption is to obtain good savings when setup times are explicitly included in scheduling decisions.

Case study

- Baykasoğlu and Ozsoydan (2018) reported a case study in Heat Treatment Stage (HTS). Indeed, consecutive jobs, processed in HTS, might require different temperature levels. For this reason, a heat treatment furnace needs to be heated up or cooled down between these consecutive jobs. However, heating up and cooling down operations of the furnaces differ considerably. Heating the furnaces up from 400°C to 600°C is not identical to cooling them down from 600°C to 400°C. Thus, assuming that $i \neq j$, elapsed time while preparing a furnace from job i to job j is not equal to the elapsed time between jobs j and i. In the present problem, those elapsed times correspond to sequence-dependent setup times and setup only occurs while heating up or cooling down period of a heat treatment furnace, which means that a setup can be initialized whether the forthcoming job is released or not. This reveals an interesting practical situation.

- Chen (2009) reported a real-life scheduling problem in a textile company. The company specializes in the manufacturing textiles such as polypropylene, synthetic silk, and chemical chip. As the machine overload in 24 hours production, periodic maintenance is usually arranged in a planned schedule to avoid any sudden machine breakdowns which occur frequently in the shop. The production system of this company consists of an expensive main machine and some peripheral equipment, such as extractor, distribution pipe, spinning pump, quenching pipe, oil disk, idle roller, capstan roller, gear reel. The processing time of a job can be determined completely by the main machine. Thereby the problem can be treated as a single-machine scheduling problem. The considered workshop is running three shifts a day, 8 hours a shift, and 7 days a week for a fully utilization of the main machine and equipment already mentioned above. Each maintenance period is scheduled after a 20-day which is equivalent to 480 hours interval. The amount of time to perform one maintenance requires 24 h. Once maintenance is performed, the job being processed must be stopped. After the maintenance is completed, the unfinished job is resumed according to the original schedule. When switching between two radically different products, occurs more cleaning is required. Thus, a sequence-dependent setup times is considered. While only minor cleaning is involved when the switch is between two similar products, such as pure white and mixture white textile.

1.3.5 Weighted sum of completion times (WCT) criterion

Makespan of a schedule measures the time by which all the jobs in the schedule finish. However, when jobs are independent and competing for the same resource, a more natural measure of performance is the weighted sum of completion times of jobs (WCT). According to Pinedo (2016), the sum of completion times is in the literature often referred to as the flow time. The total weighted completion time is then referred to as the weighted flow time.

The pioneering work that considers this objective function was Smith (1956).

The weighted sum of completion times or average weighted completion time criterion is certainly less studied than the makespan. Nonetheless, many works dealing with this criterion are considered in the related literature. In (Leung, 2004), the authors made a survey about the approximation algorithms (see Section 2.5.2) used to

solve scheduling problems with aims of minimizing average (weighted) completion times.

WCT criterion remains interesting on the theoretical and practical level. Indeed, the weights can quantify the holding cost per unit of time of the products to transform. Thus, this criterion can represent the global holding cost. It also could represent the amount of value already added to a task or an order.

Moreover, in 1996 (Chekuri et al., 1996) studied an application to instruction scheduling in VLIW processors. In this application the weight of a job (or instruction) is derived from profile information and indicates the likelihood of the program block terminating at that job. This is a compelling application for scheduling jobs with precedence constraints on a complex machine environment to minimize weighted sum of completion times.

In addition to applications, minimizing the weighted sum of completion times presents interesting theoretical problems and has spurred work on polyhedral formulations (Schulz, 1996) as well as combinatorial methods.

1.3.6 Jobs rejection cost (JR) criterion

Machine scheduling problems have been extensively studied in the literature under the assumption that all jobs have to be processed. However, in many practical cases, the manufacturer may wish to reject the processing of some jobs in the shop. The decision to reject jobs may be due to a low machine capacity or high scheduling costs. In such a case, a rejected job may be either outsourced or not get served at all, resulting in a rejection penalty either due to outsourcing cost or loss of income and reputation. In such a framework, the scheduler must first decide which jobs will be rejected and which will be accepted. Then the set of accepted jobs has to be efficiently scheduled among the machines as to minimize a given predefined scheduling criterion. An instance for a single machine scheduling problem with jobs rejection is defined by the number of jobs to be processed, the jobs processing times and rejection penalties. A solution is defined by a partition of the jobs into a set of accepted, and a set of rejected jobs, by an allocation of the jobs to the machines of the shop, and by a job sequence of the set of allocated jobs on each machine. Hence, the quality of a solution is measured by two criteria: The First is the total rejection cost and the second is generally a scheduling criterion depending on the jobs completion times. Since scheduling with rejection is essentially a problem with two criteria, in this dissertation, the WCT criterion is also optimized as a second criterion.

1.4 Main contributions and thesis outline

This thesis is organized as follows:

Chapter 2 is a literature review based on the scientific state of the art in the related field of research, mainly all that has been applied to define the addressed issues and the resolution methods used to process them.

Chapters 3, 4 and 5 detail the main contributions relating to this thesis. More precisely, Chapter 3 proposes solutions methods based on greedy constructive heuristics, mathematical models, and a metaheuristic to solve the single machine scheduling problem with periodic maintenance in order to minimize the weighted sum of completion times criterion. Also, three properties for this problem which generalize already existing works are presented and applied to construct the proposed heuristics.

To evaluate the performances of the proposed methods, several lower bounds are provided.

In Chapter 4, we propose two mathematical models, two heuristics based on a multistart strategy to solve the single machine scheduling problem with sequence dependent setup time. As already done in Chapter 3, the considered criterion to minimize is the weighted sum of completion times.

In Chapter 5 we solved a two parallel machine scheduling problem with periodic maintenance, jobs rejection and weighted sum of completion times using a greedy heuristic and two different methods based on a Tabu Search algorithm. A mathematical model is also proposed for solving small-sized instances.

Finally Chapter 6 is a discussion over the study and it also presents some of our prospects.

Chapter 2

State of the art

In the presented work, we favored the use of an exact method whenever the time required to find an optimal solution was considered as "acceptable" by the decision-maker. However, to deal with large-sized instances of the target problems, our works led us to use heuristics and/or metaheuristics methods to get one or more high quality solution(s) in reasonable time. The objective of this chapter is to introduce definitions and notations of scheduling theory and maintenance, to define combinatorial optimization tool and to present literature review about works dealing with joint optimization of scheduling and maintenance problems.

Contents

2.1 Shop scheduling

First researches on scheduling appeared in the 50's. From then on, problems became more and more complex because of the numerous practical constraints to take into account...

Indeed, there are many variants of scheduling problems amongst the huge literature of this field. As mentioned in (Graham et al., 1979); (Carlier and Chrétienne, 1988); (GOThA, 1993); (Shmoys et al., 1994); (Chrétienne et al., 1995); (Pinedo, 1995); (Esquirol and Lopez, 1999); (Pinedo and Chao, 1999); (Allahverdi et al., 1999), (Billaut and T´Kindt, 2002); (Billaut et al., 2005); (Baptiste et al., 2005); (Artiba et al., 2011); (Jarboui et al., 2013), scheduling problems are present in all sectors of the economy: industry, hospitals, administrations, computer science... Scheduling is a crucial issue because the quality of generated schedules directly affects the production (in terms of efficiency, competitiveness, flexibility, deadlines fulfillment, customer satisfaction...).

2.1.1 Definitions

By definition, *scheduling* is "to forecast the processing of a work by assigning resources to tasks and fixing their start times" (Billaut and T´Kindt, 2002; Carlier and Chrétienne, 1988). A *schedule* is a solution to a scheduling problems. It consists of a set of start dates, and assigned resources (i.e. machines, operators, etc.) to each operation. Among the family of scheduling problems, our works focused on *shop scheduling*. A simplified definition of shop scheduling problems, largely inspired by (Billaut et al., 2005), is the following: a shop scheduling problem corresponds to a model consisting of a set I of m different machines on which to run a set J of n jobs. Each job J_j is described by n_j tasks (named operations). The j^{th} operation of job J_j that has to be performed on machine M_i is denoted by O_{ij}. The operation of a same job cannot be performed simultaneously. We distinguish two types of task:

- **Resumable job**: a job is resumable if it cannot be finished before the unavailable period of a machine and can continue after the machine is available again.
- **Non-resumable job**: a job is nonresumable if it has to restart, rather than continue after the machine becomes available.

The following notations are associated with each job j:

Processing time (p_{ij})**:** p_{ij} is the processing time of job j on machine i. The index i is omitted if the processing time of job j does not depend on the machine or if job j is only to be processed on one given machine. Thus, the processing time will be denoted by p_j instead of p_{ij}.

Release date (r_j)**:** The release date r_j of job j is the time the job arrives at the system, i.e., the earliest time at which job j can start its processing.

Due date (d_j)**:** The due date d_j of job j represents the committed shipping or completion date (i.e., the date the job is promised to the customer). Completion of a job after its due date is allowed, but then a penalty

is incurred.

Deadline ($\tilde{d}_j$)**:** When completion of a job j after its due date is forbidden, it is referred to as a deadline and denoted by $\tilde{d}_j$.

Weight (w_j)**:** The weight w_j of job j is basically a priority factor, denoting the importance of job j relatively to the other jobs in the system.

Rejection cost (u_j)**:** The cost u_j is associated to each job j and represents the expense related to the decision of rejecting(i.e not scheduling) that job and not schedule it.

Here we give some standard notations used throughout this thesis:

- PM: Refers to a preventive maintenance action,
- T: Period between two consecutive PM,
- T_i: Each machine M_i must undergo a PM at intervals that cannot exceed T_i time units,
- δ_i: Maintenance duration in machine i. When only one machine is considered, δ_i becomes δ,
- Batch: Refers to the set of jobs that can be scheduled in each period of time T,
- Block: Refers to the set of jobs actually scheduled between two consecutive preventive maintenances.

In this thesis, the difference between the blocks and the batches is that a batch has always the same capacity T while a block does not necessarily have the same size.

- S: current solution of a predefined problem (P),
- S^*: Best solution found by an algorithm for (P),
- $S^{'}$: Neighbor solution of S.

2.1.2 Shop scheduling description

Scheduling problems are described by a triplet $\alpha|\beta|\gamma$. The field α describes the *machine environment* and contains just one entry. The field β provides details of processing *characteristics and constraints* and may contain no entry at all, a single entry, or multiple entries. The γ field describes the *objective to be minimized* and often contains a single entry. This notation is proposed in (Graham et al., 1979) and extended by Billaut and T´Kindt (2002).

Machine environment

Single machine(1): The case of a single machine is the simplest of all possible machine environments and is a special case of all other more complicated machine environments.

Parallel machines(Pm): There are m identical machines in parallel. Job j requires a single operation and may be processed on any one of the m machines or on any one that belongs to a given subset. If job j cannot be processed on just any machine, but only on any one belonging to a specific subset M_j, then the entry M_j appears in the β field. There are three field kinds of parallel machines:

- Identical parallel machines (P): the operating times of the tasks do not depend on machines, i.e. the speed of execution of a task is the same on all machines.
- Uniform parallel machines (Q) : each machine has a clean and constant execution speed.
- Non-uniform parallel machines (R): the execution speed is different for each machine and for each task.

Dedicated machines: In this class, each job consists of several operations o_{ij} and the machines are specialized in carrying out certain operations. Three types of workshops are considered :

- **Flow shop (F):** in this workshop the sequencing tasks path is unique, the tasks to be realized go through all the machines in the same order. In the literature, particular case is often studied, for which the sequence of tasks going throw every machine is the same for all machines.
- **Job shop (J):** this workshop is considered as a generalization of the flow shop. Indeed, the operating range is different from one operation to another. In other words, the order of operations on machines can be different from one task to another.
- **Open shop (O):** this workshop is the least studied shop in the literature and the least used in the enterprises. The order of execution of the operations is not fixed.

Constraint characteristics

The set of constraints and characteristics can be various, so we will introduce just the ones used in this thesis:

pmtn: This means that the job can be preempted, so it can be started on one machine, stopped, and resumed in a later time, also on a different machine. When preemptions are allowed *pmtn* is included in the β field, when *pmtn* is not included, preemptions are not allowed.

r_j: This indicates the presence of some release dates which means that for $r_j = x \geq 0$ the job j can be started only after x units of time from the origin of time.

$\mathbf{St_{sd}}$: According to (Pinedo, 2016), the st_{jk} represents the sequence dependent setup time that is incurred between the processing of jobs j and k. s_{0k} denotes the setup time for job k if job k is first in the sequence and st_{j0} the clean-up time after job j if job j is last in the sequence (of course, st_{0k} and st_{j0} may be zero). If the setup time between jobs j and k depends on the machine, then the subscript i is included, i.e., s_{ijk}. If no st_{jk} appears in the β field, all setup times are assumed to be 0 or sequence independent, in which case they are simply included in the processing times.

PM: This notation refers to the preventive maintenance policy.

In problems with unavailability periods (due to preventive maintenance or not), that latter are considered as constraints related to the environment of execution. This notation has been presented in detail in the surveys on the problems with known unavailability by (Schmidt, 2000) and (Ma et al., 2010). The followed notations are then introduced.

$\mathbf{h_{l,k}}$: indicates that the processed scheduling problem has unknown number of unavailability periods on each machine k

$\mathbf{h_1}$: Indicates that in one single machine environment, only one unavailability period is considered.

$\mathbf{h_k}$: Indicates that k unavailability periods are considered.

Any other entry that may appear in the β field is self explanatory. For example, $p_j = p$ implies that all processing times are equal.

Optimization criterion

Examples of possible objective functions to be minimized are:

Makespan (Cmax): The completion time of job j is denoted by c_j. The, the makespan is defined as $max(c_1, ..., c_n)$ which is equivalent to the completion time of the last job in the schedule. A minimum makespan usually implies a good utilization of the machine(s).

Maximum Lateness (Lmax): The maximum lateness, Lmax, is defined as $\max(L_1, ..., L_n)$, with L_j is the lateness of a job j computed as follows:

$$L_j = c_j - d_j \tag{2.1}$$

The maximum lateness measures the worst violation of the due dates.

Total weighted completion time ($w_j c_j$): This criterion is already introduced in Chapter 1.

Total weighted tardiness ($\sum_{j=1}^{n} w_j T_j$): This criterion is a more general cost function than the total weighted completion time. We define by T_j the tardiness of a job which is computed as follows:

$$T_j = max(c_j - d_j, 0) = max(L_j, 0) \tag{2.2}$$

Total rejection cost ($\sum_{j=1}^{n} u_j$): This criterion is already defined in Chapter 1.

Other criteria are defined in Pinedo (2016).

In this thesis, we focus on two types of shop scheduling problems: the single machine and the parallel machine environments.

Hence, the addressed scheduling problems in Chapters 3, 4 and 5 are denoted respectively as follows: $\mathbf{1/PM/\sum_{i=1}^{n} w_j c_j}$, $\mathbf{1/PM, St_{sd}/\sum_{i=1}^{n} w_j c_j}$ and $\mathbf{P2/PM, \tilde{d}_j = \tilde{d}/\sum_{i=1}^{n} u_j, \sum_{i=1}^{n} w_j c_j}$.

2.1.3 Scheduling representation

Each schedule can be represented by a Gantt chart, which has been developed by H.-L. Gantt in 1910. The Gantt chart is also used in companies for project planning. In the case of production management, this diagram consists of several horizontal lines (or floors) that represent the machines. Tasks performed on a given machine are represented by rectangles of length proportional to their duration. Therefore, the Gantt chart gives an overview of the tasks performed on all machines over time.

Figure 2.1 shows a Gantt chart for a three parallel machine shop. The processing times of each job are summarized in Table 2.1.

jobs	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9
p_j	4	2.5	1.5	3	3	3	1	1	2

Table 2.1: Jobs processing times

2.2 Scheduling problems under unavailability constraint due to a preventive maintenance policy

Production scheduling and preventive maintenance planning are the most common and significant problems faced by the manufacturing industry. Scheduling the maintenance in manufacturing systems has gradually become a common practice in many companies. Indeed, according to practical experience, it sometimes can be found that some of the machines are awaiting maintenance while there are jobs waiting to be processed by these machines. This is due to the lack of coordination between maintenance planning and production scheduling. Therefore, there is a need to develop efficient scheduling methods to improve the situation by deriving a satisfactory schedule that considers both jobs and maintenance activities simultaneously.

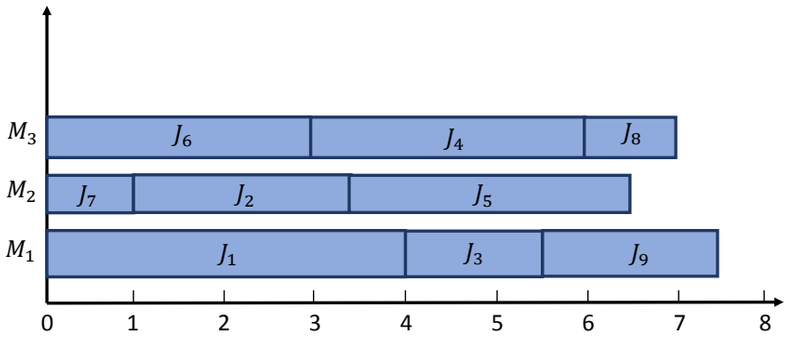

Figure 2.1: Gantt chart

With proper planning of the maintenance activities, the shop can improve production efficiency and safety, resulting in increased productivity and heightened safety awareness (Arts et al., 1998). The scheduling of maintenance activities can be determined either before the scheduling of jobs, or jointly with the scheduling of jobs. In the first case, the maintenance periods are known and fixed in advance. Thus the problem of scheduling jobs with this kind of maintenance reduces to the problem often referred in the literature as scheduling with machine availability constraints.

Whereas scheduling tends to efficiently assign resources to jobs and generally suppose that all resources are always available when no jobs are processed.

The two activities are in conflict because, as we know, maintenance operations consume production time while delaying maintenance operations can increase the probability of machine failure.

Maintenance operations need to access to resource so as to maximize their efficiency while trying to reduce failure risks. Indeed, maintenance problems are crucial aspect of nowadays industrial problems. However, the quest of the efficient periodicity of maintenance for all components of a system is far from an easy task to accomplish when considering all the antagonistic criteria of the maintenance and production views of a production system. It is expected that production will be more efficient and revenues will increase if preventive maintenance is well managed.

In this thesis, we study the coupling between production and maintenance (Roux et al., 2009); (Roux et al., 2013); (Benmansour et al., 2011); (Benmansour et al., 2012); (Benmansour et al., 2014); (Rivera Gómez et al., 2013); (Pozzetti et al., 2014). Thus, the objective is to simultaneously ensure a low frequency of failures by an efficient periodic preventive maintenance and minimize the unavailability of the system due to preventive maintenance. This implies a minimum impact on the production through an adequate maintenance strategy that takes production tasks into account.

A *maintenance strategy* is defined as a decision rule which establishes the sequel of maintenance actions. Each maintenance action allows one to maintain or restore the system in a specified state by using the appropriate resources. Cost and duration are incurred to execute each maintenance action. Many papers and books dealing

with preventive maintenance and replacement strategies have been published previously (Ben Daya et al., 2009); (Boschian et al., 2009); (Wang, 2002). In this thesis, we consider one basic replacement policy (Bloc Replacement Policy BRP) and assume that the maintenance costs are negligible.

As in (Barlow and Proshan, 1976), we consider in Chapters 3 and 4 the BRP where the replacements are undertaken at exactly KT with $K = 1, 2, \ldots, T$ a fixed time, or at failures. while in Chapter 5, we consider a flexible preventive maintenance where the replacement takes place before each T_i time of use of the machine M_i. In both cases, items used in each replacement are supposed to be as good as new ones.

However, scheduling problems are often classified among the most difficult combinatorial problems. Thereby, using the exact methods to solve them is practically unusable and are to be banished in aim of finding optimal solutions in a reasonable time for large sized instances, especially in a just in time environment. Consequently, it becomes necessary to use approximate methods to solve them.

2.3 Scheduling problems complexity

The theory of complexity classifies problems according to their difficulties. In general, the computational time of an algorithm obviously depends on the machine on which it is running. However, if we count the number of steps in this algorithm, it is possible to ignore such environmental constraints. Thus, the time complexity function $T(n)$ of an algorithm expresses the maximum time, in the sense of the number of steps, necessary to solve an instance of size n of a predefined problem.

However, the complexity of an algorithm can refer to the memory space that is needed when running the algorithm. Hence, in this dissertation we will only refers to the time complexity and then use the term of complexity instead of time complexity.

Two types of problems are often studied. Decision problems and search problems.

Decision problems are those for which we try to answer a question by *yes* or *no*, and when we treat a research problems then, the goal is to find a solution to a given problem.

Optimization problems form a subclass of search problems. In this case, the solution must optimize an objective function.

In scheduling, we encounter both problems of decisions and optimization. Decision problems are mostly used to solve the optimization problems associated with it, or to help determining its complexity class. In general, scheduling problems are classified as difficult combinatorial problems which means that there are no universal optimal algorithms to solve all cases.

Depending on its complexity, a problem can belong to one of the following four classes (Garey and Johnson, 1979):

- **P problems class:** Called polynomials, P is the complexity class containing decision problems which can be solved by a deterministic *Turing machine* using a polynomial amount of computation time, or polynomial time. P is often taken to be the class of computational problems which are *efficiently solvable* or *tractable*. Problems that are solvable in theory, but cannot be solved in practice, are called intractable.

- **NP problems class:** NP (Non deterministic Polynomial time) is the set of decision problems solvable in polynomial time on a non deterministic Turing machine (for more details please refers to (Bovet et al., 1994); (Vitanyi and Li, 1997)). Called NP-hard problems. The solutions obtained in a reasonable time can never be qualified as optimal. All the problems in this class have the property that their solutions can be checked effectively.

- **NP − Complete problems class**: $NP - complete$ problems class is a sub-class of the NP problems. In complexity theory, the $NP - complete$ problems are the most difficult problems in NP in the sense that they are the ones most likely not to be in P.

  Moreover, NP-Complete class represents the set of all problems X in NP for which it is possible to reduce (for more details see (Vitanyi and Li, 1997)) any other NP problem Y to X in polynomial time.

- **NP − hard problems class:** These are the problems that are at least as hard as the NP-complete problems but are not necessarily in NP and do not have to be decision problems.

  More precisely, a problem X is $NP - hard$, if there is an $NP - complete$ problem Y, such that Y is reducible to X in polynomial time. But since any $NP - complete$ problem can be reduced to any other $NP-complete$ problem in polynomial time, all $NP-complete$ problems can be reduced to any $NP-hard$ problem in polynomial time. Then, if there is a solution to one $NP - hard$ problem in polynomial time, there is a solution to all NP problems in polynomial time.

2.4 Combinatorial optimization

Nowadays, many real-world problems are complex and difficult to solve. We can enumerate problems in production systems, constraints management and routing problems in transport systems, image processing and air quality management in the environmental sector. When it becomes possible to model all these problems using mathematics, we give them the denomination of optimization problems. The goal is usually to optimize one or more criteria taking into account several constraints.

When enumerating all feasible solutions of research space of a given problem grows rapidly and becomes exponential, we speak then about combinatorial explosion of the search space. It becomes impossible to find optimal solutions to the considered problem in a reasonable time. More precisely, when the decision variables of the problem are discrete, these kind of problems are called combinatorial optimization (Papadimitriou and Steiglitz, 1998) problems. These latter are therefore divided into two categories: those solved by efficient algorithms (with polynomial complexity) which allow to find optimal solutions in a reasonable time, and those whose resolution can take an exponential time for the large size instances.

Hence, the main objectives of many researchers is to formulate a combinatorial optimization problem and try to solve it. For that, the procedure consists by first defining the parameters, the variables, the research space and the objectives to be optimized. Then, model them and at the end choose a method of resolution. at this level, a parameter comes into account: The complexity of the problem. This latter allows to decide

between using exact methods or an approximate method. Before presenting this range of methods, let us briefly introduce combinatorial optimization.

2.4.1 Mathematical definition of a combinatorial optimization problem (COP)

The mathematical definition of a COP is generally formulated as follows: $min_{S \in E} f(s)$ with f defines a mathematical function with real values. It is the objective function that evaluates the solutions of E pointed out by S. Without loss of generality, we consider here a problem of minimization. That is, a maximization problem amounts to minimizing $-f$.

The deal is the minimize f of all the feasible solutions belonging to the search space E also called set of feasible solutions. Then, solving a COP amounts to find an optimal solution $S^* \in E$ for which the objective function f will return a minimal value, that is to say $f(S^*) \leq f(S)$, $\forall S \in E$. Then, S^* is called a global optimum of COP.

Before going farther in the explanation, let us introduce three fundamental notions:

Neighborhood operator: Let E be the feasible solution set. An operator or a move Δ is defined as transformation in a solution S. Mathematically, Δ is a function of E in itself ($\Delta \in E^2$) which allows to go from a solution S to a neighbor solution $S^{'}$.

Neighborhood structure: The set of transformations that can be applied to the current solution S define a set of neighboring solutions in the search space that we denote by $\mathscr{N}(S)$. Formally, $\mathscr{N}(S)$ is a subset of the search space E defined by: $\mathscr{N}(S) = \{$ solutions obtained by applying an operator (move) to $S\}$. The size of a neighborhood corresponds to the number of solutions it contains, $|\mathscr{N}(S)|$.

Local optimum: It is a solution for which a neighborhood does not contain any candidate solution for which the objective function value is better. In a context of minimization, a solution $S^* \in E$ is a local optimum if $\forall S \in \mathscr{N}(S^*), f(S^*) \leq f(S)$.

The solutions S can be a mathematical object of diverse nature, often S represents a vector with n dimensions when f designates a function in R^n. The set of feasible solutions E is generally determined by constraints often expressed as inequalities.

Figure 2.2 inspired from the work of (Veerapen, 2012) illustrates an optimization problem to make the difference between local and global optima.

Note that when faced with a COP, we must:

- Develop a mathematical model: Expression of constraints to respect and objectives to optimize,
- Develop a resolution algorithm,
- Evaluate the quality of the solutions found, with benchmarks and / or performance indicators like lower bounds.

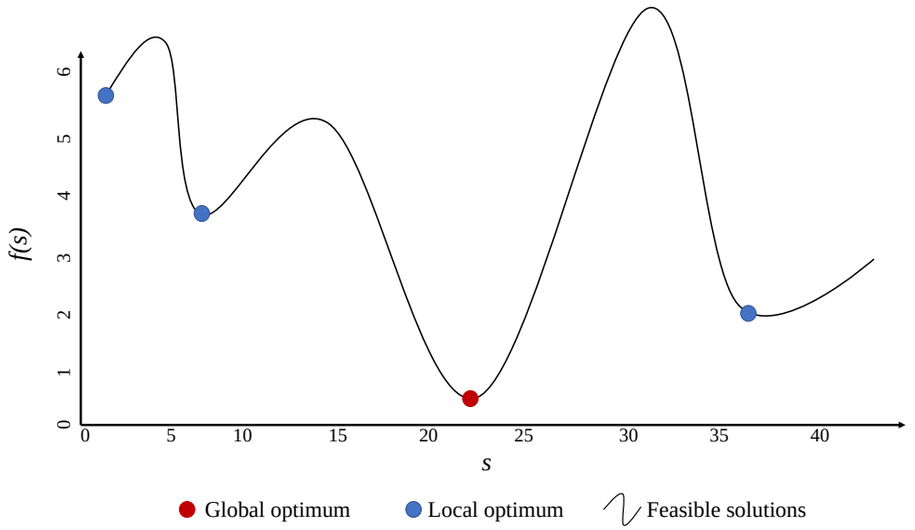

Figure 2.2: A simple optimization problem (minimization) illustrating local and global optima.

A COP is characterized as cited above by the number of objectives to be optimized. We will therefore talk about mono-objective optimization problems and multi-objective optimization problems.

2.4.2 Multiobjective Optimization (MO)

Multiobjective optimization is a very important research area because of the multiobjective nature of the most real problems. First works on multiobjective problems area were carried out in the 19th century in economics studies by *Edgeworth* and generalized later by *Pareto*. A survey for multicriteria scheduling problems is done by T'kindt and Billaut (2001).

Contrary to mono-objective optimization, which are treated in two problems described in Chapters 3 and 4, the goal of solving a multiobjective problem is to simultaneously optimize several objective functions, composed of several decision variables, to satisfy a specific problem. In most cases, these objectives are in conflict: The improvement of one objective causes the deterioration of another. Consequently, the optimization final result is no longer given by a single solution but rather by a set of solutions. We speak then of a multiplicity of solutions that represents a compromise between the different objectives. In chapter 5, we have studied a two-criteria scheduling problem (bi-objectives). Lexicographic optimization is used for the resolution.

2.4.3 Lexicographic Optimization (LO)

Lexicographic optimization approach establishes a hierarchical order among all the optimization objectives instead of giving them a specific weight. With this approach, the decision maker have to formulate preferences in order to establish a predefined ranking between the competing objectives (formally, $f_1 > f_2 > ...f_k$)

Consequently, two methods are considered. The first one consists in using a multi-stage sorting algorithm, comparing the solutions on the basis of f_1, then on the basis of f_2 in case of identical values of f_1 then on the

basis of f_3 in case of identical values of f_1 and f_2, and so on...

The second one is based on a specific weighted sum (see Duvivier (2015) for additional details) which computes $f = w_1 * f_1 + w_2 * f_2 + \cdots + w_k * f_k$, so that sorting solutions on the basis of f will result in the same result provided by a multi-stage sorting algorithm.

However, the goal of this subsubsection is not to present a state of the art of lexicographic methods, but only to give sufficient information for an application in the context of this thesis. Additional answers concerning "lexicographic optimization" are available in the book of Collette and Siarry (2002).

This strategy is adapted to solve the tackled problem in Chapter 5.

2.5 Combinatorial optimization methods

It exists a large number of methods to solve optimization problems. Figure 2.3 inspired from the work of (Talbi, 2009) and (Reeves, 1993) suggests a possible hierarchy of these methods.

Following Figure 2.3, optimization methods could be separated into two main families : exact methods and approximate methods. The methods of the first family ensure to find an optimal solution for the considered instance whereas the methods of the second family do not give any guarantee on the optimality of the solution found.

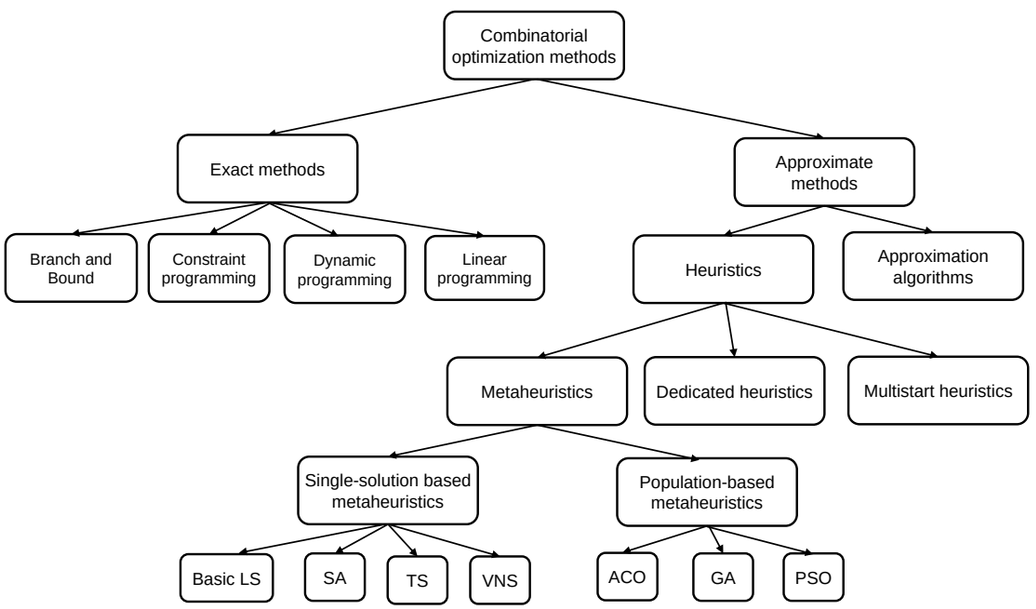

Figure 2.3: A possible hierarchy of a subset of combinatorial optimization methods

2.5.1 Exact methods

There are several families of exact methods: In this thesis, we briefly introduce dynamic programming, branch and bound and the linear programming. These methods take an exponential computational time to reach optimal solutions, this explains why they are used only for small sized instances of target problems.

Mathematical Programming: Linear programming (LP)

It is the most basic mathematical program denoted by LP and refers to an optimization problem in which the objective and the constraints are linear in the decision variables. An LP can be formulated as follows:

$$min\ f = \quad \sum_{j=1}^{n} c_j x_j \tag{2.3}$$

$$subject\,to \quad a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n \leq b_1 \tag{2.4}$$

$$a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n \leq b_2 \tag{2.5}$$

$$a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n \leq b_3 \tag{2.6}$$

$$\vdots \tag{2.7}$$

$$a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n \leq b_m \tag{2.8}$$

$$x_j \geq 0 \quad \forall j = 1, ..., n \tag{2.9}$$

The objective is the minimization of costs. The c_1,...,c_n vector usually refers to the cost vector. The notations x_1,...,x_n refers to decision variables that should be determined in such a way that the objective function is minimized. The column vector a_{1j},...,a_{mj} is referred to as an activity vector j. The value of the variable x_j refers to the level at which this activity j is utilized. The $b_1, ..., b_m$ is usually referred to as the resources vector. Hence, n refers to the number of decision variables and m to the number of constraints, then $n \times m$ denotes the size of the defined problem LP.

However, in scheduling theory n usually refers to the number of jobs; and m denotes the number of resources (machines). Obviously, in linear programming this has nothing to do with the m refers to the number of machines in scheduling theory.

There are several algorithms or classes of algorithms for dealing with an LP. The most important ones is the *simplex methods* (Sierksma and Zwols, 2015). Although simplex methods work very well in practice, until today, there is no known version that solves the LP problem in polynomial time.

A special case of the linear program is the so-called *Integer Program* (IP). IP is basically a linear program with the additional requirement that the variables x_1,...,x_n are integers.

When only a subset of the variables are required to be integer and the remaining ones are allowed to be real, the problem is referred to as a *Mixed Integer Linear Program* denoted by MILP. Contrast to the LP, an efficient (polynomial time) algorithm for the IP or MILP does not exist.

Dynamic programming method (DP)

Dynamic programming method was introduced by Bellman et al. (1954), (Bellman, 1966). It is a method for efficiently solving a broad range of optimization problems which exhibits the characteristics of overlapping sub-problems and optimal sub-structures. In one hand, overlapping sub-problems means that the problem can be broken down into sub-problems which are reused multiple times. More precisely, when solving the incumbent treated sub-problem, dynamic programming will try to use the results of more than one already treated sub-problems. In the other hand, optimal sub-structures means that, if the problem was broken up into a series of sub-problems and the optimal solution for each sub-problem was found, then the resulting solution would be realized through the solution to these sub-problems. A problem that does not have this kind of structure cannot be solved with DP. More detailed informations are available in Cormen et al. (2009).

Branch and Bound methods: (B & B)

This method is based on an implicit and intelligent enumeration of all feasible solutions of the domain of research. The "Separation" step consists of breaking down the set of solutions into several sub-solutions which are then decomposed according to an iterative process. This process can be visualized as an enumeration tree, the nodes of the tree correspond to the subsets and the leaves correspond to feasible solutions. To accelerate the search for optimal solutions by avoiding the unnecessary exploration of certain nodes, we use an "evaluation" procedure which allows to compute at each level of the tree, the value of each node and thus makes it possible to decide on the node to develop. For more details about this method please refer to Lee et al. (1970); Brusco and Stahl (2006); Scholz (2011)

2.5.2 Approximate methods

When it becomes necessary to obtain solutions, not necessarily optimal, for large sized problems, in limited time, it becomes necessary to use the so-called approximated methods:

As illustrated in Figure 2.3, approximate methods could be divided in two sub-families: Heuristics and approximation algorithms. Approximation algorithms allow to provide a guarantee using a "ratio" which states how close to optimality is the obtained solution. On the other hand, heuristics does not guaranty any information in this regard.

Approximation algorithms

In operations research field, approximation algorithms are efficient algorithms that find approximate solutions to NP-hard optimization problems with provable guarantees on the distance of the returned solution to the optimal one. Approximation algorithms mainly arise in the field of theoretical computer science as a consequence of the widely believed $P \neq NP$ conjecture. Under this conjecture, a wide class of optimization problems cannot be solved exactly in polynomial time. The field of approximation algorithms, therefore, tries to understand how closely it is possible to approximate optimal solutions to such problems in polynomial time. In the majority of the cases, the guarantee of such algorithms is a multiplicative one expressed as an approximation ratio or

approximation factor i.e., the optimal solution is always guaranteed to be within a (predetermined) multiplicative factor of the returned solution.

However, there are also many approximation algorithms that provide an additive guarantee on the quality of the returned solution. A notable example of an approximation algorithm that provides both is the classic approximation algorithm done by Lenstra et al. (1990) for unrelated parallel machines scheduling problems.

The design and analysis of approximation algorithms crucially involves a mathematical proof certifying the quality of the returned solutions in the worst case (Williamson and Shmoys, 2011). This distinguishes them from heuristics and metaheuristic algorithms, which find reasonably good solutions on some inputs, but provide no clear indication on when they may succeed or fail.

Single machine scheduling problem with availability constraints that are solved using approximation algorithm are investigated in (Kacem, 2008); (Kacem and Mahjoub, 2009); (Kacem, 2009); (Breit, 2007); (Sadfi et al., 2005).

Heuristics

Regarding to Reeves (1993), a heuristic is a method which seeks good (i.e. near-optimal) solutions at a reasonable computational cost without being able to guarantee optimality, and possibly not feasibility. Unfortunately, it may not even be possible to state how close to optimality a particular heuristic solution is.

Heuristics have been used since the beginnings of operations research to tackle difficult combinatorial problems. With the development of complexity theory in the early 1970s, it became clear that, since most of these problems were proved to be NP-hard, there was a little hope to find efficient exact solution procedures for these problems. This emphasized the role of heuristics for solving the combinatorial problems that were encountered in real life applications and that needed to be tackled, whether or not they were NP-hard.

These methods have the advantage of only covering a fraction of the search space to reach an acceptable solution. Consequently, the computational time of these methods are most of the time polynomial. This explains why they are generally used for large-sized problems.

Heuristics could be divided into metaheuristics, which are generic methods not dedicated to a particular problem and, dedicated heuristics, which concern specific methods to a given problem, as their name indicates it.

Metaheuristics

Based on Talbi (2009) definition, metaheuristic search methods are upper level heuristic which can be defined as a general methodologies (templates) that can be used as guiding strategies in designing underlying heuristics to solve specific optimization problems.

Metaheuristics are not dedicated to a given problem, they are flexible and can be adapted to any problem. These methods are in most cases derived from mechanisms found in nature. Until today there is no exact definition of these methods. Jonson defines them as a set of generic strategies that form a set of procedures that can find good solutions quickly. Glover (1986) was the first to name these methods by *metaheuristics*.

According to Gendreau and Potvin (2010), combinatorial optimization was shattered by the appearance of the work of Kirkpatrick et al. (1983) where it was shown that a new heuristic approach called Simulated Annealing (SA) could converge to an optimal solution of a combinatorial problem, but in infinite computing time. Based on an analogy with physical mechanics, SA can be interpreted as a form of controlled random walk in the space of feasible solutions. The emergence of SA indicated that it is possible to look for other ways to tackle combinatorial optimization problems and arouse the interest of the research community. In the following years, many other new approaches were proposed, mostly based on analogies with natural phenomena (like Tabu Search, Ant Colony Optimization, Particle Swarm Optimization) which with Genetic Algorithm (Holland, 1992) gained an increasing popularity. Nowadays, these methods have become over the last 20 years the leading edge of heuristic approaches for solving combinatorial optimization problems. These methods include: Simulated Annealing (SA), Tabu Search (TS), Variable Neighborhood Search (VNS), Genetic Algorithm (GA), Memetic algorithms which combine genetic algorithms with local search methods, Ant Colony (AC) (Dorigo and Di Caro, 1999), etc.

There are several classifications of metaheuristics, such as nature-inspired versus non-nature inspired, dynamic versus static objective function, One versus various neighborhood structures or memory usage versus memory-less methods. In this thesis, we investigated the single-solution versus population-based metaheuristic classification.

Single-solution based metaheuristics: They are based on a single solution, starting by constructing an initial solution either in a random way or by using a heuristic, which is iteratively improved by choosing a new solution in its neighborhood. The most popular of them are basic local searches heuristics.

Basic Local search heuristics (LS): Local search heuristics (LS) are the basis of most metaheuristic search methods. LS is defined as an iterative search procedure that, starting from an initial feasible solution S, progressively improves it by moving to a *neighbor* solution that differs only slightly from the current one (in fact, the difference between the previous and the new solutions consists on applying local modifications mentioned above (definition of an operator). LS stops when all neighbors are inferior to the current solution. LS can find good solutions very quickly. However, it can be trapped in local optima in the search space that are better than all their neighbors, but not necessarily representing the best possible solution (a global optimum).

Figure 2.4 illustrates the iterative search procedure of LS from a neighborhood structure to another within a search space E.

In LS, the quality of the obtained solution and computing times are usually dependent on the effectiveness of the chosen transformations (operators) considered at each iteration of the heuristic. In the reminder of this thesis, let us denote by $argmin_{S' \in \mathcal{N}_{\Delta}(S)} f(S')$ the local search which returns the best solution found in the generated neighborhood of the solution S when using an operator Δ and which depends on the objective function value of each generated neighboring solution S'.

The most common search strategies used in a local search heuristic are the **first improvement strategy** and the **best improvement strategy**. The first strategy is to stops the search as soon as an improved solution

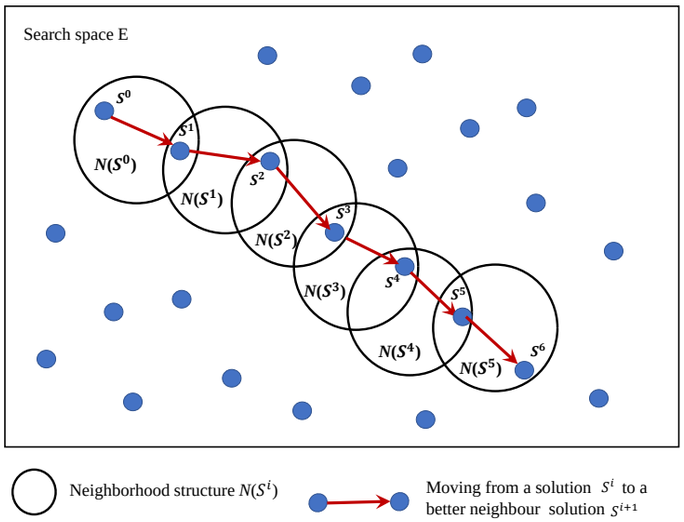

Figure 2.4: Illustration of Local search methods behavior

in a neighborhood structure $\mathscr{N}(S)$ is detected, it is defined to be the new existing solution, while for the second strategy, the best among all improving solutions is $\mathscr{N}(S)$ is set to be the new incumbent solution.

To improve the effectiveness of LS and escape from local optima, various techniques have been introduced over the years. These methods are presented in the next Section 2.5.2.

Descent search heuristic (DSH): One of the most popular local search algorithms is the descent search heuristic. Algorithm 1 describes the main steps of this heuristic. DSH stops once a local optima is reached. The name *descent* comes from the fact that implicitly a minimization problem is considered, therefore we are looking for the smallest value of the objective function. Moreover, the english term *hill-climbing* refers to the same method DSH but in a context of maximization. Furthermore, when repeating the call to DSH until a counter is reached by changing each time the initial solution and recording the best solution obtained, we then speak about a multiple DSH.

Variable Neighborhood Descent heuristic (VND): A VND is a local search heuristic which tends to explore several neighborhood structures with the aim to improve a given solution. This heuristic has been successfully applied for solving different combinatorial optimization problems (Hansen et al., 2010b); (Hansen et al., 2001); (Mjirda et al., 2017).

The search strategy within VND may be either the first improvement strategy or the best improvement search strategy. A **neighborhood change strategy** is most of the time performed within VND in a deterministic way as described in the following paragraphs:

Sequential neighborhood change strategy: Several neighborhood structures are firstly ordered in a list $\mathscr{N} = \{\mathscr{N}_1, \mathscr{N}_2, ..., \mathscr{N}_{k_{max}}\}$ and after that examined one after another respecting the established order. As soon

Algorithm 1: A basic Descent Search Heuristic

Input: $\mathcal{N}$ a neighborhood structure, f an objective function, S initial solution
Output: S^* best solution found.

$S^* \longleftarrow S$
repeat
  $S \longleftarrow argmin_{S' \in \mathcal{N}_\Delta(S)} f(S')$
  if $f(S) < f(S^*)$ **then**
    $S^* \longleftarrow S$
  end
until $(f(S) \geq f(S^*))$;

as an improvement of the current solution in some neighborhood structures occurs, the best solution found is updated, then VND proceeds to the next (in the defined order) neighborhood structure of the new current solution. The whole process is stopped if the current solution can not be improved in any of the neighborhood structures k_{max} or either if a stopping criterion is not reached.

Nested neighborhood change strategy: This strategy explores a large neighborhood structure obtained as a composition of several neighborhoods. This latter is defined by $\mathcal{N}^* = \mathcal{N}_1 \circ \mathcal{N}_2 \circ, ..., \circ \mathcal{N}_{k_{max}}$.

Mixed neighborhood change strategy: This strategy combines ideas of sequential and nested strategies. Namely, it uses a set of Neighborhood structure: $\mathcal{N} = \{\mathcal{N}_1, \mathcal{N}_2, ..., \mathcal{N}_{k_{max}}\}$ to define a nested neighborhood and, after that, on each element in this nested neighborhood applies sequential strategy. The defined neighborhood structure is defined as follows: $\mathcal{N}' = \{\mathcal{N}_1', \mathcal{N}_2', ..., \mathcal{N}_{k'_{max}}\}$.

More detailed study of the different methods and strategies used within VND is available in (Todosijević, 2015).

It should be emphasized that TS, SA and VNS metaheurisics are also defined as local searches (Reeves, 1993).

Population-based metaheuristics: Biological phenomena have been the source of these algorithms. Population-based approaches maintain and improve multiple candidate solutions, often using population characteristics to guide the search. These methods make it possible to better understand the large areas of research because of their power when using diversification mechanism defined in the following paragraphs.

Genetic algorithms, particle swarm optimization (PSO) and Ant Colony Optimization (ACO) are the most famous population-based metaheuristics.

Some mechanisms used within metaheuristics: As an approximate method, a good metaheuristic has to cleverly conjugate two generally divergent objectives during the exploration of the search space:

Intensification mechanism: It is an algorithmic mechanism which consists on the ability to obtain

increasing quality solutions by exploring deeply the portions of the search space that seem *promising* in order to make sure that the best solutions in these areas are indeed found.

Diversification mechanism: It is an algorithmic mechanism which allow to explore efficiently the research space by forcing the search into previously unexplored areas.

One of the main problems of all methods based on local search approaches, is that they tend to spend most of their time in a restricted area of the search space. In fact, the consequence is that these methods may fail to explore the most interesting parts of the search space and thus end up with solutions that are still pretty far from the optimal ones. Hence, diversification mechanism tries exactly to alleviate this problem.

Dedicated heuristics

Dedicated heuristics, constructive heuristics also called greedy algorithms are specific methods that are closely related to a given problem but stay irrelevant for other problems, this because there are usually built from information collected from the properties relative to optimal solutions of the studied problem. They are empirical methods that usually give good solutions without knowing if they are optimal solutions. They are based on simplified rules to optimize one or more criteria. The general principle of this category of methods lies in the framework of the scheduling problems and integrates decision strategies to try to build a solution close to optimal while seeking to have a reasonable computing time. Some of these rules are optimal for some scheduling problem. Among these strategies, we distinguish:

- The SPT rule: In the Shortest processing time rule (SPT), jobs are sequenced in order of the processing time required in the workshop. The job requiring the least processing time is scheduled first. The SPT rule is optimal for the $1//\sum_j c_j$ and $1/h_1, pmtn/\sum_j c_j$.

- MSPT rule: This rule tries to optimize the solution returned by SPT by exchanging one job scheduled before the maintenance with another job scheduled after the period of maintenance in order to minimize the objective function.

- WSPT rule: This heuristic schedules jobs in increasing order according to the ratio p_j/w_j. It was introduced by Smith (1956). Smith also proved that the WSPT rule solves optimally the scheduling problem $1//\sum_j w_j c_j$.

- MWSPT rule: This rule tries to optimize the solution returned by WSPT by the same way used in MSPT.

- LPT rule: Longest Processing Time (LPT): LPT is a well-known dispatching rule which allows to schedule jobs according to an order of their decreasing processing times.

- EDD rule: Earliest Due Date is a well-known rule sometimes called as Jackson's rule due to R. Jackson who studied it in 1954. It processes jobs in non-decreasing order of their due dates d_j. EDD is optimal for $1//L_{max}$, $1//T_{max}$ and $1/r_j, pmtn/T_{max}$.

Multistart methods

Multistarts methods are another aspect of heuristic approach that can be defined as strategies designed to provide diversity to the search and overcome local optima. A multistart method is one that executes multiple times from different initial points in the solution space. Once a new solution has been generated, a variety of options can be used to improve it, ranging from a simple greedy algorithm to a complex metaheuristic.

The earliest works in multistart methods were developed for non linear unconstrained optimization problems and consisted of a simply evaluation of the objective function at randomly generated points. In metaheuristic optimization, the multistart algorithms are composed generally of two phases:

- **Diversification generator:** Different strategies can be used to generate an initial solution.
- **Improvement method:** The aim of this phase is to improve the quality of the generated solution from the first phase. Then, each global iteration produces a solution (usually a local optimum) and the best overall is the multistart algorithm's output. For more details please refer to (Martí et al., 2010).

In this dissertation, we propose to solve the studied scheduling problem in Chapter 4 by using a multistart method based on a Lehmer code (Lehmer, 1960) to generate solutions in the first phase, while a Variable Neighborhood Search metaheuristic is investigated in the improvements phase. Further details about these methods are discussed in Chapter 4.

2.6 Investigated metaheuristics within this dissertation

In this thesis, we mainly used two very well known metaheuristics to solve the considered scheduling problems in the following chapters. We will present here the principle of functioning of the basic versions of these methods. More sophisticated versions of these metaheuristic have been studied and proposed in this thesis. The implementation of these methods are detailed in each concerned chapter.

2.6.1 Variable Neighborhood Search (VNS)

Variable neighborhood search (VNS) is a flexible framework for building heuristics proposed by (Mladenović and Hansen, 1997). It uses the idea of changing the neighborhood systematically in order to avoid being trapped in a local optimum during the search. According to (Hansen et al., 2010a) VNS is based on three simple facts:

- **Fact 1:** A local minimum with respect to one neighborhood structure is not necessarily so for another;
- **Fact 2:** A global minimum is a local minimum with respect to all possible neighborhood structures;
- **Fact 3:** For many problems, local minima with respect to one or several neighborhood structure $\mathscr{N}_k$ are relatively close to each other.

VNS is originally designed for approximate solution of combinatorial optimization problems. It was extended to address mixed integer programs, nonlinear programs, and recently mixed integer nonlinear programs. VNS is applied to various fields problem as scheduling, vehicle routing, network design, lot-sizing, artificial intelligence, engineering, pooling problems, biology, phylogeny, reliability, geometry, telecommunication design, etc. (Hansen et al., 2000); (Mladenovic, 1995); (Moreno-Vega and Melián, 2008); (Zhang et al., 2018).

The basic VNS combines two search approaches: a deterministic approach in which a local search methods is applied to find a local optimum and a stochastic approach that is defined as a perturbation of the incumbent solution by jumping to the k^{th} neighborhood in order to escape from local optima.

The perturbation phase: Known as "shaking procedure", the aim of this procedure used within a VNS heuristic is to hopefully escape from local minima traps by selecting a random solution from the k^{th} neighborhood structure.

The pseudo-code of the basic VNS is illustrated in the Algorithm 2

Algorithm 2: A basic Variable Neighborhood Search Metaheuristic

Input: $\mathscr{N}_k$ a neighborhood structure($k \in \{1, 2, ..., k_{max}\}$), f an objective function, S initial solution
Output: S^* best solution found.

$S^* \longleftarrow S$
while $(k \leq k_{max})$ **do**
  $k \longleftarrow 1$
  while *stopping criterion is not reached* **do**
    $S^{'} \longleftarrow$ *perturbation phase*$(\mathscr{N}_k)$
    $S \longleftarrow argmin_{S^{''} \in \mathscr{N}_k(S^{'})} f(S^{''})$
    if $f(S) < f(S^*)$ **then**
      $S^* \longleftarrow S$
      $k \longleftarrow 1$
    end
    else
      $k \longleftarrow k + 1$
    end
  end
end

When we replace the local search in the deterministic phase by the Variable Neighborhood Descent (VND), it can be seen as a generalization of a local search since it explores several neighborhood structures at once instead of one. The resulting algorithm is called a General VNS(GVNS). This latter has received much attention and showed good performances compared to other VNS variants. It has been applied to many optimization problems (Ilić et al., 2010; Lei et al., 2012; Todosijević et al., 2016).

Several other VNS approaches have been derived from the basic VNS namely, Reduced VNS, Skewed VNS, nested VNS, two-level VNS, variable neighborhood decomposition search (VNDS), The primal-dual VNS. All details about these variants can be found in (Hansen et al., 2010a); (Todosijević, 2015).

2.6.2 Tabu Search (TS)

TS is an extension of classical LS methods. In fact, basic TS is based on neighborhood search with local optima avoidance, but in a deterministic way which try to model human memory processes.

Tabu Search Algorithm (TS) is a well known generic metaheuristic for solving hard combinatorial optimization problem, like vehicle routing (Gendreau et al., 1999, 1994), job shop scheduling (Kawaguchi and Fukuyama, 2016), quadratic assignments (James et al., 2009), technician routing and scheduling (Mathlouti et al., 2018), and many others. This method was introduced by Fred Glover (Glover, 1989).

The general idea of TS is to keep track of the regions of the solution space that have already been searched in order to avoid repeating the search near these areas. It starts from a random initial solution and successively moves to one of the neighbors of the current solution. The difference of TS from other metaheuristic approaches is based on the notion of tabu list, which is a special short term memory, that is composed of previously visited solutions that includes prohibited moves. In fact, short term memory stores only some of the attributes of solutions instead of whole solution. We call this characteristic a tabu status. So it gives no permission to revisited solutions and then avoids cycling and being stuck in local optima. During the local search only those moves that are not tabu will be examined if the tabu move does not satisfy the predefined aspiration criteria. These aspiration criteria are used because the attributes in the tabu list may also be shared by unvisited good quality solutions.

Tabu list concept: Tabu list is composed of previously visited solutions that includes prohibited moves. In fact, short term memory stores only some of the attributes of solutions instead of whole solution. This characteristic is called a "tabu status". So it gives no permission to revisited solutions in order to avoid cycling and being stuck in local optima.

During the local search only those moves that are not tabu will be examined. This is achieved by declaring tabu moves that reverse the effect of recent moves for some number of iterations. This number is called the "tabu tenure" of the move.

Aspiration criterion: It is described as algorithmic devices that allows revoking tabus. Indeed, tabus are sometimes too powerful. Consequently, they may prohibit attractive moves even when there is no danger of cycling and then they may lead to an overall stagnation of the search in process which certainly prohibits to reach best ever visited solutions.

Stopping criterion: As explained in (Gendreau and Potvin, 2010), the most commonly used stopping criteria in TS are:

- After a fixed number of iterations (or a fixed amount of CPU time),
- After some number of iterations without an improvement in the objective function value (the criterion used in most implementations),
- When the objective reaches a pre-specified threshold value.

Algorithm 3 describes the main steps of the basic version of a Tabu Search metaheuristic.

Algorithm 3: A basic Tabu Search Metaheuristic

Input: $\mathscr{N}$ a neighborhood structure, f an objective function, S initial solution, T_l a tabu list.
Output: S^* best solution found.

$T_l \longleftarrow \emptyset$
$S^* \longleftarrow S$
while *stopping criterion in not reached* **do**
  $S \longleftarrow argmin_{S' \in \mathscr{N}_\Delta(S)} f(S')$
  if $f(S) < f(S^*)$ **then**
    $S^* \longleftarrow S$
  end
  record tabu for the current move in T_l and delete oldest entry if necessary
end

2.7 Literature review about integrating production scheduling and preventive maintenance

Due to appropriate tradeoffs to be found between maintenance and production activities, many researchers have investigated ways to jointly schedule both activities. Integration of production scheduling and preventive maintenance have been studied by many researchers.

2.7.1 Single machine scheduling problem

The deterministic PM on a single machine research area, where the period and duration of maintenance is known in advance, has been addressed by a significant number of works. A survey of scheduling with deterministic maintenance has been provided by Ma et al. (2010). Benmansour et al. (2012) considered the minimization of the sum of maximum weighted earliness and tardiness costs on a single machine under reliability constraints. They proposed two mathematical models, and a simulation model to solve that problem in the case of stochastic failures. More efficient methods were proposed later in (Benmansour et al., 2014). They also proposed a MILP formulation and heuristics to solve the scheduling problem with PM in order to minimize the weighted sum of maximum earliness and maximum tardiness costs. Ángel-Bello et al. (2011) studied a single machine scheduling problem with availability constraints and sequence-dependent setup costs, with the aim of minimizing the makespan. They derived a mixed integer programming model and valid inequalities that strengthen the linear relaxation and proposed an efficient heuristic procedure that provides a starting feasible solution to the solver. Recently Nesello et al. (2017) proposed new index formulations to solve the problem with aim of minimizing the makespan.

However, there are some works in the literature which attempt to find the better periods of maintenance using simulation. Roux et al. (2013) proposed a hybridization of Nelder-Mead method and multimodel simulation

which is able to integrate production aspects and maintenance strategies. Benmansour et al. (2011) proposed a model of integrated production-maintenance strategy for a failure-prone machine in a just in time environment. Their works focuses on finding simultaneously the period at which preventive maintenance actions have to be performed and the sequence of jobs, in order to minimize the maintenance costs and the expected total earliness and tardiness costs.

2.7.2 Parallel machine scheduling problem

Xu and Yang (2013) addressed the problem of two parallel machines with a periodic availability constraint in order to minimize the makespan. In this purpose, they established a mathematical model by transforming the two parallel machine setting into a single machine setting. They also presented an average-case analysis of the Longest Processing Time first (LPT) algorithm and the List Scheduling (LS) to present the problem. Recently, He et al. (2016) studied the two parallel machine scheduling problem. The objective is to optimize the makespan. They proposed several heuristics with lower bounds to solve and evaluate the performance of their proposed algorithms. In (Li et al., 2017), the authors proposed two mathematical programming models and two improved heuristics to solve the m parallel machine scheduling problem.

2.7.3 Periodic preventive maintenance based scheduling problem

Let us turn to problems dealing with periodic PM. For the single machine problem, many criteria were considered in the literature. Lee and Liman (1992) treated the sum of job flow times. They proved the NP-hardness of the problem and stated a $\frac{2}{7}$ worst-case error bound of the SPT rule. Maximum tardiness criteria was investigated by Liao and Chen (2003) who developed a heuristic and a branch-and-bound algorithm to solve the problem. The makespan criterion is also considered by Ji et al. (2007). They showed that the worst case ratio of the LPT rule is 2, and proved that there is no polynomial time approximation algorithm with a worst-case ratio less than 2 unless $P = NP$. Chen (2007) proposed an efficient heuristic to minimize the total flow time and the maximum tardiness simultaneously. Many other researchers like Chen (2006); Lee and Kim (2012); Low et al. (2010); Ebrahimy Zade and Fakhrzad (2013) contributed among others in this topic.

2.8 Conclusion

This chapter gave a quick outline of scheduling theory and combinatorial optimization tools. Scheduling problems complexity were mentioned and methods for solving these problems were presented. Furthermore, we presented the literature review of scheduling problems under preventive maintenance consideration.

Chapter 3

Single machine scheduling with periodic maintenance policy and weighted sum of completion times (1-PMWCT)

This chapter tackles the single machine scheduling problem with periodic preventive maintenance in order to minimize the weighted sum of completion times. We denote it by 1-PMWCT. This latter is proved to be NP-hard. In this purpose, a MILP formulation is proposed to solve small-sized instances. To solve large-sized instances, we present three properties which generalize already existing works. These properties have been of great use in designing efficient greedy heuristics capable of solving instances with up to 1000 jobs. A GVNS algorithm is also used to solve this problem in order to obtain near optimal solutions for the large-sized instances in a small amount of computational time. To evaluate the performances of the investigated methods, we used the job splitting approach and some special cases of 1-PMWCT to design lower bounds.

Contents

3.1 Introduction

In this chapter, we consider the case of a multitask machine where only minor negligible breakdowns appear (e.x: fuses or butt joint to replace, etc.) and where the cost of repairing big failures is very high. Consequently, the machine must undergo, with much lower cost, preventive maintenance policy to minimize the risk of failures. PM is carried out to meet a regulatory requirement or is applied at regular intervals, and for practical reasons, as a cleaning, cooling or a tool change. If we do not practice a preventive maintenance, a significant risk of failures can be detected on the machine (e.x: electric motor failure) what is going to engender higher cost to repair it. The risk of failures is negligible as the time duration of these latters is insignificant compared to the processing time of the jobs to schedule. Furthermore, PM remains necessary because if it is not carried out then the specified duration to repair the failures increases. In this regard, we address in this chapter a scheduling problem with a single critical machine on the shop denoted as 1-PMWCT (Single machine with Preventive Maintenance and Weighted sum of Completion Times). The considered machine on which a workload is accentuated is subject to unavailability constraints caused by periodic preventive maintenance activities. The aim is to minimize the weighted sum of completion times criterion. Indeed, the weights can represent the holding or inventory cost per time unit for the corresponding products and their priority level (importance, urgency). Firstly, we also proposed a mixed integer linear program (MILP) which can be applied to find an optimum solution on instances of small sizes. Secondly, To solve large-sized instances of the 1-PMWCT problem, we proposed heuristic approaches based on three properties which generalize already existing works. To evaluate the performances of the proposed heuristics, lower bounds based on special cases of the problem are provided. Computational experiments show that the average percentage error of the best heuristic is less than 10. Thirdly, we proposed a GVNS algorithm to solve this NP-hard problem. The proposed algorithm was tested and compared with a MILP model for small-sized instance, and with four lower bounds for instance sizes greater than 20, three of them based on job splitting procedure and one based on the relaxation of one integrity constraint. The computational results show that the proposed GVNS is very efficient both in terms of quality of the objective function values and computational time.

The remainder of this chapter is organized as follows: A formal description of 1-PMWCT is given in Section 3.2. In Section 3.3, we present a small comparison between the 1-PMWCT and the bin-packing problem. Then, a literature review of previous work dealing with PM activity and WCT criterion is proposed in Section 3.4. Basic properties of the optimal schedule and some special cases of the 1-PMWCT problem are presented in Section 3.5. In Section 3.6, we give a description of the whole developed methods to solve the 1-PMWCT namely, a MILP formulation to solve optimally small-sized instances of the problem, nine effective heuristics and a GVNS metaheuristic designed to solve large-sized instances of the problem. Lower bounds derived from special cases of 1-PMWCT and on job splitting approach are then proposed respectively in Sections 3.7 and 3.8. Computational experiments are given in section 3.9 to demonstrate the performance of all the developed methods. Section 3.10 concludes the chapter with a discussion and future extensions.

3.2 Formal description of 1-PMWCT

Formally, the addressed problem can be described as follows: We consider a set $J = \{1, 2, \ldots, n\}$ of n jobs to be processed non-preemptively on a single machine. This latter has to undergo a periodic preventive maintenance (PM) each fixed interval T at each time point $jT - \delta$ ($j \in \mathbb{N}^*$) during the time horizon of schedule where δ refers to the duration of each maintenance activity. Furthermore, we consider the following assumptions:

- The n independent jobs are simultaneously available for processing at the beginning of the planning horizon,
- Each job J_i, with a deterministic processing time p_i, has to be processed without preemption on a single machine that can handle only one job at a time,
- The machine is new at the beginning of the schedule ($t = 0$) and each preventive maintenance action PM renews the machine,
- The set of scheduled jobs between two consecutive maintenance activities is called a batch, denoted by B_j with $j \in \{1, 2, \ldots, n\}$. Generally, T is not enough to process all the jobs and more than one batch have to be programmed,
- Jobs can not be interrupted for a PM action,
- Let $\max_{1 \leq i \leq n}\{p_i\} \leq T - \delta$, otherwise some jobs can not be scheduled,
- Since the time interval T cannot be reduced, idle times can occur before the maintenance activity.

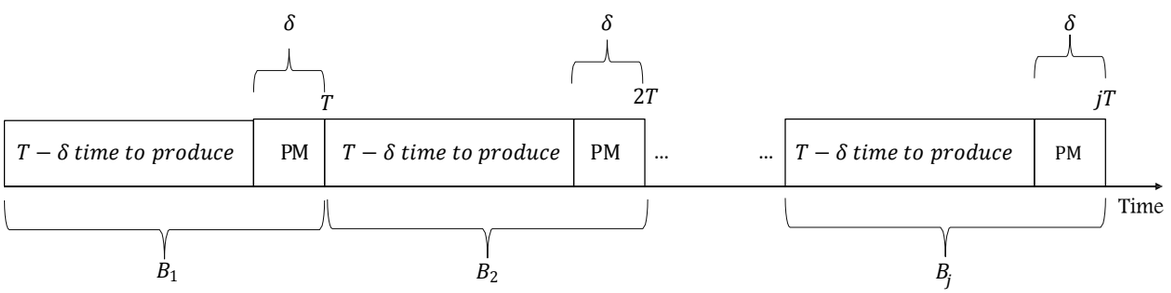

Figure 3.1: Illustration of a feasible solution of the 1-PMWCT problem

Figure 3.1 illustrates a feasible solution of the 1-PMWCT problem.
Before going further in the explanation, let us introduce some notations used in this chapter.

Notations:

- $f(S)$: Objective function value of a feasible solution S of 1-PMWCT problem,

- m: The minimum number of batches that we can have in an optimal solution, m is computed as follows:

$$m = \lceil \frac{\sum_{i=1}^{n} p_i}{T - \delta} \rceil \tag{3.1}$$

- m^* : The number of batches in an optimal solution, with $m^* \geq m$,
- n_j: The number of jobs in the batch B_j with $n = n_1 + n_2 + ... + n_m$,
- W_{B_j} is the sum over the weights of all jobs belonging to a same batch. It is computed as follows: $W_{B_j} = \sum_{i \in B_j} w_i$.

3.3 Bin-packing versus 1-PMWCT

3.3.1 Definition of the Bin-packing problem

Given n items of different weights and bins with each one of capacity c, assign each item to a bin such that number of total used bins is minimized. It is assumed that all items have weights smaller than bin capacity. More details about the bin-packing problem could be found in (Garey and Johnson, 1979).

The Bin-Packing Problem, denoted by BPP, is stated to be NP-hard by Johnson (1985).

3.3.2 Greedy heuristics used to solve the bin-packing problem

There are many heuristics in the related literature dedicated to solve the bin-packing problem. We give here a small definition of the most famous greedy heuristics investigated in the related literature to solve BPP.

Best-Fit heuristic (BF)

Because of its simplicity and good performance, the best-fit heuristic is considered as one of the most effective heuristics to solve the bin-packing problem (Garey and Johnson, 1979). The heuristic BF assigns items to bins according to the order they appear in the list. It places item j in the bin whose current content is the largest but does not exceed the bin. The idea is to place the next item in the bin where a smallest empty space is left.

BF can be implemented in $O(n \log(n))$ time by using Self-Balancing Binary Search Trees.

First-Fit heuristic (FF)

When processing the next item, assign it to the first bin whose current content is the largest but does not exceed the bin. Start a new bin only if it does not. In other words, when FF is called next time, the idea is to start the search of an empty space from the beginning and stop the search at the first bin found with a tightest

spot.

The implementation of First Fit requires $O(n^2)$ time, but First Fit can be implemented in $O(n \log(n))$ time using Self-Balancing Binary Search Trees.

Next-Fit heuristic (NF)

When processing next item, check if it fits in the same bin as the last item (where it left off and not from the beginning). We place that item in a new bin only if it does not exceed the size of the bin. Next Fit is a simple algorithm. It requires only $O(n)$ time to process n items.

Next-Fit is 2-approximate algorithm, which means that the number of bins used by this algorithm is bounded by twice of optimal.

Minimum Bin Slack heuristic (MBS)

MBS is a greedy heuristic which was developed by Gupta and Ho (1999). It is defined as a bin-focused heuristic. At each step, MBS makes an attempt to find a set of items that fit the bin capacity as much as possible.

At each stage a list L of the items not assigned to bins are sorted in a decreasing order of sizes is saved. Each time a packing is determined, the items involved are placed in a bin and removed from L, preserving the sort order. The process ends when the list L becomes empty.

Each packing is determined using a search procedure that tests all possible subsets of items on the list which fits the bin capacity. The subset that allows for the smallest slack is selected. Whoever, if the algorithm finds a subset that fills the bin up completely, the search is stopped. The algorithm restarts to look for a new packing (subset) of the list L.

The complexity of MBS is estimated to $O(2^n)$.

3.3.3 Comparison between 1-PMWCT and BPP

Since preemption is not allowed, any interrupted job should be renewed after the relating maintenance activity. This feature makes the problem similar to the bin-packing problem where the objective is to minimize the number of bins. Indeed, herein the bins represent the batches. Among other things, instead of filling bins with items, we fill batches of limited capacity $T - \delta$ by jobs with time duration p_i. However, the considered criterion in the problem 1-PMWCT is to minimize the weighted sum of completion times. Otherwise, if the considered criterion was the minimization of the makespan, then solving the problem 1-PMWCT goes back to solve the bin-packing problem. In other words, minimizing the number of bins is exactly similar to minimizing the number of batches. With empirical tests, we find counterexamples showing that minimizing the number of batches does not necessarily means minimizing the weighted sum of completion times.

3.4 Related works with PM policy and WCT criterion

Referring to the notation of (Graham et al., 1979), the 1-PMWCT problem is denoted by $1/pm/\sum_{j=1}^{n} w_j c_j$.

Few works dealing with *periodic* preventive maintenance on a single machine have been made but to the best of our knowledge, no one considered such a problem with the objective of minimizing the weighted sum of completion times. We provide hereafter a brief overview of previous related works.

Comprehensive reviews about works dealing with total (weighted) completion times problems with only PM activity have been provided by (Yang et al., 2011).

Lee and Liman (1992) proved that the $1, h_1// \sum_{j=1}^{n} c_j$ is NP-hard and showed that the shortest processing time (SPT) sequence has a worst-case error bound of $\frac{2}{7}$. Regarding the work of Lee (1996), they studied the weighted version and proved the NP-hardness of the problem. Sadfi et al. (2005) solved the same problem with a Modified SPT (MSPT) rule (see Section 2.5.2) with a performance guarantee of 20/17. Sadfi et al. (2002) proposed a dynamic programming algorithm to solve this problem. The same problem with a resumable scenario is considered by Lee (1996). They proved the NP-hardness of the problem. Kacem and Chu (2008b) studied the WSPT and the Modified WSPT (see Section 2.5.2) rules and showed that the worst-case performance ratio is 3 for the two heuristics in some cases and it is unbounded otherwise. When the preemption is allowed $1, h_1/pmtn/\sum_{i=1}^{n} w_j c_j$ problem was proved to be NP-hard by Lee (1996). Kacem and Chu (2008a) proposed new properties of the worst-case performance of the WSPT heuristic. They gave a tighter approximation of the worst-case error and showed that the worst-case bound is equal to 2 under certain conditions. (Kacem and Paschos, 2013) investigated the differential approximation concept to analyze the WSPT rule, proving that a slight modification of this rule provides a $\frac{3-\sqrt{5}}{2}$ differential approximation. Chen et al. (2011) considered the semi-resumable model of a single machine scheduling problem with a non-availability period. They proposed a tight worst-case ratio of LPT rule for the problem with the objective of minimizing makespan, and an approximation algorithm with a worst-case ratio less than 2 for the objective of minimizing the total weighted completion time.

There are also a lot of works dealing with multiple availability constraints such as the number of unavailability periods of the machine is known in advance. (Wang et al., 2005) considered the preemptive single machine scheduling problem with multiple unavailability constraints denoted by $1, h_k/pmtn \sum_{j=1}^{n} w_j c_j$ so as to minimize the total weighted completion time and showed that the problem is NP-hard in the strong sense. Sadfi et al. (2009) considered the same problem for the nonresumable case. They showed that the problem is NP-hard and proposed some lower bounds.

3.5 Properties and some special cases

The aim of this section is to give a brief overview of some special cases and some important properties of an optimal solution of the 1-PMWCT problem. The properties investigated in this section are used to design

effective heuristics dedicated to solve 1-PMWCT.

3.5.1 Properties

Property 1 *In any optimal solution, jobs belonging to each batch are scheduled by Smith's rule Smith (1956).*

Proof The jobs have to be sorted in increasing order of the ratios p_i/w_i in each batch B_j. This rule is called WSPT (weighted SPT) rule or Smith's rule due to W.E. Smith who introduced it in 1956.

The following property generalizes the result obtained by Qi tt al. in Qi et al. (1999). They considered to minimize the sum of job completion times on a critical machine subject to availability constraints. They proved that in an optimal solution, the batches are sorted from the one which has the largest number of jobs to the one with the least number. More precisely we have:

Property 2 *In any optimal solution we have* $W_{B_j} \geq W_{B_{j+1}} \quad 1 \leq j \leq m^* - 1$

Proof Let S_1 and S_2 be two feasible schedules for the problem 1-PMWCT, such that S_1 is obtained from S_2 by swapping two consecutive batches, and let σ be a permutation of the set J.
We suppose that in S_1, the batch B_j, contains the jobs $J_{\sigma(1)}$, $J_{\sigma(2)}$, ..., $J_{\sigma(n_1)}$. The batch B_{j+1} contains the jobs $J_{\sigma(n_1+1)}$, $J_{\sigma(n_1+2)}$, ..., $J_{\sigma(n_1+n_2)}$. In the sequence S_2, only these two batches are interchanged. In order for S_1 to be a better schedule than S_2, we must have the following inequality

$$\Phi = f(S_1) - f(S_2) \leq 0 \tag{3.2}$$

It is obvious that jobs outside B_j and B_{j+1} contribute in the same way to the objective functions $f(S_1)$ and $f(S_2)$. Hence in order to calculate Φ we calculate only the contribution of the jobs inside these two batches. Formally :

$\Phi = \Big(\sum_{i=1}^{n_1} w_{\sigma(i)}c_{\sigma(i)} + \sum_{i=1}^{n_2} w_{\sigma(n_1+i)}c_{\sigma(n_1+i)}\Big) - \Big(\sum_{i=1}^{n_2} w_{\sigma(n_1+i)}c_{\sigma(n_1+i)} + \sum_{i=1}^{n_1} w_{\sigma(i)}c_{\sigma(i)}\Big)$
$\Phi = f(b_1) + f(b_2) - (f(b'_1) + f(b'_2))$.
In $f(S_1)$ we have :
$c_i = jT + \sum_{j=1}^{i} p_j, i = 1...n_1$, and $c_{n_1+i} = (j+1)T + \sum_{j=n_1+1}^{n_1+i} p_j, i = 1...n_2$
In $f(S_2)$ we have:
$c_i = (j+1)T + \sum_{j=1}^{i} p_j, i = 1...n_1$, and $c_{n_1+i} = jT + \sum_{j=n_1+1}^{n_1+i} p_j, i = 1...n_2$.

$f(b_1) = \sum_{i=1}^{n_1} w_i(jT + \sum_{j=1}^{i} p_j) = jT\sum_{i=1}^{n_1} w_i + \sum_{i=1}^{n_1} w_i \sum_{j=1}^{i} p_j$
$f(b_2) = \sum_{i=1}^{n_2} w_{n_1+i}((j+1)T + \sum_{j=n_1+1}^{n_1+i} p_j)$
$f(b'_1) = \sum_{i=1}^{n_1} w_i((j+1)T + \sum_{j=1}^{i} p_j)$
$f(b'_2) = \sum_{i=1}^{n_2} w_{n_1+i}(jT + \sum_{j=n_1+1}^{n_1+i} p_j)$
$\Phi = T(\sum_{i=1}^{n_2} w_{n_1+i} - \sum_{i=1}^{n_1} w_i)$

Hence in order to have $f(S_1) \leq f(S_2)$ the following inequalities must hold.

$$\sum_{1}^{n_2} w_{\sigma(n_1+i)} \leq \sum_{i=1}^{n_1} w_{\sigma(i)} \tag{3.3}$$

$$\sum_{J_{\sigma(i)} \in B_j} w_i - \sum_{J_{\sigma(i)} \in B_{j+1}} w_i \leq 0 \tag{3.4}$$

Let us suppose that the Equation 3.4 is true.
Then, the calculation of both $f(S_1)$ and $f(S_2)$ could be done as follows:

$f(S_1) = f(b_1) + f(b_2)$

$f(S_1) = jT \sum_{i=1}^{n_1} w_i + (j+1)T \sum_{i=1}^{n_2} w_{n_1+i} + \sum_{j=n_1+1}^{n_1+i} p_j + \sum_{i=1}^{n_1} w_i \sum_{j=1}^{i} p_j$

$f_(S_1) = T(j \sum_{i=1}^{n_1} w_i + (j+1) \sum_{i=1}^{n_2} w_{n_1+i}) + \sum_{i=1}^{n_1} w_i \sum_{j=1}^{i} p_j + \sum_{i=1}^{n_2} w_{n_1+i} \sum_{j=1}^{i} p_{n_1+j}$

$f(S_2) = f(b'_1) + f(b'_2)$

$f(S_2) = jT \sum_{i=1}^{n_2} w_{n_1+i} + (j+1)T \sum_{i=1}^{n_1} w_i + \sum_{i=1}^{n_2} w_{n_1+i} \sum_{j=1}^{i} p_{n_1+j} + \sum_{i=1}^{n_1} w_i \sum_{j=1}^{i} p_j$

$f(S_2) = T(j \sum_{i=1}^{n_2} w_{n_1+i} + (j+1) \sum_{i=1}^{n_1} w_i) + \sum_{i=1}^{n_2} w_{n_1+i} \sum_{j=1}^{i} p_{n_1+j} + \sum_{i=1}^{n_1} w_i \sum_{j=1}^{i} p_j.$

If $f(S_1)$ is an optimal value, then the Equation 3.2 should be true.
Let's proceed to the calculation of Equation 3.2:
$f(S_1) - f(S_2) = T\,[j \sum_{i=1}^{n_1} w_i + (j+1) \sum_{i=1}^{n_2} w_{n_1+i}] - T\,[j \sum_{i=1}^{n_2} w_{n_1+i} + (j+1) \sum_{i=1}^{n_1} w_i]$

$f(S_1) - f(S_2) = T\,[j \sum_{i=1}^{n_1} w_i + j \sum_{i=1}^{n_2} w_{n_1+i} + \sum_{i=1}^{n_2} w_{n_1+i} - j \sum_{i=1}^{n_2} w_{n_1+i} - j \sum_{i=1}^{n_1} w_i - \sum_{i=1}^{n_1} w_i]$

$f(S_1) - f(S_2) = T(\sum_{i=1}^{n_2} w_{n_1+i} - \sum_{i=1}^{n_1} w_i)$

$f(S_1) - f(S_2) = \Phi$
At the beginning, we supposed the inequality designed by Equation 3.4
But we succeed to prove the contrary (Equation 3.5)

$$\Phi > 0 \tag{3.5}$$

This means that in any optimal solution of 1-PMWCT the inequality designed by (3.4) is guaranteed.

$$\sum_{J_{\sigma(i)} \in B_j} w_i < \sum_{J_{\sigma(i)} \in B_{j+1}} w_i. \tag{3.6}$$

Property 3 *Let us consider I_j as the idle time in the batch B_j and J_μ the job with the smallest processing time in the batch B_k, with $k \in \{j+1, j+2, ..., m\}$ given that m is the total number of batches.*
In an optimal solution, we have $I_j < p_\mu \ \forall B_j$, with $j \in \{1, ..., m\}$.

Proof The proof is trivial because we can always improve the current solution by moving the job J_μ in the batch B_j.

3.5.2 Special cases

Jobs with equal processing times

When $p_j = p$ for all j in J, the problem 1-PMWCT becomes $1/pm, p_j = p/\sum_{j=1}^{n} w_j c_j$.

It is easy to notice that it exists a polynomial algorithm to solve that scheduling problem. As the processing times are equal, we schedule the job with the largest weight first.

Jobs with equal weights

When $w_j = w$, for all j in J, the objective function becomes $w\sum_{j=1}^{n} c_j$. This problem is NP hard since $1/pm/\sum_{j=1}^{n} c_j$ was proved to be NP-hard by Qi et al. (1999).

Jobs with weights equal to processing times

When $\frac{p_j}{w_j} = 1$, the $1/pm, p_j = w_j/\sum_{i=j}^{n} w_j c_j$ problem was studied and proved to be NP-hard in the two cases of resumable and nonresumable jobs by Lee (1996) and Wang et al. (2005).

3.6 Proposed approach to solve 1-PMWCT

The remainder of this Section is to present the method used to solve 1-PMWCT problem. Thereby, we fist investigated in Sub-section 3.6.1 a MILP formulation of the considered problem, greedy heuristics in Sub-section 3.6.2 and finally a GVNS metaheuristic in Sub-section 3.6.3.

3.6.1 MILP formulation for the 1-PMWCT problem

Since Qi et al. (1999) proved that the special case of minimizing the sum of completion times ($1/pm/\sum_{j=1}^{n} c_j$) is NP-hard, the problem studied in this paper is also stated to be NP-hard.

We propose in this section a MILP formulation for the problem 1-PMWCT based on the completion time variables. This formulation can be used to solve optimally and efficiently small instances of the problem.

Let x_{ik} and y_{ij} be two binary variables defined as:

$$x_{ik} = \begin{cases} 1 & \text{if job } J_i \text{ is scheduled before job } J_k, \\ 0 & \text{otherwise.} \end{cases}$$

$$y_{ij} = \begin{cases} 1 & \text{if job } J_i \text{ belongs to batch } B_j, \\ 0 & \text{otherwise.} \end{cases}$$

As previously mentioned, the set of jobs scheduled between two consecutive periods of maintenance defines a batch B_j. It is obvious that we can have at most n batches. Let us define by $B = \{1, 2, \ldots, n\}$ the set of n batches and by R a big positive integer. On this basis, the MILP model is defined as follows:

$$min\ Z = \sum_{i=1}^{n} w_i c_i \tag{3.7}$$

$$s.t. \quad \sum_{i=1}^{n} p_i\, y_{ij} \leq T - \delta \quad \forall j \in B, \tag{3.8}$$

$$\sum_{j=1}^{n} y_{ij} = 1 \quad \forall i \in J, \tag{3.9}$$

$$c_i \geq p_i + (j-1)T y_{ij} \quad \forall i \in J, \forall j \geq 1, \tag{3.10}$$

$$c_i \leq c_k - p_k + R(1 - x_{ik}) \quad i = 1, 2, \ldots n-1,\ k = i+1, \ldots n, \tag{3.11}$$

$$c_k \leq c_i - p_i + R x_{ik} \quad i = 1, 2, \ldots n-1,\ k = i+1, \ldots n, \tag{3.12}$$

$$x_{ik} \in \{0, 1\} \quad \forall (i,k) \in J^2, \tag{3.13}$$

$$y_{ij} \in \{0, 1\} \quad \forall (i,j) \in J^2. \tag{3.14}$$

Equation (4.1) represents the objective function to be minimized. Constraints (3.8) require that the sum of the processing times of jobs between two consecutive maintenance activities is less or equal to $T-\delta$. Constraints (4.2) allow each job to be performed once in one of the batches. Additionally, in each batch, B_j, from $j = 2$, the execution of any job J_i in this batch is done after the end of the preventive maintenance (4.3). Next sets of constraints, (4.7) and (4.5) indicate that no two scheduled jobs, J_i and J_k, can overlap in time. Finally constraints (4.8) and (4.9) are integrity requirements.

3.6.2 Greedy heuristics

Three rules are considered to form a dispatching rule, based on the sequence of sorted jobs:

- In decreasing order of their corresponding processing times;
- In increasing order of their corresponding processing times;
- In decreasing order of WSPT rule.

Next, for the assignment step, we apply the First-Fit (FF), Best-Fit(BF) and Next-Fit(NF) heuristics dedicated to solve the Bin-Packing problem.

We present here two pseudo-codes of the 9 heuristics we developed.

Let, $J_{[i]}$ denotes the job in the i^{th} position and $p_{[i]}$ and $c_{[i]}$ its corresponding processing time and completion time.

Algorithm 4: Heuristic H_{WSPTBF}. Input: J. Output: $f = \sum_{j=1}^{n} w_j c_j$

Step 1: Sort the jobs according to the WSPT rule. Denote by $S = (J_{\sigma(1)}, J_{\sigma(2)}, ..., J_{\sigma(n)})$ the resulting sequence.
Schedule all jobs in S according to BF strategy. Denote by $S' = (J_{\sigma\prime(1)}, J_{\sigma\prime(2)}, ..., J_{\sigma\prime(n)})$ the resulting sequence.
An Initial feasible solution S' is provided.
Step 2: Denote by $m^{'}$ the number of batches in the initial solution.
Compute W_{B_i}, $\forall(i = 1, 2, ...m^{'})$
Step 3: Sort the batches in decreasing order of W_{B_i} // Property 2
Step 4: Sort the jobs according to the WSPT rule in each batch B_i // Property 1
Denote by $S^{"}$ the resulting sequence.
Step 5: Calculate $c_{[j]}$ the completion time of each job $J_{[j]}$ of $S^{"}$.

Algorithm 5: Heuristic H_{WSPTFF}. Input: J. Output: $f = \sum_{j=1}^{n} w_j c_j$

Step 1: Sort the jobs according to the WSPT rule. Denote by $S = (J_{\sigma(1)}, J_{\sigma(2)}, ..., J_{\sigma(n)})$ the resulting sequence.
Step 2: Schedule all jobs of S with First Fit (FF) strategy. Denote by $S^{'} = (J_{\sigma\prime(1)}, J_{\sigma\prime(2)}, ..., J_{\sigma\prime(n)})$ the resulting sequence. An initial feasible solution $S^{'}$ is provided.
Step 3: Apply steps 3-5 of H_{WSPTBF}

Remark 1 *All the implemented heuristics are deterministic.*

A numerical example

To illustrate the proposed heuristic algorithms, we present a simple example to show clearly the functioning of the H_{WSPTBF} heuristic.

Let $n = 12$ jobs, $T = 50$, $\delta = 1$, the processing times and the weights of the jobs are given in Table 3.1.

	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9	J_{10}	J_{11}	J_{12}
p_i	6	15	11	17	7	13	18	11	4	10	18	5
w_i	10	14	10	14	19	11	14	18	12	19	15	15

Table 3.1: Numerical example

Steps 1 and 2: Sort the jobs according to the WSPT rule and use the Best Fit strategy to schedule the jobs. The resulting sequence of the initial solution is

$S = (J_9, J_{12}, J_5, J_{10}, J_1, J_8, J_2, J_3, J_6, J_{11}, J_4, J_7)$

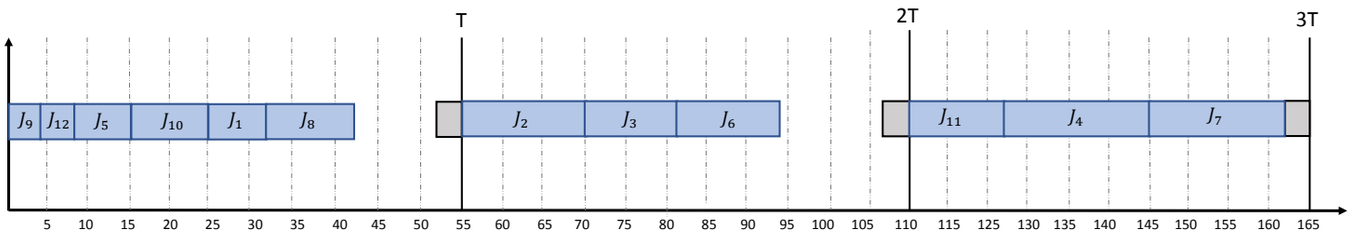

Figure 3.2: Gantt chart of the initial solution returned by steps 1 and 2

with the objective value $f(S) = \sum_{j=1}^{12} w_j c_j = 10526$.

Step 3: The initial solution is composed of three batches: The calculation of W_{B_i} of each batch i gives:

- $B_1 : \{J_9, J_{12}, J_5, J_{10}, J_1, J_8\}$,
- $B_2 : \{J_2, J_6, J_3\}$,
- $B_3 : \{J_{11}, J_4, J_7\}$,

With: $\sum_{j \in B_1} w_j = 93$, $\sum_{j \in B_2} w_j = 35$ and $\sum_{j \in B_3} w_j = 43$.

Step 4: Sort the batches in decreasing order of W_{B_i}.

Step 5: Sort the jobs in each batch with the WSPT rule.

In this example, the jobs in each batch, are already sorted according to the WSPT rule.

Thus, the objective value of the solution returned by the H_{WSPTBF} heuristic is

$f(S) = \sum_{j=1}^{12} w_j c_j = 10266$.

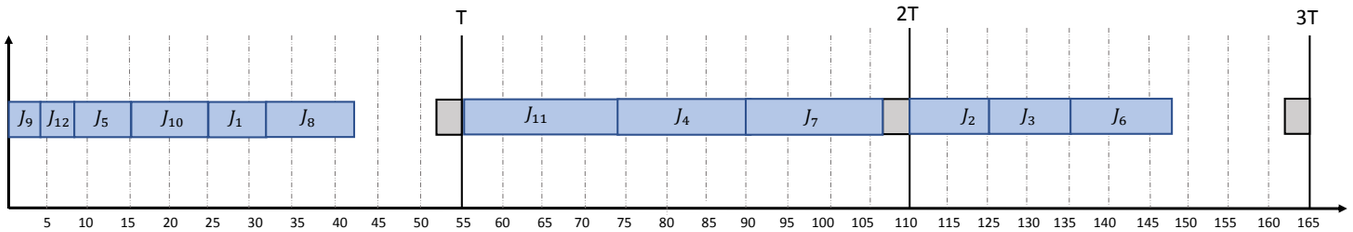

Figure 3.3: Step 4 of the H_{WSPTBF}

3.6.3 General Variable Neighborhood Search to solve 1-PMWCT

A GVNS metaheuristic is investigated in this section to solve the large sized instances of 1-PMWCT. As already mentioned in Chapter 2, GVNS is a variant of VNS where the intensification phase designed by a simple local search procedure is replaced by a VND procedure. The perturbation phase, well know in the literature as the shaking procedure, remains the same.

Algorithm 6: GVNS. Input: L_{max}, k_{max}, S. Output: S^*

```
S* ← WLS(S)
while number of iterations ≤ L_max do
    S^shake ← shaking(S*)
    if S^shake is better than S* then
        S* ← S_shake
    end
    while f(S') < f(S*) do
        S' ← VND(S^shake, k_max)
        if f(S') < f(S*) then
            S^shake ← S'
            S* ← S'
        end
    end
end
```

Algorithm 6 provides the main steps of the implemented GVNS for 1-PMWCT problem. It begins by an initial feasible solution returned by a greedy heuristic named WLS procedure. Then, while the counter L_{max} is not reached, a new solution named $Sshake$ using the shaking procedure is generated. That solution is enrolled within the VND procedure which returns at its end a new optimum for the local search. At each iteration the best solution S^* is recorded.

In the following sub-sections we explain how we managed to develop our initial solution generator WLS and we give respectively details about both the perturbation and intensification phases. Before this, let us introduce the investigated neighborhood structure used within the different steps in the implemented GVNS.

Neighborhood structures:

The GVNS for the 1-PMWCT is based on moves using one or more operators. A move is defined as the action of changing the position of a job from its current place in the sequence to another one. We define here five operators Δ_i ($i \in \{1, ..., 5\}$) which are used to generate respectively five neighborhood structures $\mathscr{N}_i(S)$) of a solution S. These neighborhood structures have been widely used in literature to solve different combinatorial optimization problems specially for those which are represented by permutations(Guo et al., 2013).

$\mathbf{\Delta_1}$: This operator consists to swap (interchange) only two consecutive jobs of S.

The complexity for going through the entire neighborhood structure $\mathscr{N}_1(S)$ is $O(n)$.

$\mathbf{\Delta_2}$: This operator consists to swap (interchange) all pairs of jobs in S.

The complexity for going through the entire neighborhood structure $\mathscr{N}_2(S)$ is $O(n^2)$.

$\mathbf{\Delta_3}$: It consists to insert each job of the sequence S at the position k ($1 \leq k \leq n$).

The complexity for going through the entire neighborhood structure $\mathscr{N}_3(S)$ is $O(n)$.

$\mathbf{\Delta_4}$: The 2-*opt* operator is the most classical one when it comes to solve the traveling salesman problem (Helsgaun, 2006). It removes two edges from the circuit and reconnects the two paths created. In our implementation, the 2-*opt* operator selects two jobs J_i and J_j from the current sequence, then constructs a new sequence which deletes the connection between J_i and its successor J_{i+1} and the connection between J_j with its successor J_{j+1}, and then connects J_i with J_j and J_{i+1} with J_{j+1}. Furthermore, the sub-sequence between J_{i+1} and J_{j+1} is reversed.

The complexity for going through the entire neighborhood structure $\mathscr{N}_4(S)$ is $O(n^2)$.

$\mathbf{\Delta_5}$: It consists to construct a new sequence S' from S by starting S' from each job J_i and finish with the job J_{i-1} ($i \geq 2$).

The complexity for going through the entire neighborhood structure $\mathscr{N}_5(S)$ is $O(n)$.

Initial solution generator: WLS procedure

Since GVNS is a single-solution based metaheuristic, we need to start from a given solution. In this purpose we proposed the **WLS** procedure described in algorithm 7 to generate feasible solution to 1-PMWCT problem.

Due to the impact of the WSPT on the quality solution, our WLS procedure starts by sorting jobs of S with the WSPT rule. Then, a Descent Search Heuristic DSH^2 (see Chapter 2 Section 2.5.2) is used until a local optimum S^* is reached. The index 2 in DSH^2 refers to the used operator Δ_2 within the local search used in DSH.

Note that to generate a neighborhood structures in DSH, the five operators presented in Sub-section 3.6.3 are tested before choosing Δ_2. This choice is motivated by preliminary tests that are presented in what follows.

Algorithm 7: WLS.	Input: S.	Output: S^*
Ranks the jobs in S according to the WSPT rule		
$S^* \longleftarrow DSH^2(S)$		

Testing local searches

We propose to compare local searches based on the predefined five neighborhood structures (i.e. $\mathscr{N}_1$, $\mathscr{N}_2$, $\mathscr{N}_3$, $\mathscr{N}_4$, $\mathscr{N}_5$) in order to determine the best one to use within the WLS heuristic (Algorithm 7) proposed to generate initial solutions. For each operator, we start each time from the same feasible solution returned by the WSPT rule. The summarized results are reported in Table 3.2. $MinV$ gives the minimum value among the five results ($Value_{\mathscr{N}_i}$, $i \in \{1, ..., 5\}$) returned by each tested local search for each instance of size n.

$$MinV = \min\{Value_{\mathscr{N}_1}, Value_{\mathscr{N}_2}, Value_{\mathscr{N}_3}, Value_{\mathscr{N}_4}, Value_{\mathscr{N}_5}\}. \tag{3.15}$$

The gap reported in columns 2, 3, 4 and 5 is calculated as follows:

$$100 * ((Value_{\mathscr{N}_i} - MinV)/Value_{\mathscr{N}_i}). \tag{3.16}$$

While comparing the results in Table 3.2, we observe that local search based on $\mathscr{N}_2$ (Δ_2 operator), significantly

n	$gap_{\mathscr{N}_1}$	$gap_{\mathscr{N}_2}$	$gap_{\mathscr{N}_3}$	$gap_{\mathscr{N}_4}$	$gap_{\mathscr{N}_5}$
10	1.01	0.00	0.10	0.06	3.04
15	2.37	0.01	0.00	0.02	6.69
20	3.92	0.00	0.71	0.26	6.58
25	4.89	0.00	1.29	0.28	8.81
30	4.78	0.00	1.11	0.45	9.12
35	2.99	0.00	0.57	0.10	9.19
40	5.70	0.00	1.02	0.40	9.83
45	6.04	0.00	0.80	0.18	10.03
50	4.79	0.00	0.62	0.05	9.48

Table 3.2: Gap comparison between the five LS procedures for $n = 10$ to $n = 50$

outperforms all the other local searches as it obtains the minimum value at almost every instance. Otherwise, local search based on $\mathscr{N}_4$ (*Restart* operator) provides the worst values. To summarize, the Neighborhoods are ranked as follows in increasing order with respect to their efficiency: $\mathscr{N}_4$, $\mathscr{N}_1$, $\mathscr{N}_3$, $\mathscr{N}_5$ and $\mathscr{N}_2$. This result justifies our choice to use the *Swap* operator in the local search proposed within the initial solution generator.

Perturbation phase: Shaking procedure

As some random jumps in the search space could be efficient to escape from local optima and contribute to diversify the search, our shaking procedure consists of randomly generating a neighboring solution S' from the current solution S using the 2-*opt* operator. Due to its path reversion, the 2-opt operator is very efficient when applied to explore the search space.

Preliminary tests show us that, for our problem, the shaking procedure with more operators reduces the quality of the results. In order to tune the strength of the diversification induced by the 2-*opt* operator, the shaking procedure is repeated L_{max} times in the proposed GVNS presented in Algorithm 6. This parameter is

fixed on the basis of primarily tests presented in the followings.

$\mathbf{L_{max}}$ calibration

Here, we examined the influence of the parameters L_{max} in the GVNS. To do this, we tested different values of L_{max}. The testing is performed by varying L_{max} from 5 to 80.

L_{max} can be understood as the amount of time that the shaking procedure is used within the proposed GVNS. The obtained results are presented in Table 3.3 for fifteen instances of size $n = 100$. The large size of these instances could reveal the performance of the initial solution for a given value L_{max}. For each value of L_{max} the average solution value found and also the average time spent upon reaching these solutions are presented. As GVNS is a stochastic heuristic, we run this latter 10 times for each instance. Since the coefficient of variation (i.e. the relative standard deviation) is smaller than 0.1% we report only the best of these 10 replicates.

L_{max}	$Aver.Value$	$Aver.Time(s)$
5	9,204,698	0.99
10	9,204,119.26	1.07
20	9,201,641.53	1.23
30	9,200,712.33	1.37
40	9,196,670.20	1.55
60	9,196,668.10	1.81
80	9,196,666.50	2.06

Table 3.3: WLS with different values of L_{max}

From the summarized results in Table 3.3, it follows that the proposed GVNS is very sensitive to the value of L_{max} parameter. Indeed, it turns out that the proposed algorithm returns the best solution values when the parameter L_{max} is upper to 40. Same conclusion can be drawn for L_{max} parameter, as the results returned after $L_{max} = 40$ seem to be very close. Consequently, for the rest of the testing, L_{max} is set to 40 when consuming the least amount of time (1.55 seconds).

Intensification phase: VND procedure

A complete local search is designed as a Variable Neighborhood Descent (VND) for each solution returned by the shaking procedure. In this work, we fixed the number of neighborhoods to three ($k_{max} = 3$), as it is advised in the relating literature (Glover and Kochenberger, 2006).

Neighborhood change strategy

The proposed VND operates a *changing neighborhood strategy*, which consists to explore a complete neighborhood using the same operator Δ_i as long as there is an improvement (Best improvement), otherwise another

operator is used. The pseudo code of VND proposed in this work is illustrated by Algorithm 8.

Algorithm 8: VND. Input: S^{shake}, k_{max}. Output: $S^{'}$

```
S' ← S^shake
while number of iterations ≤ k_max do
    S^shake ← argmin_{S''∈N_{i(S')}} f(S'')
    if S^shake is better than S' then
        S' ← S^shake
    end
    else
        add an iteration
    end
end
```

To efficiently explore possible solutions it is important to establish the "best" order of the three neighborhood structures defined above. Due to this, we performed preliminary tests. Six different orderings ($a_{i,j}$) of the three neighborhood structures are listed in Table 3.5.

Neighborhoods order within VND

It is important to establish the "best" order of the neighborhood structures since it affects the efficiency of the GVNS algorithm. To do this, we performed a set of preliminary experiments. The different orderings are listed in Table 3.5 presented in Section 3.9. Focusing on results observed in the last paragraph we decided to implement the $\mathscr{N}_5$, $\mathscr{N}_3$, $\mathscr{N}_2$ complete neighborhood structures based respectively on the following operators: 2-*Opt*, *Insertion* and *Swap*. The summarized results are reported in Table 3.4. To test the performances of each case order a_{ij} presented in Table 3.5, we compared its results with $minV$ (Equation 3.17) for each instance size, namely $n = 10, 15, 20, 25, 30, 35, 40, 45, 50, 60, 70, 80, 90, 100$.

$$minV = \min\{Value_{a_{1,1}}, Value_{a_{1,2}}, Value_{a_{1,3}}, Value_{a_{2,1}}, Value_{a_{2,2}}, Value_{a_{2,3}}\}. \tag{3.17}$$

The formulation below is used for the comparison:

$$gap_{a_{i,j}} = 100 * ((Value_{a_{i,j}} - minV)/Value_{a_{i,j}}). \tag{3.18}$$

The entries $Value_{a_{i,j}}$ represent the value returned by the proposed VND.

As seen in Table 3.4, the returned results of the six tests are almost similar. It follows that, for our problem, VND is not very sensitive to the exploration order of the neighborhood structures. In spite of this, the results show that the order $a_{1,1}$ (*swap*, *insertion*, 2-*Opt*) is better than the other orders, since it returns the minimum value or a very tight value from the minimum for the large instances. In our proposed GVNS we suggest to use this order of neighborhoods in the VND heuristic.

n	$gap_{a_{1,1}}$	$gap_{a_{1,2}}$	$gap_{a_{1,3}}$	$gap_{a_{2,1}}$	$gap_{a_{2,2}}$	$gap_{a_{2,3}}$
10	0.16	0.16	0.20	0.00	0.23	0.23
15	0.23	0.19	0.00	0.00	0.06	0.23
20	0.26	0.12	0.08	0.02	0.00	0.06
25	0.00	0.27	0.49	0.34	0.44	0.34
30	0.27	0.37	0.25	0.14	0.00	0.13
35	0.18	0.07	0.00	0.15	0.17	0.04
40	0.02	0.00	0.24	0.02	0.39	0.05
45	0.03	0.00	0.05	0.09	0.08	0.13
50	0.00	0.17	0.17	0.34	0.14	0.25
60	0.04	0.12	0.11	0.10	0.42	0.12
70	0.20	0.20	0.12	0.12	0.38	0.14
80	0.18	0.10	0.23	0.13	0.47	0.09
90	0.04	0.17	0.19	0.13	0.11	0.00
100	0.04	0.17	0.19	0.13	0.11	0.00
Average	0.11	0.15	0.16	0.13	0.21	0.14

Table 3.4: Neighborhood structures test cases

Table 3.5: Neighborhood structure test cases

$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
Swap	Swap	Insertion
Insertion	2-Opt	Swap
2-Opt	Insertion	2-Opt
$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
Insertion	2-Opt	2-Opt
2-Opt	Insertion	Swap
Swap	Swap	Insertion

3.7 Lower bounds derived from relaxed problems of 1-PMWCT

This Sub-section presents lower bounds through polynomial relaxed problems. This can be useful in order to evaluate the quality of the heuristics presented in the previous section.

For the first proposed bound, the relaxation of maintenance constraints is considered. In that case, we consider the $1//\sum_{i=j}^{n} w_j c_j$ problem which is proved to be polynomially solvable via the WSPT rule Smith (1956).

Proposition 1

$$LB_1 = \sum_{i=1}^{n} w_i \sum_{j=1}^{i} p_j \tag{3.19}$$

The equation 3.19 is a valid lower bound for the 1-PMWCT problem.

Proof Solving the problem 1-PMWCT without taking into account the maintenance constraints gives rise to the following problem $1//\sum_{i=1}^{n} w_i c_i$. The optimal value f^* of the latter problem is obtained via the WSPT rule and constitutes a lower bound for 1-PMWCT.

In order to compute the value of f^* we order the jobs by the WSPT rule i.e. $(\frac{p_1}{w_1} \leq \frac{p_2}{w_2} \leq \ldots, \leq \frac{p_n}{w_n}\}$ and schedules the jobs such that $c_i = \sum_{j=1}^{i} p_j$. Hence, the optimal value is:

$$f^* = \sum_{i=1}^{n} w_i c_i$$
$$f^* = w_1 c_1 + w_2 c_2 + ... + w_n c_n$$
$$f^* = w_1 p_1 + w_2(p_1 + p_2) + w_3(p_1 + p_2 + p_3) + ... + w_n(p_1 + p_2 + p_3 + ...p_n).$$
$$f^* = w_1 p_1 + w_2 \sum_{j=1}^{2} p_j + w_3 \sum_{j=1}^{3} p_j + ... + w_n \sum_{j=1}^{n} p_j$$
$$f^* = \sum_{i=1}^{n} w_i \sum_{j=1}^{i} p_j.$$

Proposition 2

$$LB_2 = p \sum_{i=1}^{n} i w_i + T \sum_{k=1}^{m-1} \left(k \sum_{i=1}^{n_{k+1}} w_{n_1+n_2+...+n_k+i} \right) \tag{3.20}$$

The Equation 3.20 is a valid lower bound of the 1-PMWCT problem.

Proof Let us consider the scheduling problem (P') denoted by $1/pm, p'_i = p/\sum_{i=1}^{n} w_i c_i$. It is obvious that (P') is a special case of 1-PMWCT. (P') is shown to be polynomial in Sub-section 3.5.2.

We consider the special case where p is set to the minimum value of all the processing times of the jobs in 1-PMWCT i.e. $p = \min_{1\leq i\leq n}\{p_i\}$. Then an optimal solution of P' is obtained by sorting the jobs in decreasing order of their weights w_i, and the corresponding value of this solution can be calculated as follows:

$f^* = \sum_{i\in B_1} w_i c_i + \sum_{i\in B_2} w_i c_i + ... + \sum_{i\in B_m} w_i c_i$ and $c_i = n_j p + (j-1)T \ \forall i \in B_j$.

Hence,

$f^* = \sum_{i\in B_1} w_i n_1 p + \sum_{i\in B_2} w_i (n_2 p + T) + ... + \sum_{i\in B_m} w_i (n_m p + (m-1)T)$

$$f^* = p\left(\sum_{i=1}^{n_1} i w_i + \sum_{i=1}^{n_2} i w_{n_1+i} + ... + \sum_{i=1}^{n_m} i w_{n_1+n_2+...+n_{m-1}+i}\right) +$$

$$T\left(\sum_{i=1}^{n_2} w_{n_1+i} + 2\sum_{i=1}^{n_3} w_{n_1+n_2+i} + ... + (m-1)\sum_{i=1}^{n_m} w_{n_1+n_2+...+n_{m-1}+i}\right)$$

$$f^* = p\sum_{i=1}^{n} i w_i + T\sum_{k=1}^{m-1}\left(k\sum_{i=1}^{n_{k+1}} w_{n_1+n_2+...+n_k+i}\right).$$

As P' is a relaxed problem of 1-PMWCT, LB_2 is a valid lower bound of 1-PMWCT problem.

In order to derive the third lower bound LB_3, let consider the problem (P'') denoted as following: $1, h_1 // \sum_{i=1}^{n} w_i c_i$. The first field means that there exists only one machine and specifies that only one preventive maintenance (h_1) is applied during the whole scheduling horizon.

Let us denote by J_g the job scheduled just after the maintenance period, J_{m_1} and J_{m_2} are respectively the last job scheduled in batches B_1 and B_2.

LB_3 is formulated in the following proposition:

Proposition 3 *The lower bound*

$$LB_3 = \sum_{i=1}^{m_1} w_i \sum_{j=1}^{i} p_j + \sum_{i=g}^{m_2} w_i (T + \sum_{j=g}^{i} p_j) \tag{3.21}$$

The Equation 3.21 is a valid lower bound for 1-PMWCT.

Proof LB_3 is based on an optimal solution returned by a dynamic programming algorithm which is proposed by Kacem et al. (2008) to solve P''. This algorithm sorts the jobs according to the WSPT and schedules them in increasing order according to their index.

The dynamic formulation of P'' is based on the following recursive equation:

$$f(i,t) = min\left\{f(i-1,t) + w_i \cdot (T + \sum_{k=1}^{i} p_k - t), f(i-1, t-p_i) + w_i \cdot t\right. \tag{3.22}$$

Where $f(i,t)$ is the smallest total weighted completion time of jobs $\{1,,i\}$, with the total processing time of jobs in the first batch B_1 starts at $t \leq T-\delta$. If such a solution does not exist, set $f(i,t) = +\propto$, $f(0,0) = 0$, $f(0,t) = +\propto, \forall t \leq (T-\delta)$ and $f^* = \min\limits_{0\leq t\leq(T-\delta)} f(n,t)$.

We point out that the time complexity of this algorithm is in $O(n \cdot (T-\delta))$.

P'' can be defined as a relaxed problem of 1-PMWCT due to the fact of moving from multiple periods of maintenance to only one period.

Let be $f^*_{S''}$ an optimal value of a solution S'' for the problem P''. The value $f^*_{S''}$ can be computed as follows:
$f^*_{S''} = \sum\limits_{i\in B_1} w_i c_i + \sum\limits_{i\in B_2} w_i c_i$

$f^*_{S''} = w_1c_1 + w_2c_2 + ... + w_{m_1}c_{m_1} + w_g c_g + w_{g+1}c_{g+1} + ... + w_{m_2}c_{m_2}$

$f^*_{S''} = w_1p_1 + w_2(p_1+p_2) + ... + w_{m_1}(p_1+p_2+...+p_{m_1}) + w_g(T+p_g) + w_{g+1}(T+p_g+p_{g+1}) + ... + w_{m_2}(T+p_g+p_{g+1}+...+p_{m_2})$

$f^*_{S''} = \sum\limits_{i=1}^{m_1} w_i \sum\limits_{j=1}^{i} p_j + T\sum\limits_{i=g}^{m_2} w_i + \sum\limits_{i=g}^{m_2} w_i \sum\limits_{j=g}^{i} p_j$

$f^*_{S''} = \sum\limits_{i=1}^{m_1} w_i \sum\limits_{j=1}^{i} p_j + \sum\limits_{i=g}^{m_2} w_i (T + \sum\limits_{j=g}^{i} p_j)$

As (S'') is a relaxed problem of 1-PMWCT, LB_3 is equal to $f^*_{S''}$.

In a special case where all the jobs are scheduled in the first batch ($\sum\limits_{i=1}^{n} p_i \leq T-\delta$), LB_3 is equal to the optimal value of the problem 1-PMWCT. This is due to the WSPT rule which is applied in the dynamic programming algorithm.

Proposition 4

$$LB_4 = \sum_{i=1}^{n} p_i w_i + T \cdot (\frac{m(m-1)}{2}) \tag{3.23}$$

The Equation 3.23 is a valid lower bound for 1-PMWCT problem.

Proof Let f^* be the objective function value of an optimal solution of 1-PMWCT and w the smallest weights among all the weights related to an instance of 1-PMWCT. w is computed as follows:

$$w = \min_{1\leq i\leq n} \{w_i\} \tag{3.24}$$

f^* is computed as follows:

$$f^* = \sum_{i\in B_1} w_i c_i + \sum_{i\in B_2} w_i c_i + ... + \sum_{i\in B_{m^*}} w_i c_i \tag{3.25}$$

When developing the Equation 3.25, we have:

$f^* = w_1p_1+w_2(p_1+p_2)+...+w_{n_1}(p_1+p_2+...+p_{n_1})+w_{n_1+1}(T+p_{n_1+1})+w_{n_1+2}(T+p_{n_1+1}+p_{n_1+2})+...+w_{n_1+n_2}(T+p_{n_1+1}+p_{n_1+2}+...+p_{n_1+n_2})+...+w_{n_1+n_2+...+n_{m^*-1}+1}((m^*-1)T+p_{n_1+n_2+...+n_{m^*-1}+1})+w_{n_1+n_2+...+n_{m^*-1}+2}((m^*-1)T+p_{n_1+n_2+...+n_{m^*-1}+1}+p_{n_1+n_2+...+n_{m^*-1}+2})+...+w_n((m^*-1)T+p_{n_1+n_2+...+n_{m^*-1}+1}+...+p_n)$

$$f^* = \sum_{i=1}^{n} p_i w_i + \sum_{i=2}^{n} w_i \sum_{j=1}^{i-1} p_j + T(\sum_{i=1}^{n_2} w_{n_1+i} + 2\sum_{i=1}^{n_3} w_{n_1+n_2+i} + ... + (m^*-1)\sum_{i=1}^{n_{m^*}} w_{n_1+n_2+...+n_{m^*-1}+i}).$$

Since $m \leq m^*$, it is clear that:

$$T(\sum_{i=1}^{n_2} w_{n_1+i} + 2\sum_{i=1}^{n_3} w_{n_1+n_2+i} + ... + (m^*-1)\sum_{i=1}^{n_{m^*}} w_{n_1+n_2+...+n_{m^*-1}+i}) \geq T(w+2w+3w+...+(m-1)w)$$

This is equivalent to:

$$T\sum_{k=1}^{m^*-1}\left(k\sum_{i=1}^{n_{k+1}} w_{n_1+n_2+...+n_k+i}\right) \geq Tw\sum_{i=1}^{m-1} i.$$

Since $\sum_{i=1}^{m-1} i = \frac{m(m-1)}{2}$, we consequently have:

$$\sum_{i=1}^{n} p_i w_i + T\cdot(\frac{m(m-1)}{2}) \leq \sum_{i=1}^{n} p_i w_i + \sum_{i=2}^{n} w_i \sum_{j=1}^{i-1} p_j + T\sum_{k=1}^{m^*-1}\left(k\sum_{i=1}^{n_{k+1}} w_{n_1+n_2+...+n_k+i}\right).$$

LB_4 is a valid lower bound for 1-PMWCT.

Remark 2 *The code of the three lower bounds LB_1, LB_2, LB_4 is composed of elementary operations and sorting functions which can be implemented with a complexity of $O(n\log(n))$.*

3.8 Lower bounds derived from job splitting procedure

3.8.1 Job splitting

Job splitting is proposed as a general technique to obtain lower bounds. This concept was introduced by (Posner, 1985) and used later by (Belouadah et al., 1992). They proved that if a job J_j is split into two jobs J_i and J_k with processing times $p_j = p_i + p_k$ and weights $w_j = w_i + w_k$, it decreases the total weighted completion time by $p_k w_i$. This method was used by (Kacem et al., 2008) to propose lower bounds for a single machine scheduling problem with only one unavailability period of the machine during the scheduling horizon.

Let P denote the original problem with no split jobs and let P_1 be the corresponding problem in which each job J_j is split into n_j pieces. Each piece $o_{k,j}$ has a processing time $p_{k,j}$ and a weight $w_{k,j}$ ($j \in J, k \in \{1,\ldots,n_j\}$)

such that $p_j = \sum_{k=1}^{n_j} p_{k,j}$ and $w_j = \sum_{k=1}^{n_j} w_{k,j}$, the pieces $o_{k,j}$ are constrained to be scheduled contiguously. Let σ^* be an optimal schedule for P and σ_1^* the corresponding optimal schedule for P_1. (Belouadah et al., 1992) proved the following relation:

$$\sum_{j\in J} w_j c_j(\sigma^*) - \sum_{j\in J} w_j c_j(\sigma_1^*) = CBRK \tag{3.26}$$

$$CBRK = \sum_{j\in J} \big(\sum_{k=1}^{n_j-1} w_{k,j} \sum_{h=k+1}^{n_j} p_{h,j} \big) \tag{3.27}$$

As suggested by (Belouadah et al., 1992), CBRK may be thought as the cost of breaking all the jobs J_j into n_j pieces.

By relaxing the contiguity constraint in problem P_1, we obtain a relaxed problem denoted as P_2. Let σ_2^* be an optimal schedule for problem P_2. (Belouadah et al., 1992) proposed the following relation:

$$\sum_{j\in J} w_j c_j(\sigma^*) \geq \sum_{j\in J} w_j c_j(\sigma_2^*) + CBRK \tag{3.28}$$

To get the lower bounds, the jobs should be split in such a way that the resulting problem P_2 corresponds to a special case for which an optimal schedule σ_2^* is known. This is particularly the case when processing times are constant. In this context, let us consider three special splits:

3.8.2 Suggested lower bounds

First split: Each job J_j ($j \in J$) is split into two pieces such that $p_{1,j} = p_{2,j} = p/2$ and $w_{1,j} = w_j$, $w_{2,j} = 0$ with $p = \min_{1\leq j\leq n} p_j$. Let $\sigma_2^{'*}$ denote the optimal solution of this problem. Hence, we have:

$$LB_5 = \sum_{j\in J} w_j c_j(\sigma_2^{'*}) + CBRK \tag{3.29}$$

Second split: The second lower bound ($LB2$) is obtained by considering the following split: Each job J_j ($j \in J$) is split into two pieces such that $p_{1,j} = p_{2,j} = p/2$ and $w_{1,j} = w_{2,j} = w/2$ with: $p = \min_{1\leq j\leq n} p_j$ and $w = \min_{1\leq j\leq n} w_j$. Let denote $\sigma_2^{''*}$ the optimal solution of this problem. Hence, we have:

$$LB_6 = \sum_{j\in J} w_j c_j(\sigma_2^{''*}) + CBRK \tag{3.30}$$

Third split: P_2 can be effectively solved when all processing times are set to one unit of time ($p_{k,j} = 1$), which means that each job J_j ($j \in J$) is split into p_j pieces. The weights are set as follows:

If $w_j \geq p_j$, $w_{k,j} = 1$, for $k \in \{1, \ldots, p_j\}$,

If $w_j \leq p_j$, for $w_{k,j} = 1$, $k \in \{1, \ldots, w_j\}$ and $w_{k,j} = 0$, for $k \in \{w_j + 1, \ldots, p_j\}$
Let $\sigma_2^{'''*}$ denote the optimal solution of this problem. Hence, we have:

$$LB_7 = \sum_{j \in J} w_j c_j(\sigma_2^{'''*}) + CBRK \tag{3.31}$$

3.9 Computational results

In order to assess the extent of the MILP formulation, to evaluate the performance of the proposed heuristics and the proposed GVNS, some computational experiments are conducted.

Since we do not find test instances for the problem under study in the related literature, we decided to carry out experiments with randomly generated instances. Theses instances are available via the following URL: http://sakai.uphf.fr/wiki/site/1b6900a5-38f1-4003-94e9-317898cbf168/instancesmaintenancewct.html

3.9.1 Instance generator

The instances used in the experiments are divided in four benchmark classes named CLxDy. When $x = 1$, p_i is uniformly distributed over $[20, 100]$. When $x = 2$, then p_i is uniformly distributed over $[25, 50]$ and $T = 100$. When $y = 0$, then $\delta = 0$ and if it is equal to 1, $\delta = 2\%T$. Each weight w_i, it is uniformly distributed over $[1, n]$. The period of maintenance is generated as follows:

$$T = \max\left(\max_{1 \leq j \leq n}(p_j), 4 * \left(\sum_{j=1}^{n} p_j/n\right)\right). \tag{3.32}$$

To ensure feasibility of instances from CL1D0 and CL1D1 classes, T is generated in a way that makes it greater than the largest processing time p_i. Also, to avoid solutions with only one job per batch, we choose T greater than 4 times the average of the processing times. To get instances with different levels of tightness regarding the duration of maintenance δ, this latter is set to $\lfloor \frac{2 \cdot T}{100} + 0.5 \rfloor$ which is denoted by $\delta = 2\%T$. For example, if T is equal to 150, δ is equal to 3. In order to make it easier to write, we denote the calculated value of T for instances belonging to CL1D0 and CL1D1 by T_g.

Both the greedy heuristics and GVNS but also the whole proposed lower bounds are coded with JAVA. The MILP is solved with CPLEX 12.4. All computational work is performed on an i7 Intel Core at 2.50 GHz and Linux with 8 GB of memory.

3.9.2 MILP performance

The MILP is launched on size instances from n=2 to 20 belonging to each benchmark class. This gives results observed over 1140 test problems.

The time limit of CPLEX is set to 20 minutes. The computational results of the MILP are summarized in Figure 3.4. Each point on the graph corresponds to 60 instances. The time limit of 20 minutes occurs as soon

as the size of instances is greater than 12. Figure 3.4 shows that more instances require more than *MILP_ min total time* as their size increases, this is also confirmed by the average time denoted by *MILP_ average total time*. When $n = 20$, almost all instances reach the 20 minutes time limit (59 instances out of 60 instances) which is confirmed by the average value close to 1200 seconds (1186.84). Due to the NP-hardness of the considered problem, it takes at least 20 minutes to obtain an optimal solution for each instance with size greater than 14. The gaps reported by CPLEX is still about 70% after one hour.

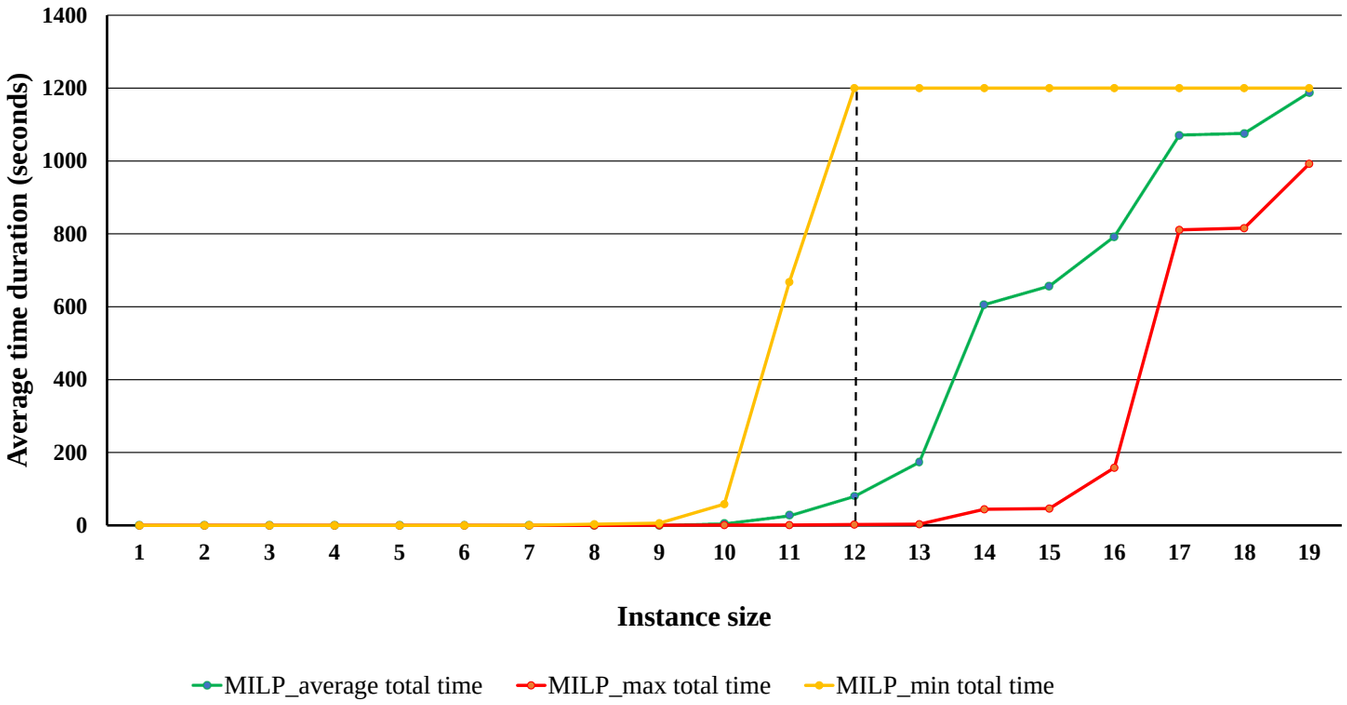

Figure 3.4: MILP performance

3.9.3 Greedy heuristics tests performance

Computational experiments on the proposed greedy heuristics and lowers bounds derived from spacial cases of 1-PMWCT are conducted using 52 different problem sizes, namely n =2,3,4,...,50,100,500 and 1000. Fifteen replications are performed for each instance of size n from each benchmark's class mentioned above. All the results are observed over 3120 test problems.

Lower bounds evaluation

The lower bounds are tested on instances where the number of jobs ranges from 2 to 20 in order to compare them with the results returned by the MILP. The different results of the lower bounds are summarized in Tables 3.6 to 3.9. Each Table refers to the results relating to each class of instances mentioned above. Thus, $Gap_{LB_1}(\%)$, $Gap_{LB_2}(\%)$, $Gap_{LB_3}(\%)$, $Gap_{LB_4}(\%)$ represent the improvement of the relating lower bound compared to the MILP. $Gap_{\bullet}(\%)$ is calculated in the following manner:

$$Gap_{\bullet}(\%) = 100 * \frac{V_{MILP} - V_{\bullet}}{V_{MILP}}. \tag{3.33}$$

Where $V_{\bullet}$ is the value of $LB_{\bullet}$, and V_{MILP} is the best solution found by CPLEX so far. Let us consider the following inequality:

$$LB_{Best} = \max(LB_1,\ LB_2,\ LB_3,\ LB_4) \tag{3.34}$$

n	$Gap_{LB_1}(\%)$	$Gap_{LB_2}(\%)$	$Gap_{LB_3}(\%)$	$Gap_{LB_4}(\%)$	$Gap_{LB_{Best}}(\%)$
2	0.00	18.16	0.00	0.00	0.00
3	0.00	24.72	0.00	2.15	0.00
4	7.89	37.86	0.00	11.24	0.00
5	2.42	36.67	0.00	31.79	0.00
6	3.75	45.97	0.92	46.79	0.92
7	3.27	46.70	0.87	37.70	0.87
8	4.47	48.88	0.72	46.80	0.72
9	3.24	51.43	8.20	53.64	3.24
10	3.79	55.75	0.29	59.38	0.29
11	3.12	52.57	0.03	56.66	0.03
12	2.86	54.50	3.33	58.41	2.86
13	2.70	53.26	0.26	62.70	0.26
14	2.20	51.08	0.54	68.49	0.54
15	2.66	53.53	0.31	62.21	0.31
16	2.54	56.28	1.46	64.19	1.46
17	2.13	57.35	0.01	67.67	0.01
18	2.86	55.92	0.84	71.25	0.84
19	2.34	59.48	0.08	65.39	0.08
20	2.70	58.15	0.14	70.74	0.14

Table 3.6: Average results for lower bounds compared to the MILP: case CL1D0

n	$Gap_{LB_1}(\%)$	$Gap_{LB_2}(\%)$	$Gap_{LB_{3(\%)}}$	$Gap_{LB_4}(\%)$	$Gap_{Best}(\%)$
2	0.00	18.16	0.00	0.00	0.00
3	0.00	24.72	0.00	0.00	0.00
4	7.51	37.86	0.20	11.24	0.20
5	3.15	37.29	0.00	32.46	0.00
6	3.61	45.97	3.61	46.79	3.61
7	3.74	46.97	1.83	37.53	1.83
8	5.67	49.21	0.65	47.54	0.65
9	4.38	51.37	2.18	54.24	2.18
10	4.17	55.96	0.13	59.58	0.13
11	3.72	52.75	0.33	54.88	0.33
12	4.45	55.23	0.06	59.12	0.06
13	4.30	53.95	0.28	63.34	0.28
14	4.00	51.81	2.90	68.58	2.90
15	3.94	53.71	0.53	60.26	0.53
16	4.04	56.83	0.16	64.76	0.16
17	3.70	57.49	0.07	68.20	0.07
18	4.16	55.90	0.67	69.83	0.67
19	3.71	59.45	0.88	65.53	0.88
20	4.00	58.02	0.71	71.14	0.71

Table 3.7: Average results for lower bounds compared to the MILP: case CL1D1

From Tables 3.6 to 3.9 we state that LB_1 and LB_3 are quite tight contrary to LB_2 and LB_4 where the gap is rather high. However, in most cases LB_3 is better than LB_1. In Table 2 (when $\delta = 0$ and $T = T_g$) LB_1 is better than LB_3 about 10%, whereas in the other tables (instances from the three other classes) LB_3 remains the best lower bound. To that end we took $LB_{best} = \max(LB_1,\ LB_3)$, which is used to compute the performances of the heuristics developed in this work.

Tables 3.10, 3.11, 3.12 and 3.13 summarize the average computational time in seconds of each lower bound (LB_1 to LB_4), and for each instance of size n, namely, $n = 10, 15, 20, 25, 30, 35, 40, 45, 50, 100, 500, 1000$ belonging to each benchmark class CLxDy. As can be seen, the average execution time of LB_1, LB_2 and LB_4 is very

n	$Gap_{LB_1(\%)}$	$Gap_{LB_2(\%)}$	$Gap_{LB_3(\%)}$	$Gap_{LB_4(\%)}$	$Gap_{Best}(\%)$
2	0.00	10.02	0.00	0.00	0.00
3	5.44	16.88	0.00	0.00	0.00
4	6.66	17.42	0.00	10.56	0.00
5	10.24	18.98	0.00	19.62	0.00
6	6.03	21.68	0.45	30.33	0.45
7	7.95	25.22	2.27	31.83	2.27
8	5.00	22.87	1.82	37.08	1.82
9	6.79	22.73	0.20	44.41	0.20
10	6.61	23.43	1.38	43.19	1.38
11	4.30	27.01	0.91	47.18	0.91
12	5.97	22.60	2.11	50.03	2.11
13	7.15	25.51	2.01	53.53	2.01
14	5.31	29.22	2.57	54.48	2.57
15	5.55	22.40	3.07	52.92	3.07
16	5.06	24.26	3.14	53.45	3.14
17	5.21	23.51	3.03	57.15	3.03
18	4.70	25.71	2.59	57.54	2.59
19	5.07	31.38	3.86	56.37	3.86
20	5.51	28.04	1.36	58.35	1.36

Table 3.8: Average results for lower bounds compared to the MILP: case CL2D0

n	$Gap_{LB_1(\%)}$	$Gap_{LB_2(\%)}$	$Gap_{LB_3(\%)}$	$Gap_{LB_4(\%)}$	$Gap_{Best}(\%)$
2	0.00	10.02	0.00	0.66	0.00
3	6.12	17.36	0.00	1.22	0.00
4	6.97	17.17	0.00	7.89	0.00
5	10.75	19.01	0.00	20.30	0.00
6	7.69	21.67	0.61	28.48	0.61
7	8.85	23.38	0.17	27.83	0.17
8	6.37	22.60	0.24	35.63	0.24
9	7.69	19.56	2.20	45.00	2.20
10	8.71	23.31	2.56	42.02	2.56
11	6.92	22.80	2.82	46.63	2.82
12	7.07	21.77	3.22	49.00	3.22
13	8.24	23.38	2.38	50.27	2.38
14	7.07	21.51	3.18	54.39	3.18
15	7.24	19.67	4.19	52.50	4.19
16	7.17	18.53	4.36	53.29	4.36
17	7.03	18.47	3.96	57.09	3.96
18	6.28	20.33	3.00	55.41	3.00
19	6.98	20.96	5.41	54.56	5.41
20	6.74	22.36	3.53	57.08	3.53

Table 3.9: Average results for lower bounds compared to the MILP: case CL2D1

close to zero or almost equal to zero for some instance sizes as shown in Table 3.12. As already mentioned in Section in Section 3.7, the complexity of these lower bounds is $O(n\log(n))$. This explains the obtained results, whereas the lower bound LB_3 is a special case of the 1-PMWCT problem that can be solved by dynamic programming. The complexity of the associated algorithm to LB_3 is $O(n \cdot (T - \delta))$. The average execution time of LB_3 is much greater than the remainder of the lower bounds. For example, when $n = 1000$, LB_1, LB_2 and LB_4 do not exceed 1 second regardless of the CLxDy instance classes. In the other hand, the average computational time of LB_3 takes 2 hours and 8 minutes for instances belonging to CL1D1, about 10 minutes for those belonging to CL1D0 and at most 29 minutes 3 seconds for those belonging to CL2D0 and CL2D1. Moreover, the average computational time of LB_3 is still very small for instances of size 100 and 500. To summarize, when $n = 500$, it takes at most 2 minutes and 12 seconds for instances belonging to CL2D1 to find a lower bound. For $n = 100$ the computational time goes down to 0.89 seconds. Finally, for $n \leq 50$ a computational time less than 0.3 seconds is required.

n	10	15	20	25	30	35	40	45	50	100	500	1000
LB_1	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010	0.0150	0.0450
LB_2	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010	0.0150	0.0200	0.0960
LB_3	0.0000	0.0160	0.0310	0.0470	0.0160	0.0470	0.0550	0.1090	0.1160	0.5123	38.8710	605.3520
LB_4	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010	0.0160

Table 3.10: Average computational time of lower bounds: case CL1D0

n	10	15	20	25	30	35	40	45	50	100	500	1000
LB_1	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010	0.0030	0.0050	0.0150
LB_2	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010	0.0020	0.0030	0.0050	0.0080	0.0250
LB_3	0.0050	0.0070	0.0250	0.0480	0.0450	0.0580	0.0680	0.1200	0.1300	0.5770	60.7330	6079.7900
LB_4	0.0010	0.0010	0.0020	0.0020	0.0020	0.0030	0.0040	0.0040	0.0040	0.0080	0.0100	0.0450

Table 3.11: Average computational time of lower bounds: case CL1D1

n	10	15	20	25	30	35	40	45	50	100	500	1000
LB_1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0150
LB_2	0.0010	0.0010	0.0010	0.0020	0.0010	0.0020	0.0020	0.0020	0.0020	0.0020	0.0020	0.0100
LB_3	0.0150	0.0160	0.0310	0.0460	0.0630	0.0780	0.1400	0.1720	0.2840	0.8950	99.9080	1727.9900
LB_4	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0002	0.0003	0.0003	0.0040	0.0060	0.0120

Table 3.12: Average computational time of lower bounds: case CL2D0

n	10	15	20	25	30	35	40	45	50	100	500	1000
LB_1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0050	0.0150
LB_2	0.0010	0.0010	0.0010	0.0020	0.0020	0.0020	0.0020	0.0020	0.0020	0.0020	0.0300	0.1600
LB_3	0.0000	0.0150	0.0320	0.0470	0.0780	0.1400	0.1090	0.2080	0.1610	0.8180	126.6260	1741.9420
LB_4	0.0001	0.0001	0.0001	0.0001	0.0001	0.0002	0.0002	0.0003	0.0003	0.0050	0.0090	0.0220

Table 3.13: Average computational time of lower bounds: case CL2D1

Greedy heuristics computational time comparison

It is important to mention that all the developed heuristics require less than 5 seconds for all the generated instances related to 1-PMWCT problem, even for instances with size $n = 1000$. Thus, tables 13 to 16 summarize the average completion times duration of each heuristic for each instance of size n, namely, 10, 15, 20, 25, 30, 35, 40, 45, 50, 100, 500, 1000 and for each benchmark class CLxDy.

n	H_{WSPTFF}	H_{WSPTBF}	H_{WSPTNF}	H_{DFF}	H_{DBF}	H_{IFF}	H_{IBF}	H_{DNF}	H_{INF}
10	0.0011	0.0006	0.0002	0.0001	0.0003	0.0005	0.0003	0.0003	0.0005
15	0.0015	0.0006	0.0005	0.0005	0.0006	0.0007	0.0006	0.0005	0.0005
20	0.0013	0.0010	0.0008	0.0009	0.0011	0.0009	0.0011	0.0008	0,0011
25	0.0021	0.0015	0.0007	0.0014	0.0015	0.0016	0.0013	0.0005	0.0009
30	0.0025	0.0020	0.0007	0.0015	0.0024	0.0013	0.0008	0.0006	0.0011
35	0.0030	0.0025	0.0009	0.0021	0.0018	0.0011	0.0010	0.0007	0.0013
40	0.0035	0.0037	0.0009	0.0015	0.0018	0.0011	0.0011	0.0009	0.0016
45	0.0043	0.0036	0.0011	0.0011	0.0020	0.0013	0.0014	0.0011	0.0016
50	0.0051	0.0026	0.0021	0.0015	0.0020	0.0015	0.0013	0.0010	0.0024
100	0.0063	0.0053	0.0022	0.0039	0.0078	0.0030	0.0027	0.0018	0.0023
500	0.0502	0.0915	0.0171	0.0167	0.0329	0.0221	0.0396	0.0101	0.0515
1000	0.1019	0.1479	0.0781	0.0417	0.0771	0.0743	0.0513	0.0350	0.3108

Table 3.14: Average computational time of heuristics: case CL1D0

n	H_{WSPTFF}	H_{WSPTBF}	H_{WSPTNF}	H_{DFF}	H_{DBF}	H_{IFF}	H_{IBF}	H_{DNF}	H_{INF}
10	0.0010	0.0003	0.0005	0.0005	0.0003	0.0003	0.0005	0.0002	0.0003
15	0.0015	0.0005	0.0005	0.0005	0.0007	0.0007	0.0006	0.0005	0.0006
20	0.0015	0.0009	0.0008	0.0009	0.0011	0.0010	0.0011	0.0006	0.0011
25	0.0021	0.0013	0.0005	0.0012	0.0017	0.0015	0.0015	0.0005	0.0008
30	0.0025	0.0021	0.0005	0.0017	0.0021	0.0010	0.0010	0.0004	0.0012
35	0.0029	0.0027	0.0009	0.0022	0.0015	0.0011	0.0011	0.0007	0.0011
40	0.0037	0.0035	0.0010	0.0011	0.0017	0.0011	0.0011	0.0009	0.0015
45	0.0043	0.0035	0.0010	0.0014	0.0020	0.0011	0.0013	0.0011	0.0017
50	0.0052	0.0029	0.0017	0.0015	0.0021	0.0015	0.0014	0.0008	0.0025
100	0.0061	0.0050	0.0022	0.0045	0.0075	0.0033	0.0031	0.0023	0.0023
500	0.0502	0.0917	0.0179	0.0174	0.0332	0.0229	0.0381	0.0105	0.0508
1000	0.1037	0.1523	0.0780	0.0412	0.0815	0.0740	0.0517	0.0355	0.3004

Table 3.15: Average computational time of heuristics: case CL1D1

Greedy heuristics evaluation

In this Sub-section, experiments focus on revealing the computational effectiveness of the heuristics. Comparison of the individual heuristics is done at first as shown in Table 3.18. Columns 2 to 10 show how many times each heuristic reaches an optimal solution over the 60 instances.

As can be stated from Table 3.18, the two heuristics H_{WSPTFF} and H_{WSPTBF} perform slightly better than the other heuristics. Indeed, these two heuristics seem to give the same results as they gave the same number of optimal solutions. In addition, as stated in Sub-section 3.9.3, all the developed heuristics presented in this article, reach good solutions in less than 5 seconds for all the generated instances. For this purpose, the heuristic used for the second step of experiments is expressed as follows:

n	H_{WSPTFF}	H_{WSPTBF}	H_{DFF}	H_{DBF}	H_{IFF}	H_{IBF}	H_{DNF}	H_{INF}	H_{WSPTNF}
10	0.0011	0.0005	0.0002	0.0003	0.0002	0.0004	0.0003	0.0002	0.0005
15	0.0011	0.0009	0.0001	0.0009	0.0001	0.0009	0.0001	0.0009	0.0003
20	0.0017	0.0009	0.0005	0.0008	0.0007	0.0009	0.0008	0.0007	0.0007
25	0.0019	0.0010	0.0010	0.0009	0.0011	0.0010	0.0013	0.0009	0.0005
30	0.0023	0.0012	0.0014	0.0015	0.0011	0.0011	0.0008	0.0008	0.0004
35	0.0026	0.0019	0.0017	0.0017	0.0013	0.0008	0.0011	0.0009	0.0004
40	0.0030	0.0024	0.0022	0.0013	0.0009	0.0010	0.0013	0.0011	0.0006
45	0.0037	0.0031	0.0016	0.0011	0.0011	0.0011	0.0017	0.0012	0.0003
50	0.0043	0.0026	0.0015	0.0013	0.0011	0.0012	0.0016	0.0011	0.0007
100	0.0032	0.0026	0.0028	0.0029	0.0029	0.0011	0.0012	0.0014	0.0017
500	0.0423	0.0258	0.0121	0.0374	0.0127	0.0124	0.0067	0.0099	0.0035
1000	0.0860	0.0925	0.0302	0.0219	0.0417	0.0407	0.0234	0.0181	0.0114

Table 3.16: Average computational time of heuristics: case CL2D0

n	H_{WSPTFF}	H_{WSPTBF}	H_{WSPTNF}	H_{DFF}	H_{DBF}	H_{IFF}	H_{IBF}	H_{DNF}	H_{INF}
10	0.0011	0.0005	0.0005	0.0002	0.0002	0.0003	0.0003	0.0002	0.0002
15	0.0014	0.0005	0.0004	0.0005	0.0005	0.0004	0.0006	0.0005	0.0005
20	0.0017	0.0007	0.0006	0.0007	0.0007	0.0007	0.0008	0.0009	0.0009
25	0.0017	0.0011	0.0005	0.0009	0.0010	0.0013	0.0012	0.0012	0.0010
30	0.0022	0.0013	0.0005	0.0016	0.0013	0.0014	0.0012	0.0008	0.0009
35	0.0026	0.0019	0.0008	0.0017	0.0018	0.0011	0.0009	0.0009	0.0009
40	0.0030	0.0023	0.0008	0.0022	0.0011	0.0010	0.0011	0.0010	0.0012
45	0.0034	0.0029	0.0007	0.0015	0.0011	0.0011	0.0013	0.0011	0.0013
50	0.0042	0.0030	0.0008	0.0011	0.0013	0.0011	0.0013	0.0016	0.0013
100	0.0052	0.0033	0.0015	0.0024	0.0031	0.0033	0.0028	0.0011	0.0013
500	0.0422	0.0256	0.0037	0.0123	0.0375	0.0135	0.0135	0.0063	0.0207
1000	0.0863	0.0915	0.0125	0.0299	0.0227	0.0436	0.0432	0.0221	0.1319

Table 3.17: Average computational time of heuristics: case CL2D1

n	H_{WSPTFF}	H_{WSPTBF}	H_{WSPTNF}	H_{DFF}	H_{DBF}	H_{IFF}	H_{IBF}	H_{DNF}	H_{INF}
2	60	60	60	60	60	60	60	60	60
3	60	60	46	46	44	44	60	59	60
4	47	47	13	13	25	25	45	45	45
5	47	47	9	9	17	17	38	38	40
6	20	20	4	4	6	6	16	16	16
7	25	24	8	8	6	6	19	19	19
8	22	22	7	7	5	5	18	18	18
9	23	23	3	3	3	3	10	10	10
10	9	9	3	3	3	3	6	6	6
11	11	11	3	3	3	3	7	7	7
12	10	10	3	3	3	3	4	4	4
13	9	8	2	3	3	4	3	5	4
14	4	4	2	2	2	2	3	3	3
15	4	4	3	3	3	3	4	4	4
16	7	7	3	3	3	3	4	4	4
17	5	5	3	3	3	3	3	3	3
18	3	3	3	3	3	3	3	3	3
19	2	2	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0

Table 3.18: Performance of the heuristics with regard to the number of optimal solutions found for $n = 2$ to $n = 20$ jobs instances

$$H_{best} = \min\left(H_{WSPTBF}, H_{WSPTFF}\right) \tag{3.35}$$

Greedy heuristics versus MILP

In order to evaluate the performance of H_{best}, we compare its performance with that of an optimal solution returned by the MILP for sizes ranging from 2 to 20. The computational results are summarized in Table 3.19. For n= 21, 22,.., 50, 100, 500 and 1000, the comparison is done with LB_{Best}. The results are summarized in Table 3.20.

n	$PED_{CL1D0}(\%)$	$PED_{CL1D1}(\%)$	$PED_{CL2D0}(\%)$	$PED_{CL2D1}(\%)$
2	0	0	0	0
3	0	0	0	0
4	0.70	0.70	0.23	1.23
5	0	0	0.11	0.28
6	1.81	1.81	1.40	1.04
7	1.12	1.79	0.88	1.32
8	0.69	1.27	2.63	2.58
9	2.84	1.94	2.58	2.71
10	1.16	1.17	1.89	1.53
11	1.21	1.44	4.93	3.83
12	3.60	3.84	3.03	3.04
13	3.16	2.68	3.42	4.58
14	3.14	2.99	6.79	5.27
15	2.31	2.43	4.45	3.28
16	2.79	3.06	5.36	3.88
17	2.70	2.71	5.56	3.96
18	3.83	2.81	4.72	3.52
19	5.46	4.61	5.84	5.04
20	3.15	2.43	6.85	4.25

Table 3.19: Computational results of H_{best} for $n = 2$ to $n = 20$ jobs instances

Table 3.19 provides the average percentage error deviation (PED) for instances belonging to each class CL1D0, CL1D1, CL2D0, CL2D1. PED represents the improvement of H_{best} compared with the MILP. It is computed as follows:

$$PED\,(\%) = [(H_{best} - V_{MILP})/V_{MILP}] \times 100 \tag{3.36}$$

It is observed from Table 3.19 that instances belonging to benchmarks with $x = 1$ produce smaller PED than those with $x = 2$. This can be explained by the fact that more jobs could be scheduled per batch in the first case than in the second case. Indeed, when $x = 1$, processing times ranged from 20 to 100. This induces an average processing time of 60 which allows to generate T between 100 and $4 * (60 + 60)/2 = 240$. This gives an average of 4 jobs scheduled per batch. Unlike to this, when $x = 2$, processing times ranged from 25 to 50, so on the average processing time is 37.5. Hence when $T = 100$, no more than 2 jobs in average can be scheduled per batch. Consequently, it can be observed that in the second case, there is a higher probability that the machine has to undergo more preventive maintenances than in the first case. This factor induces a greater number of potential solutions which makes the heuristic less efficient for the instances belonging to $x = 2$.

n	$PER_{CL1D0}(\%)$	$PER_{CL1D1}(\%)$	$PER_{CL2D0}(\%)$	$PER_{CL2D1}(\%)$
21	2.57	3.79	8.25	9.29
22	2.30	2.97	8.74	9.55
23	2.79	4.14	9.50	9.53
24	3.81	4.68	8.67	10.28
25	2.80	3.97	8.26	8.42
26	3.45	4.75	9.11	10.55
27	3.74	4.34	9.19	10.14
28	3.72	5.07	9.34	11.18
29	4.42	5.35	9.65	10.98
30	4.25	5.79	9.20	9.97
31	4.07	4.60	9.86	10.73
32	3.78	4.97	9.38	10.70
33	4.45	5.52	9.79	10.71
34	4.08	5.46	10.12	10.61
35	3.77	4.79	9.77	10.37
36	3.33	5.43	10.37	10.45
37	4.46	5.29	9.28	10.44
38	4.28	5.42	9.80	11.20
39	4.12	4.42	10.23	11.37
40	4.07	5.21	9.76	10.87
41	3.78	5.54	9.28	11.18
42	3.87	5.39	10.34	11.83
43	3.85	5.16	10.25	11.29
44	4.55	5.41	9.36	11.10
45	4.04	5.80	11.04	11.78
46	5.01	6.15	10.29	11.67
47	4.32	5.27	9.86	11.05
48	3.61	5.00	10.78	11.83
49	4.16	5.33	9.73	11.03
50	3.84	5.12	10.51	11.45
100	3.74	5.73	10.80	12.45
500	3.14	5.31	10.37	12.29
1000	2.99	5.11	10.28	12.06

Table 3.20: Computational results of H_{best} for $n = 21$ to $n = 50$ jobs instances,$n = 100$, $n = 500$ and $n = 1000$ jobs instances

Greedy heuristics performances versus lower bounds

Table 3.20 gathers the average percentage error ratio (PER). PER represents the improvement of H_{best} compared with the best lower bound LB_{best}. PER is computed as follows:

$$PER(\%) = [(H_{best} - LB_{best})/LB_{best}] \times 100. \tag{3.37}$$

The comparison of Table 3.20 with Table 3.19 shows that PER is higher than PED. This is clearly related to the comparison done with a lower bound which obviously generates smaller values than those returned by the MILP. PER increases also slightly as the number of jobs increases. As for the cases in Table 3.20, instances belonging to CL2D0 and CL2D1 produce greater PER for the same reasons. Here the average PER is about 6.10 with a minimum of 3.01 and a maximum of 9.26. We can conclude that the heuristics are more efficient for instances belonging to CL1D0 and CL1D1.

Overhead performances

Besides the study of the heuristics performances, we investigate also the increased values of the objective function for the same instances when the duration of maintenance varies from $\delta = 0$ to $\delta > 0$.

For this purpose, Table 3.21 and Table 3.22 report the overhead added to the objective function value of the instances with $\delta = 2\%T$ (classes CL1D1 and CL2D1) compared to those relating to instances with $\delta = 0$ (classes CL1D0 and CL2D0). It represents the extra-cost (in %) induced by the duration of the maintenance actions while increasing δ from 0 to $2\%T$. That overhead is computed in Table 3.21 as follows:

$$OVH_{CLXDY} = 100 \times (V_{MILP(\delta=2\%T)} - V_{MILP(\delta=0)})/V_{MILP(\delta=2\%T)}. \tag{3.38}$$

As there is no computational result of the MILP for instances with n=21, 22... 50, 100, 500, 1000 the overhead is computed with the objective function value of the best heuristic H_{best}. The results are summarized in Table 3.22.

n	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
OVH_{CL1DY}	0.00	0.00	0.00	0.98	0.00	0.66	1.40	1.29	0.48	0.69	1.70	1.70	1.88	1.36	1.58	1.64	1.38	1.43	1.37
OVH_{CL2DY}	0.00	1.23	0.69	0.85	1.95	1.14	1.58	1.07	2.35	2.80	1.24	1.23	1.91	1.83	2.26	1.96	1.68	2.04	4.24

Table 3.21: Average computational results of the overhead for $n = 2$ to $n = 20$ jobs instances

On the basis of the obtained results, we can state that most of the instances are sensitive to the increase of the objective function value from $\delta = 0$ to $\delta = 2\%T$. We conclude that the duration of the maintenance δ has a significant impact on the solutions and on the values of the objective function. As a consequence, the duration of the maintenance activities must not be neglected in real-life/industrial studies.

Impact of Property 2

n	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37
OVH_{CL1DY}	2.60	1.55	2.62	1.35	1.62	2.40	1.30	1.84	1.92	2.64	1.16	1.76	1.96	1.95	1.48	2.82	1.58
OHV_{CL2DY}	1.57	1.42	0.73	1.94	0.95	2.03	1.28	2.25	1.75	1.30	1.20	1.76	1.28	1.19	1.08	0.58	1.50
n	38	39	40	41	42	43	44	45	46	47	48	49	50	100	500	1000	-
OVH_{CL1DY}	1.55	0.86	1.72	2.40	2.20	1.77	1.43	2.13	1.69	1.32	1.84	1.57	1.80	2.36	2.29	2.21	-
OHV_{CL2DY}	1.89	1.38	1.47	2.36	1.91	1.34	2.17	1.20	1.88	1.73	1.38	1.74	1.28	2.01	2.19	2.01	-

Table 3.22: Average computational results of overhead for $n = 21$ to $n = 50$ jobs instances, $n = 100$, $n = 500$ and $n = 1000$ jobs instances

Keeping in mind that the developed heuristics perform much better while adding Property 2, the focus of this part of experiments is to illustrate the improvement in objective function value induced by that Property. For testing purpose, we computed the Gap between the value of the objective function before and after ranking the batches according to W_{B_j}. The computational results are summarized in Table 3.23. As expected, the experiments confirm the influence of Property 2. This impact is quite stable and uniformly close to 0 for small size instances but it is more significant for large sized instances, especially for instances belonging to CL2D0. The average impact is about 10.38 % in CL2D0 and less significant in CL1D0 where the impact is about 0.07%.

n	Gap_{CL1D0}	Gap_{CL1D1}	Gap_{CL2D0}	Gap_{CL2D1}
2	0	0	0	0
3	0	0	0.36	0
4	0	0	4.30	0
5	0	0	3.29	0
10	0	0	9.25	0
15	0	0	8.8	0.15
20	0	0	13.96	0.2
25	0.02	0	12.75	0.14
30	0.02	0	12.28	0.23
35	0	0.01	13.16	0.19
40	0.06	0.04	14.18	0.22
45	0.04	0.01	13.7	0.22
50	0.03	0.03	13.11	0.43
100	0.03	0.02	15.61	0.34
500	0.03	0.03	15.84	0.47
1000	0.03	0.04	15.56	0.50

Table 3.23: Impact of Property 2

3.9.4 GVNS tests performance

In this Sub-section, we present the experiments results conduced on the GVNS metaheuristic and the lower bounds derived from the job splitting technique applied only for CL1D1 benchmark class.

These experiments are conduced using 54 different problem sizes, namely n =2, 3, 4,..., 50, 60, 70, 80, 90 and 100. Fifteen replications are generated for each instance-size n.

Note that, in all the tables presented in this section and for each instance size, we report the average value of fifteen instances. All the tests (except for the MILP) presented in this chapter are done for instances of all

sizes n =2, 3, 4,..., 50, 60, 70, 80, 90 and 100. However, to save space, we don't present each time the results of all instance sizes.

Lower bounds based job splitting evaluation

In this section, we propose to compare the computational results between the three proposed lower bounds based on job splitting and the linear bound, LB_{RP}, obtained by relaxing the integrity constraints for $x_{i,j}$ in the mixed integer programming model. Note that relaxing the integrity constraints for y_{ij} is not reported here because it appears to be useless (i.e. no significant decrease in terms of computational time consuming) similarly relaxing $x_{i,j}$ and $y_{i,j}$ at the same time does not provide a good lower bound quality. Table 3.24 shows the performances of each lower bound obtained by calculating the *Ratio* between the lower bound LB_i and the optimal value of the MILP as follows:

$$Ratio_{LB_i} = 100 * (Value_{LB_i}/Value_{MILP}). \tag{3.39}$$

Entries in the last column represent the results of the tightest value returned among the four lower bounds.

$$LB_{best} = \max\{LB_5, LB_6, LB_7, LB_{RP}\}. \tag{3.40}$$

n	$Ratio_{LB_5}$	$Ratio_{LB_6}$	$Ratio_{LB_7}$	$Ratio_{LB_{RP}}$	$Ratio_{LB^*}$
2	92.06	81.84	99.33	72.53	99.33
3	73.67	75.28	91.04	59.45	91.04
4	54.65	62.14	91.81	72.77	91.81
5	50.50	62.64	80.12	66.33	80.12
6	39.93	54.03	62.45	69.38	69.38
7	36.71	52.84	96.34	69.57	96.34
8	34.96	50.26	88.16	74.03	88.16
9	32.11	48.19	92.76	74.56	92.76
10	29.18	43.97	98.25	74.33	98.25
11	29.50	46.85	86.69	75.27	86.69
12	27.71	44.34	79.16	77.76	79.16
13	27.42	45.63	76.47	77.85	77.85
14	28.41	47.63	69.41	79.36	79.36
15	27.17	45.64	69.32	80.25	80.25
16	24.93	42.80	66.67	81.20	81.20
17	24.30	42.08	60.05	81.52	81.52
18	24.81	43.50	62.94	82.72	82.72
19	22.60	40.13	54.61	83.30	83.30
20	23.38	41.26	48.01	84.27	84.27

Table 3.24: Comparison of lower bounds: LB_5 to LB_7 and LB_{RP}

The calculated ratio clearly shows that LB_7 is much tighter than LB_5 and LB_6. Actually the larger the size of the instances, the greater the gap between the lower bounds (LB_5, LB_6, LB_7) and the optimum value. The lower bound LB_{RP} seems to be better than the others when the size of instances is greater than twelve ($n > 12$)

GVNS performances

In this Sub-section, experiments are carried out to evaluate the performance of the proposed GVNS. Due to the stochastic aspect, we run the GVNS algorithm 10 times. As we already stated, the coefficient of variation is smaller than 0.1%. Consequently only report the best of the 10 replicates. The results of comparisons between the MILP and GVNS, and also between GVNS and LB_{best} are presented in Table 3.25. The entries in columns three and five represent the ratio between the optimal value returned by the MILP ($Value_{MILP}$) and the objective function value returned by the GVNS ($Value_{GVNS}$) respectively, between the lower bound ($Value_{LB_{best}}$) and ($Value_{GVNS}$). It is calculated as follows:

$$Ratio_{MH} = 100 * (Value_{MILP}/Value_{GVNS}). \tag{3.41}$$

respectively,

$$Ratio_{LH} = 100 * (Value_{LB_{best}}/Value_{GVNS}). \tag{3.42}$$

The three last columns report the time consumed upon reaching the reported values returned respectively by the MILP, the GVNS and the lower bounds. It follows that GVNS is very fast (an average of 4.22 seconds for instances with size 100). For instances of size $n \leq 20$, the GVNS is very efficient in comparison with the MILP. The ratio $Ratio_{LH}$ becomes smaller for instance sizes ranging from $n = 11$ to $n = 21$. This is certainly due to the lower bounds performances. Indeed, the gap is practically rather large for these instance sizes (see Table 3.24). This shows that GVNS is able to reach high quality solutions in a very small amount of time.

The relaxed problem is easier to solve in the case of small sized instances than in the case of large instances. On top of this, the computational time in both cases is limited to 20 minutes, so it is more likely that, for large instances, the relaxed MILP is struggling to improve the solution. Hence the returned value of the relaxed MILP is still quite large and relatively close to the solution returned by GVNS. More precisely, when $n > 14$, the best lower bound LB_{best} is equal to the lower bound LB_{RP} based on the relaxation of the integrity constraints of the MILP model. Indeed, via the relaxed MILP, CPLEX solves optimally the problem within the calculation time limit of 20 minutes for instances with $n < 50$. When $n \geq 50$, returned solutions are not necessarily optimal and require more than 20 minutes to get the optimal solutions. Therefore, the value returned by relaxed MILP is closer to the GVNS value than to the optimal solution of the relaxed MILP (LB_{RP}). Empirical experiments for those instances show that the gap returned by CPLEX solver remains 90% (in average for the 15 instances) after 20 minutes. This explains why the calculated gap (line 3, 9 and 15 of Table 3.25) is so high for the large instances.

3.10 Conclusions and perspectives

In this chapter we studied the single machine scheduling problem with periodic maintenance. The objective of our study is to minimize the weighted sum of completion times criterion. First, we have provided a Mixed Integer Linear Programming formulation to solve this problem optimally. Then, we proved the NP-hardness of

n	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
$Ratio_{MH}(\%)$	100	100	100	100	100	100	100	99.88	100	100	99.83	99.93	99.95	99.88	99.89	99.86	99.88	99.88
$Ratio_{LH}(\%)$	99,33	91.04	91.81	80.12	69.38	96.34	88.16	92.64	98.81	86.69	79.03	77.80	79.32	80.15	81.11	81.40	82.62	83.20
$Av.Time_{MILP}(s)$	0.01	0.01	0.02	0.03	0.04	0.07	0.12	0.55	1.21	7.80	57.97	120.79	322.86	754.32	886.72	1067.49	≥ 1200	≥ 1200
$Av.Time_{GVNS}(s)$	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0.02	≈ 0.02	≈ 0.01	$\approx$0.02	$\approx$0.02	$\approx$0.02	$\approx$0.02	$\approx$0.02	0.03
$Av.Time_{LB_{best}}(s)$	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0.01	≈ 0.01	≈ 0.03	0.05	0.08	0.13	0.23	0.41	0.55	0.63	0.85	1.02	1.16	1.33
n	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37
$Ratio_{MH}(\%)$	99.81	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
$Ratio_{LH}(\%)$	84.10	83.62	84.91	85.19	85.60	86.06	86.22	86.45	86.85	87.50	87.41	87.60	88.12	88.50	88.66	89.03	89.01	88.90
$Av.Time_{MILP}(s)$	≥ 1200	≥ 1200	≥ 1200	≥ 1200	≥ 1200	≥ 1200	≥ 1200	≥ 1200	≥ 1200	≥ 1200	≥ 1200	≥ 1200	≥ 1200	≥ 1200	≥ 1200	≥ 1200	≥ 1200	≥ 1200
$Av.Time_{GVNS}(s)$	0.03	0.04	0.04	0.05	0.05	0.06	0.07	0.08	0.08	0.09	0.1	0.12	0.13	0.15	0.18	0.2	0.22	0.24
$Av.Time_{LB_{best}}(s)$	1.65	1.83	2.12	2.52	3.09	3.56	4.21	5.34	5.88	6.78	7.78	8.73	9.95	12.68	12.60	14.71	19.78	23.16
n	38	39	40	41	42	43	44	45	46	47	48	49	50	60	70	80	90	100
$Ratio_{MH}(\%)$	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
$Ratio_{LH}(\%)$	89.64	89.46	89.78	90.14	89.82	90.29	90.67	90.47	90.87	90.94	90.92	91.65	93.21	93.21	95.38	99.76	99.86	95.95
$Av.Time_{MILP}(s)$	≥ 1200	≥ 1200	≥ 1200	≥ 1200	≥ 1200	≥ 1200	≥ 1200	≥ 1200	≥ 1200	≥ 1200	≥ 1200	≥ 1200	≥ 1200	≥ 1200	≥ 1200	≥ 1200	≥ 1200	≥ 1200
$Av.Time_{GVNS}(s)$	0.26	0.27	0.29	0.3	0.32	0.35	0.37	0.39	0.42	0.44	0.48	0.5	0.52	0.65	0.74	1.07	0.94	0.87
$Av.Time_{LB_{best}}(s)$	30.22	27.38	36.65	39.21	52.35	113.43	93.85	176.18	372.50	387.30	456.32	1,138.48	$\geq$1200	$\geq$1200	$\geq$1200	$\geq$1200	$\geq$1200	$\geq$1200

Table 3.25: GVNS performances

this latter. We have also provided some properties that have been used to build several efficient heuristics.

Several lower bounds defined as a relaxation problem of 1-PMWCT are proposed and used to test the efficiency of the developed heuristics. We also developed a GVNS algorithm to solve this NP-hard problem. It was tested and compared with the MILP model for small sized instances and with four lower bounds for instance sizes greater than 20. Three of them are based on job splitting approach and one based on the relaxation of one integrity constraint.

All The proposed methods are tested on large randomly generated data. The result showed that the MILP formulation is efficient on instances with sizes less than or equal to 20 and the proposed heuristics are efficient algorithms to solve large instances. Computational experiments show that the average percentage error of the best heuristic is less than 10. The presented properties have been of great use in designing efficient heuristics capable of solving instances with 1000 jobs. The proposed GVNS is very efficient both in terms of quality of the objective function values and computational time.

In future work, it would be interesting to compare the performance of GVNS with other metaheuristics for solving the considered problem after generalizing it by adding more realistic constraints such as release times, failures, etc. Another extension of this research would be to consider the interval between two consecutive maintenance activities as a decision variable of the problem.

Chapter 4

Single machine scheduling with sequence-dependent setup times and periodic maintenance (1-PMStWCT)

We studied in this chapter two mixed-integer linear programming (MILP) formulations of the 1-PMStWCT problem. These MILPs reach optimal solutions only for small-sized instances in a reasonable amount of time. Hence, to solve large-sized instances a Multistart based GVNS metaheuristic is used to this aim. All the investigated methods are tested on randomly generated data inspired from the work of Chen (2009). Thus, computational experiments comparing the two MILPS, and the proposed GVNS end this chapter.

Contents

4.1 Introduction

This chapter gathers the same work hypothesis than already considered in chapter 4 with, in addition, sequence-dependent setup time constraints. Hence, the objective of this chapter is to compute a schedule that minimizes the WCT criterion, subject to sequence-dependent setup time between the jobs and periodic maintenance. It is assumed that preemption of jobs is not allowed and the maintenance activity generates setup times that depend on the last job scheduled just before doing the maintenance but also on the first job processed just after the end of this latter. Referring to the notation of Graham et al. (1979), we denote this scheduling problem by $1/pm, ST_{sd}/\sum_{i=1}^{n} w_j c_j$.

The work of Chen (2009) is the first attempt to incorporate sequence-dependent setup times in machine scheduling problems with availability constraints. They studied a scheduling problem for a single machine with sequence-dependent setup times and periodic maintenance for real problems arising in textile and manufacturing industries. The authors assumed that a job could be interrupted due to a PM activity and could be resumed without setup if these jobs are already in process. However, they doesn't consider setup time for doing a maintenance activity. Indeed, these assumptions simplified the problem in the sense that inclusion of maintenance activities do not influence the order in which jobs are processed as there is no setup time to perform a PM. However, as the authors pointed out, in practice, even if the job could be resumed, it is necessary to carry out activities in order to prepare the machine for maintenance. Later, after maintenance, the machine should be arranged to continue with the job. These activities, in general, depend on the job in process.

To the best of our knowledge, only the work of Liao and Chen (2003) reports studies on scheduling problems considering both sequence-dependent setup times and a preventive maintenance constraints where the maintenance activity is considered as a job with setup cost that depends on the job processed just before. They considered such a scheduling problem with the objective of minimizing the makespan. They also used a metaheuristic called Greedy Randomized Adaptive Search Procedure to solve the same problem.

Due to the comprehensive survey on scheduling problem with setup time done by Allahverdi in (Allahverdi, 2015) and to the best of our knowledge, no one considered such as a problem in the related literature with the objective of minimizing the weighted sum of completion times.

The remainder of this chapter is organized as follows: The problem is described in Section 4.2 while a literature review related to studied problem is resumed in section 4.4. Then, Section 4.3 presents two mathematical formulations dedicated to solve small sized instances of the problem while Section 4.5 exhibits the two developed approximate methods, namely, a basic heuristic and a sophisticated metaheuristic based GVNS. Computational experiments are then discussed in Section 4.5. Finally, Section 4.6 ends the chapter with conclusions and some perspectives.

4.2 Formal description of 1-PMStWCT

Based on the three field notation $\alpha|\beta|\gamma$ known as the Graham triplet Graham et al. (1979), 1-PMStWCT can be denoted by $1/pm, ST_{sd}/\sum_{i=1}^{n} w_j c_j$. It is stated to be NP-hard since Kirlik and Oguz (2012) proved that the

special case of minimizing the sum of completion times without considering PM activities is NP-hard.

During PM and setup times activities, the machine cannot process any job. Also, the preemption is not allowed, that is, jobs are not resumable. Each job J_j, with a deterministic processing time p_j and a positive weight w_j, ($j \in J = \{1, 2, ..., n\}$) has to be processed without preemption on a single machine that can handle only one job at a time. Furthermore, the machine has to undergo a PM in each period of time T. δ represents the duration of the PM. Let us consider the following research assumptions:

- All the processing times, setup times, weights are known and deterministic,
- c_j and $st_{i,j}$ represent, respectively, the completion time of the job J_j, and the setup time between the two jobs J_i and J_j,
- The setup times st_{ij} are sequence-dependent,
- There is a time interval of length T, between each two consecutive maintenance activities,
- Once maintenance is performed, no breakdown can occur in the production system for the next T units of time.

Figure 4.1 illustrates a graphical representation of a feasible solution of the 1-PMStWCT problem. We denote by PM_k the k^{th} preventive maintenance operation undergone on the machine. Since this latter occurs at each time point $kT - \delta$ ($k \in \mathbb{N}^*$) during the time schedule horizon, an idle time can observed. B_k in Figure 1 refers to the set of jobs scheduled between two consecutive PM activities.

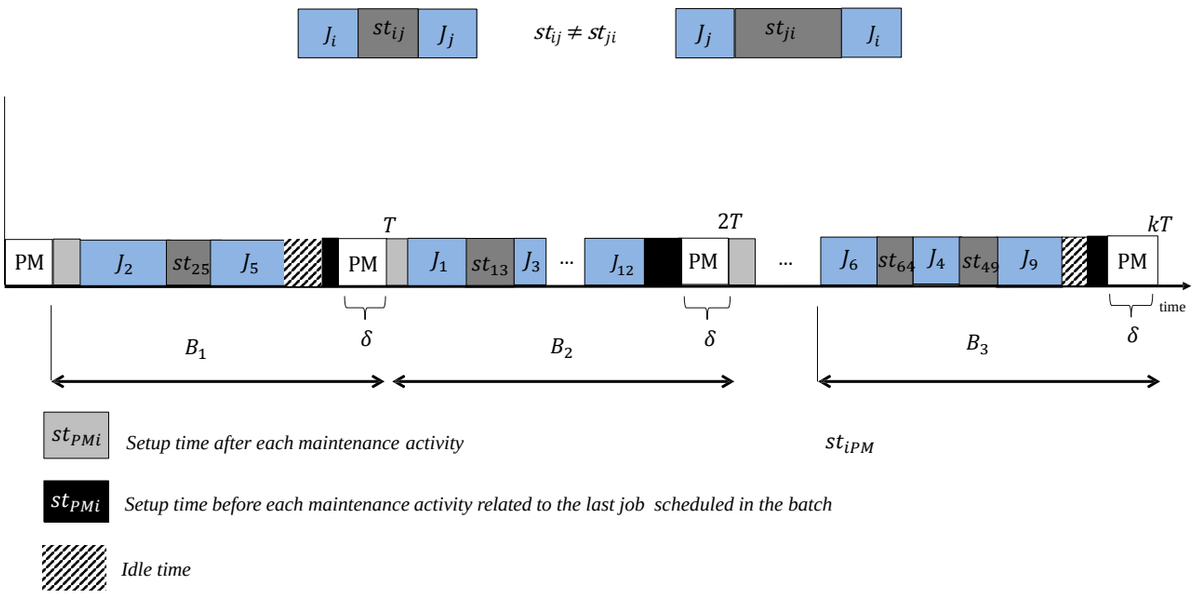

Figure 4.1: Illustration of a feasible solution of the 1-PMStWCT problem

4.3 Literature review

In the following paragraph, we provide a short description of some works related to the scheduling problem considering both sequence dependent setup times and periodic maintenance activity. To our knowledge Chen et al. (Chen, 2009) are the first to consider such a scheduling problem with the objective of minimizing the makespan. In 2011, Ángel-Bello et al. (2011) used a metaheuristic called Greedy Randomized Adaptive Search Procedure also to minimize the makespan. For the same criterion, Naderi et al. (2009) investigated in a job shop scheduling problem and proposed four metaheuristics based on simulated annealing and genetic algorithms to solve the problem. Recently Nesello et al. (2017) proposed new index formulations to solve the problem.

For more details, we direct the reader to consult the review proposed by Allahverdi et al. (Allahverdi, 2015). It's a comprehensive survey which provides an extensive review of about 500 papers that have appeared since the mid-2006 to the end of 2014.

4.4 Mathematical models

4.4.1 First mathematical formulation

The $1/pm, ST_{sd}/\sum_{j=1}^{n} w_j c_j$ problem can be formulated as a mixed integer linear programming model (MILP). The set of jobs scheduled between two periodic maintenance defines a batch. We can have at most n batches. Since maintenance occurs during each period of time T and blocks the machine for a limited time δ and since preemption is not allowed, we can formulate our scheduling problem as the classical BPP (section 3.3) as proposed in the literature (e.g. (Ji et al., 2007), (Hsu et al., 2010), (Benmansour et al., 2014)), where the bins represent batches. Among other things as already explained in Chapter 3, instead of filling bins with items, we fill batches of limited capacity $T - \delta$ by jobs with processing times p_j.

We define the following decision variables by considering the notations given earlier.

$$x_{jk} = \begin{cases} 1 & \text{if job } J_j \text{ is scheduled before job } J_k, \\ 0 & \text{otherwise.} \end{cases}$$

$$y_{jb} = \begin{cases} 1 & \text{if job } J_j \text{ is in batch } B_b, \\ 0 & \text{otherwise.} \end{cases}$$

We define R as a big positive integer. $st_{i,j}$, $st_{(n+1),j}$ and $st_{j,j}$ are respectively the setup time between the two jobs J_i and J_j, the setup time between the maintenance and the job J_j and the setup time before the maintenance related to the job J_j. The matrix of setup times is formulated as follows:

$$ST = \begin{array}{c} \\ J_1 \\ J_2 \\ . \\ . \\ J_n \\ J_{n+1} \end{array} \begin{array}{c} \begin{array}{ccccc} J_1 & J_2 & . & . & J_n \end{array} \\ \begin{bmatrix} st_{1,1} & st_{1,2} & & & st_{1,n} \\ st_{2,1} & st_{2,2} & & & st_{2,n} \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & st_{n,n} \\ st_{(n+1),1} & st_{(n+1),2} & . & . & st_{(n+1),n} \end{bmatrix} \end{array}$$

$$min \quad \sum_{j=1}^{n} w_j c_j \tag{4.1}$$

$$\sum_{j=1}^{n} y_{jb} = 1 \qquad \forall b \in J, \tag{4.2}$$

$$c_j \geq (b-1)Ty_{jb} + \delta + st_{(n+1),j} + p_j \quad \forall (j,b) \in J * J, \tag{4.3}$$

$$c_j \leq jTy_{jb} - \delta - st_{j,j} + R(1 - y_{jb}), \quad \forall (j,b) \in J * J, \tag{4.4}$$

$$c_k \geq c_j + 2st_{j,k} + p_j - R(1 - x_{jk}) - st_{j,k}(\sum_{t=1}^{n} x_{tk} - \sum_{t=1}^{n} x_{tj}) \quad \forall (j,k)_{j \neq k} \in J * J, \tag{4.5}$$

$$x_{jk} + x_{kj} = 1, \ \forall j \in 1..n-1, k \in j+1..n, \tag{4.6}$$

$$c_j \geq \delta + st_{(n+1),j} + p_j \quad \forall j \in J, \tag{4.7}$$

$$x_{jk} \in \{0,1\} \quad \forall (j,k) \in J * J, \tag{4.8}$$

$$y_{jb} \in \{0,1\} \quad \forall (j,b) \in J * J. \tag{4.9}$$

The Equation (4.1) represents the objective function to be minimized. Constraints (4.2) allow each job to be performed once in one of the batches. Constraints (4.3) and (4.7) define in each batch B_b, the execution of any job J_j in this batch. Indeed, each job should be finished before the starting of maintenance activity and its related setup time. This latter should be executed after the end of the preventive maintenance and the setup time spent after the maintenance related to that job J_j ($st_{n+1,j}$). The next sets of constraints, (4.5) and (4.6) indicate that no two jobs, J_j and J_k, scheduled can overlap in time. Finally constraints (4.8) and (4.9) are integrity requirements.

4.4.2 Second mathematical formulation

The main underlying idea of this second formulation is to consider each maintenance as a dummy job, consequently there is no need to another binary decision variable (for batches assignation) as it is considered in the first model.

We assume that there is at most n PMs. Therefore, for a better formulation of the problem, we define the following sets:

$N_1 = \{1, 2, \ldots, n\}$,

$N_2 = \{1, 2, \ldots, 2n\}$,

$N_3 = \{n+1, n+2, \ldots, 2n\}$,

with $n+1, ...2n$ are dummy jobs referring to the maintenance. In the same context, $st_{j,k}$, $st_{m,j}$ and $st_{j,m}$ are respectively the setup time between the two jobs J_j and J_k, the setup time between the maintenance and the job J_j and the setup time before the maintenance related to the job J_j.

The matrix of setup times is designed as follows:

$$ST = \begin{array}{c} \\ J_1 \\ J_2 \\ . \\ . \\ J_n \\ J_{n+1} \\ J_{n+2} \\ . \\ . \\ J_{2n} \end{array} \begin{array}{c} \begin{array}{cccccccccc} J_1 & J_2 & . & . & J_n & J_{n+1} & J_{n+2} & . & . & . & . & J_{2n} \end{array} \\ \left[\begin{array}{cccccccccc} st_{1,1} & st_{1,2} & & & st_{1,n} & st_{1,m} & st_{1,m} & . & . & . & . & st_{1,m} \\ st_{2,1} & st_{2,2} & & & st_{2,n} & st_{2,m} & st_{2,m} & . & . & . & . & st_{2,m} \\ . & . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & .st_{n,n} & st_{n,m} & st_{n,m} & . & . & . & . & st_{n,m} \\ st_{m,1} & st_{m,2} & . & . & st_{m,n} & 0 & 0 & . & . & . & . & 0 \\ st_{m,1} & st_{m,2} & . & . & st_{m,n} & 0 & 0 & . & . & . & . & 0 \\ . & . & . & . & . & . & . & . & . & . & . & 0 \\ . & . & . & . & . & . & . & . & . & . & . & 0 \\ st_{m,1} & st_{m,2} & . & . & st_{m,n} & 0 & 0 & . & . & . & . & 0 \end{array}\right] \end{array}$$

The decision variables are defined as follows:

$$x_{jk} = \begin{cases} 1 & \text{if job } J_j \text{ is scheduled before job } J_i, \\ 0 & \text{otherwise.} \end{cases}$$

$$min \quad \sum_{j=1}^{n} w_j c_j \tag{4.10}$$

$$c_j = (j-n)T \quad \forall j \in N_3, \tag{4.11}$$

$$c_j \leq st_{n+1,j} + p_j + \delta \quad \forall j \in N_1 \tag{4.12}$$

$$c_k \geq c_j + st_{j,k} + p_k - R(1 - x_{jk}), \forall (i,j)_{i<j} \tag{4.13}$$

$$x_{jk} + x_{kj} = 1 \quad \forall (j,k) \in N_2 * N_2, j \neq k \tag{4.14}$$

$$x_{jk} = 1 \quad \forall (j,k) \in N_3 * N_3, \quad j < k \tag{4.15}$$

$$x_{jk} \in \{0,1\} \quad \forall (j,k) \in N_2 * N_2, \tag{4.16}$$

The Equation (4.10) represents again the objective function to be minimized. Constraints (4.11) allow to compute the completion time of dummy jobs. Constraints (4.12) allow each job from the set N_2 to finish after the setup-time. The next sets of constraints, (4.13) and (4.14) indicate that no two jobs, J_i and J_j, scheduled can overlap in time. In the same context, Constraints (4.15) avoid each two scheduled dummy jobs from overlapping in time. Finally constraints (4.16) are integrity requirements.

4.4.3 Comparison of the two formulations

The first and the second MILP formulations (respectively MILP1 and MILP2) presented above differ in the number of variables and constraints as it is shown in table 4.6. On one side in MILP1 there are $2n^2 - n$ variables. Furthermore, there are $3n^2 + n + (n-1)n/2$ constraints. On the other side, MILP2 has $2n^2$ variables and $8n^2 - 2n + (n-1)n/2$ constraints. Due to the slightly greater number of constraints in MILP2 one may assume that it is favourable to use the first formulation (MILP1), which will be demonstrated by the numerical results in section 5.

Formulation	# constraints	# integer variables	# binary variables
MILP1	$3n^2 + n + (n-1)n/2$	n	$2n^2 - 2n$
MILP2	$8n^2 - 2n + (n-1)n/2$	$2n$	$2n^2 - 2n$

Table 4.1: Number of constraints and number of variables for each model.

4.5 Two approximate methods for 1-PMStWCT problem: MS-DSH & MS-GVNS

Since the two proposed MILPs doesn't succeed in solving PMStWCT problem for large-sized instances, we propose to tackle the addressed problem using two heuristics named MultiStart Descent Search Heuristic, denoted by "MS-DSH" that combines greedy and improving phases, and Multistart GVNS where the improving phase of MS-DSH is replaced by GVNS. These two methods are described in Sub-sections 4.5.2 and 4.5.3 respectively.

To solve 1-PMSTWCT, the same move strategy used in Chapter 3 in the context of 1-PMWCT is followed. To do this, we use the same first operators to generate six neighborhood structures. Two others operators are added to the fourth operator Δ_1 to Δ_4 already defined in Chapter 3. Note that the operator Δ_5 is not used to solve 1-PMSTWCT as it does not give satisfying results for 1-PMWCT problem studied in the previous chapter. The two added operators for investigation in this chapter are defined as follows:

$\boldsymbol{\Delta_5}$: The 3-*opt* operator (Lin, 1965) removes three job connections. The three resulting paths (i.e. subsequences of jobs) are put together in a new way, possibly reversing one or more of them. The 3-*opt* is larger

than the 2-*opt* and hence more time-consuming to search.

The complexity for going through the entire neighborhood structure $\mathscr{N}_6(S)$ is $O(n^3)$.

$\mathbf{\Delta_6}$ the *Reverse_two_not_consecutive* operator consists of all solutions obtained from a solution S by interchanging two not consecutive jobs.

The complexity for going through the entire neighborhood structure $\mathscr{N}_7(S)$ is $O(n)$.

In order to respect the principle of the multistarts algorithms, the two developed methods presented in this section are based on an instance generator that we called "*Lehmer code*" denoted by LCG. This algorithm allows MS-DSH and MS-GVNS to restart the algorithm at each iteration from a new initial solution. The LCG concept is introduced in next sub-section.

4.5.1 Lehmer Code Generator: LCG

The Lehmer code is attributed to Derrick Lehmer (Laisant, 1888), (Prouff and Schaumont, 2012), (Lehmer, 1960). It associates a unique code $L(\sigma)$ to each permutation $\sigma = \{\sigma_1, \sigma_2, ..., \sigma_k, ..., \sigma_n\}$. $L(\sigma) = \{l_1, l_2, ..., l_n\}$ is a sequence where l_i is the number of elements that are smaller than σ_i and appear to the right of σ_i in the permutation. The elements l_i in the Lehmer code always satisfy Equation 4.17:

$$0 \leq l_1 \leq n-1, 0 \leq l_2 \leq n-2, ..., 0 \leq l_{n-1} \leq 1, l_n = 0 \tag{4.17}$$

To go back from the Lehmer code to the permutation that generates it, the method used consists in looking for the σ_i elements as explained in the Equation 4.18:

$$\sigma_i = A_i\,[l_i + 1] \text{ with } A_i = J - \{\sigma_1, \sigma_2, ..., \sigma_{i-1}\} \tag{4.18}$$

We point out that a feasible solution S for 1-PMStWCT is defined as any permutation in the set J of predefined job ready to be scheduled at time $t = 0$

In this perspective, we can use LCG to build a new initial solution whenever needed. We note that LCG is implemented in such a way that the probability to construct same solutions when calling LCG several times within a loop, is minimal. This "guarantees" the diversification of the solutions reached in the research space.

4.5.2 Multistart Descent Search Heuristic for 1-PMStWCT

MS-DSH is a Multiple Descent Search heuristic (see Chapter 2 Sub-section 2.5.2) hybridized with greedy heuristics.

Algorithm 9 summarizes the main steps of MS-DSH. It starts by building a solution S using the famous BF heuristic, then improve it by sorting the jobs of S using the WSPT rule. An initial feasible solution is then provided. After that, MS-DSH launches of a descent search heuristic (DSH) until the counter I^{MS-DSH} is reached. At each iteration within the **while** loop, a new initial solution is generated using the Lehmer-Code Generator denoted by LCG and the best solution S^* is recorded accordingly. The neighborhood structure used

within DSH is based on the swap operator denoted by Δ_2. Consequently, the index "2" in DSH^2 refers to Δ_2 operator used within the neighborhood exploration to find a good solution for 1-PMStWCT. The choice of this operator is already explained in the previous Chapter, Section 3.6.3.

Algorithm 9: MS-DSH. Input: J. Output: S^*

$S \longleftarrow$ Ranks jobs of J using Best-Fit heuristic
Apply WSPT in each B_k of S
$S^* \longleftarrow S$
repeat
  $S \longleftarrow DSH^i(S)$
  if $(f(S) < f(S^*)$ **then**
    $S^* \longleftarrow S$
  end
  $S \longleftarrow LCG(S)$
  if $(f(S) < f(S^*)$ **then**
    $S^* \longleftarrow S$
  end
until *(number of iterations* $\geq I^{MS-DSH}$*)*;

4.5.3 Multistart GVNS based metaheuristic for the 1-PMStWCT

Our algorithm, called *MultiStart General Variable neighborhood Search* denoted by **MS-GVNS**, is an iterative algorithm that in each iteration constructs a new feasible solution using *Lehmer code generator* explained above and then, improves it using a **GVNS procedure**. The MS-GVNS is illustrated by Algorithm 10 where S stands for the current solution and S^* for the best solution. This search framework stops when $I^{MS-GVNS}$ is achieved.

Algorithm 10: MS-GVNS. Input: J. Output: S^*.

while *(number of iterations* $\leq I^{MS-GVNS}$*)* **do**
  $S \longleftarrow$ LCG(J)
  $S^* \longleftarrow$ GVNS(S)
end

Proposed implementation of GVNS

Algorithm 11 describes the main steps of the GVNS heuristic developed in this Chapter. It starts by initializing the best solution S^* via the same initial solution procedure WLS used in Chapter 3.

Indeed, the developed GVNS alternates phases of intensification and diversification to reach after L_{max} iterations the best ever visited solution. The WLS procedure is an intensification step where the current solution S is get trapped in a local optimum. The shaking procedure allows to escape from this latter. It allows

to visit other solutions in the search space without going too far from the current solution. In the rest of the **while** loop iterations, the intensification phase is taken over by the VND procedure.

Algorithm 11: GVNS. Input: L_{max}, k_{max}, J. Output: S^*.

Initialization:
$S \longleftarrow WLS(J)$
$S^* \longleftarrow S$
$l \longleftarrow 1$
while *($l \leq L_{max}$)* **do**
  $S^{shake} \longleftarrow$ shaking(S,l)
  if S^{shake} *is better than* S^* **then**
    $S^* \leftarrow S^{shake}$
  end
  $S \leftarrow S^{shake}$
  $S \leftarrow \text{VND}(S, k_{max})$
  if S *is better than* S^* **then**
    $S^* \leftarrow S$
  end
  $l \longleftarrow l + 1$
  if $l \geq 7$ **then**
    $l \leftarrow 1$
  end
end

Shaking procedure

Contrarily to what is proposed in Chapter 3, here the shaking procedure described by Algorithm 12 builds at each iteration within the **while** loop of Algorithm 11 a new solution named S^{shake} using one of the predefined operator introduced in this chapter. Namely, L=$\{\Delta_1, \Delta_2, \Delta_3, \Delta_4, \Delta_5, \Delta_6\}$. Hence, the set L is browsed using the index k. Such as when $k = 2$, the *swap* operator is used and when $v = 5$ the $3-opt$ operator is used. When k is incremented until 7 and L_max is not reached yet, k will restart from the first operator and set to 1.

Algorithm 12: Shaking. Input: S, k. Output: S^{shake}.

Input: (S, k)
Output: S^{shake}
Use the k^{th} neighborhood structure in L to build S^{shake}

VND procedure

The VND proposed within this work, uses the best "improvement strategy".

Three neighborhood structures according to their performance are used: $\mathscr{N}_7(S)$, $\mathscr{N}_2(S)$ and $\mathscr{N}_4(S)$. Hence, k_{max} is set to three. The choice of these latter is justified in Section 4.6. The best solution is provided at the end of each neighborhood search.

The VND procedure takes as inputs the parameter k_{max} and S^{shake} and returns $S^{'}$. If $S^{'}$ is better than the current solution S then it is updated and used in the next iteration to generate a new solution in the search space using the shaking procedure. Otherwise, S will be used once more to generate S^{shake} using the next neighborhood structure throughout the set L. the best solution S^{*} is updated if necessary.

Neighborhood change strategy

Contrary to the strategy proposed when implementing GVNS in Chapter 3 (Section 3.6.3), the neighborhood change strategy used withing VND in this Chapter illustrated by Algorithm 15 consists to use the next neighborhood structure at the first improvement. If there is no improvement, the search restarts from the first neighborhood structure.

We remind that the strategy used in Chapter 3 consists to use the next neighborhood structure if only there is no improvement. VND stops when the last neighborhood structure is used.

Algorithm 13: VND. Input: S, k_{max}. Output: S.

$k \longleftarrow 1$
while $k \leq k_{max}$ **do**
  $S^{'} \longleftarrow argmin_{S'' \in N_{\Delta_k}(S)} f(S'')$
  $S \longleftarrow NeighborhoodChange(S^{'}, S, k)$
end

Algorithm 15: NeighborhoodChange. Input: S, $S^{'}$, k. Output: S.

if $S^{'}$ *is better than* S **then**
  $S \longleftarrow S^{'}$
  $k \longleftarrow k + 1$
end
else
  $k \longleftarrow 1$
end

4.6 Computational experiments

4.6.1 Data generation

The processing times p_j and the setup times st_{ij} are picked up from the work of Chen (2009) which discussed and solved a scheduling problem with dependent setups and maintenance in a textile company. Chen et al stated that the data was generated from the studied company. The percentage distributions of processing times and setup times (in hours) are listed in the Tables 4.2 and 4.3 below: The experiments are based on two levels of T=(720, 1200) hours and two levels of δ=(24, 48) hours. This defines four tests classes, namely, B_1 with (T =720, $\delta = 24$), B_2 with ($T = 720$, $\delta = 48$), B_3 with ($T = 1200$, $\delta = 24$) and finally B_4 with ($T = 1200$, $\delta = 48$). The weights w_j are randomly generated from a discrete uniform distribution in the interval (20, 50) for the four test classes.

Processing	≤ 50	51-100	101-150	151-200	201-250	251-300	301-350	$\geq$351
Percentage	10	14	25	19	18	7	5	2

Table 4.2: Processing time distribution (in hours)

Setup time	≤ 5	6-10	11-15	16-20	21-25	26-30	31-35	$\geq$36
Percentage	26	23	20	12	10	6	2	1

Table 4.3: Setup time distribution

In addition, we consider the following special triangular inequality: $st_{jk} \leq st_{jl} + st_{lk}$ $\forall k, j, l \in J$. In other words, the setup time between jobs j and k is either equal or lower than the setup between job j and any other job l, the processing time of job l and the setup between jobs l and k. Furthermore, setup-times are asymmetric, that is, setup-time between jobs j and k might be different from setup-time between jobs k and j.

The developed algorithms are tested over twenty different job sizes for small sized instances, namely $n = 8, 10, 11, 15, 20$ and 25 and large-sized instances, namely $n = 50, 60, 70, 80, 90$. Both the MS-GVNS and MS-DSH are coded in JAVA language. All the experiments described in this section have been carried out on a i7 Intel Core at 2.50 GHz and Linux with 16 GB of memory.

4.6.2 MILPs performances

To save space, the two MILPs are launched over twenty instances with the same size for each instance of size, namely $n = 8, 10, 15, 20$ and 25. When no optimal solution has been found within the time limit, the gap is reported as a measure of closeness to the optimal cost. The time limit of CPLEX is set to 30 minutes.

The numerical results comparing the two MILPs are summarized in Tables 4.4 and 4.5. Table 4.4 presents for each instance size, the number of solved instances and Table 4.5 summarizes the average time duration in

seconds. For instances that are not solved within the time limit of 30 minutes, the average gap between the lower bound and upper bound computed with CPLEX, is reported.

As it can be seen from Tables 4.4 and 4.5 the results show that the two formulations give almost the same results with a slight superiority of MILP1 for all instances. The two MILPs are able to solve only the instances of smallest size with 8, 10 jobs and some instances with 15 jobs within the calculation time limit of 30 minutes. MILP1 was able to solve optimally 6 instances out of 20 instances of size 15 before the 30 minutes time limit, while MILP2 was able to solve only 4 instances. Still, MILP1 provided much closer gaps and therefore closer approximations of an optimal solution for the larger size instances with 15, 20 and 25 jobs. This is certainly due to the fact that the MILP1 has less constraints and decisions variables than MILP2.

instances	solution status	
size n	MILP1	MILP2
8	optimal 20/20	optimal 20/20
10	optimal 20/20	optimal 20/20
15	06 opt /20	04 opt/20
20	feasible	feasible
25	feasible	feasible

Table 4.4: Comparison of the quality results between the two MILPs

instances	average gap [%]		average solution time [s]	
size n	MILP1	MILP2	MILP1	MILP2
8	0	0	0.25	0.30
10	0	0	2.69	4.29
15	16.81	21.39	1443.26	1531.72
20	52.72	62.79	>=1800	>=1800
25	69.28	76.13	>=1800	>=1800

Table 4.5: Comparison of the computational time results between the two MILPs

4.6.3 Testing local search heuristics

We propose in this Sub-section to compare operators used with six different neighborhood structures that are introduced in Section 4.5. Algorithm 16 describes the main steps of the procedure used to evaluate these six operators. It consists to start the evaluation of each neighborhood structure with an initial solution built by the WSPT rule. This solution is improved using a basic local search heuristic (LS^i), each time with a new operator Δ_i. The main purpose of this approach is to evaluate the speed of each LS towards a local optimum in a minimum amount of time but also which one minimizes the number of calls to the evaluation function.

The local searches are tested on instances with size $n = 90$. Tables 4.6, 4.7 and 4.8 summarize average values of twenty instances, respectively for l^{EVAL} = 8,000 and 200,000 and 10,000,000 objective function evaluations

Algorithm 16: Evaluation Algorithm

Input: l^{EVAL}
Output: S
$S \longleftarrow$ Ranks jobs using WSPT rule
$S^* \longleftarrow S$
while *(number of iterations* $\leq l^{EVAL}$*)* **do**
  $S' \longleftarrow argmin_{S'' \in \mathcal{N}_{\Delta_i}(S)} f(S'')$
  number of iterations$\longleftarrow$ number of iterations + $\mid \mathcal{N}(S) \mid$
  if S' *is better than* S^* **then**
    $S^* \longleftarrow S'$
  end
  $S \longleftarrow S'$
end

and the average computational time in seconds spent by the local searches to reach a local minimum.

LS^i	$Aver_{val}$	$Aver_{time}$
LS^1	31788672.65	0.01
LS^2	33903094.7	0.01
LS^3	34536227.25	0.01
LS^4	33336786.1	0.02
LS^5	34369160.85	0.17
LS^6	33349753.15	0.01

Table 4.6: Local searches comparison for $l^{EVAL} = 8000$ evaluations and $n = 90$

LS	$Aver_{Eval}$	$Aver_{time}$
LS^1	31788672.65	0.18
LS^2	29596498.1	0.16
LS^3	29954994.75	0.18
LS^4	29833772.35	0.31
LS^5	30620641.8	0.34
LS^6	29727712.2	0.19

Table 4.7: Local searches comparison for for $l^{EVAL} = 200{,}000$ evaluations and $n = 90$

Furthermore, Figure 4.1 shows the comparison of the six neighborhood structures. It describes the behavior of each local search. We point out that running the local searches LS^i on the four benchmark families described above allow us to make same observations. In order to save space, only the results related to the test class B_1 ($T = 720$ and δ =24) are reported.

From Figure 4.1 we observe that LS using the swap, the 2-*opt* and *reverse_two_not_consecutive* operators respectively Δ_2, Δ_4, Δ_6 gives solutions of very similar quality with a slight improvement of the swap operators

LS^i	$Aver_{Eval}$	$Aver_{time}$
LS^1	31788672.65	7.86
LS^2	29596498.1	7.40
LS^3	29954994.75	8.98
LS^4	29833772.35	13.90
LS^5	29458185.15	15.35
LS^6	29727712.2	8.69

Table 4.8: Local searches comparison for $l^{EVAL} = 10{,}000{,}000$ and $n = 90$

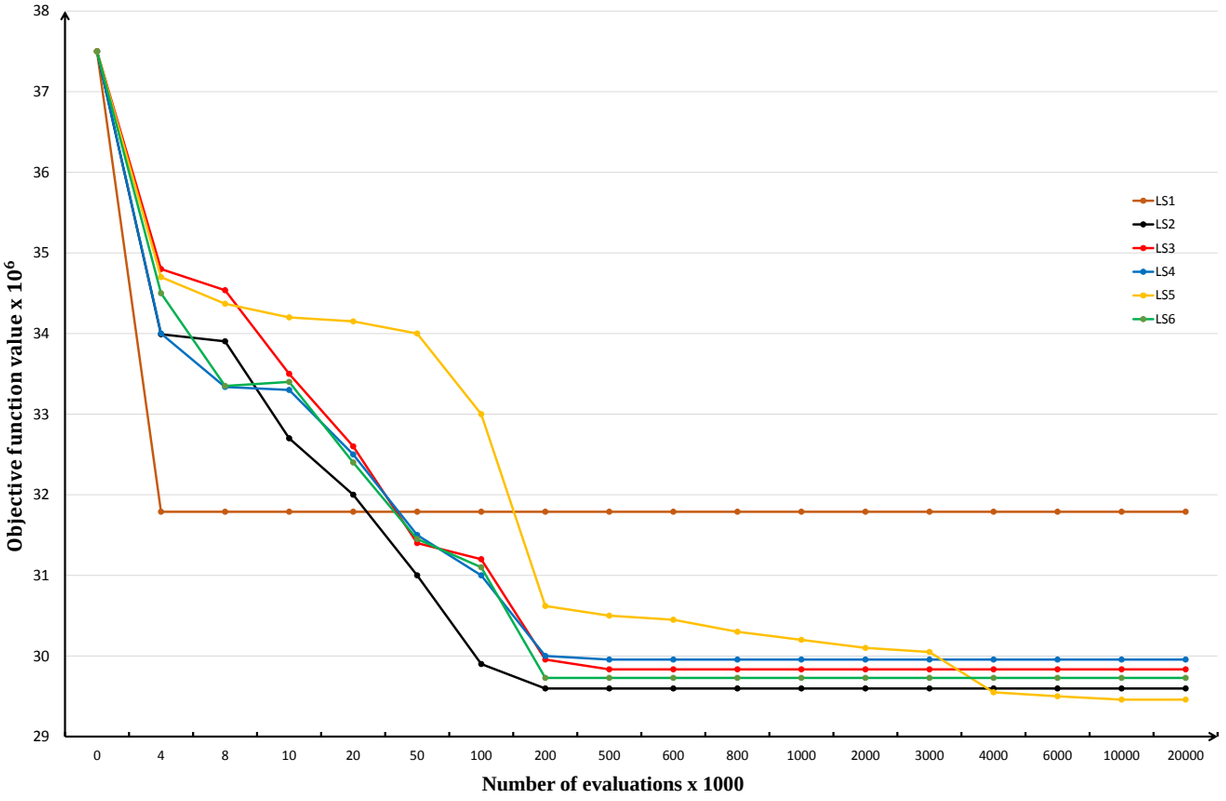

Figure 4.2: Local searches behavior comparison case: B_1

starting from 20,000 evaluations. The LS using the Δ_1 seems to give best solutions when the number of evaluations is less than 50,000 but turns to be the worst when this number exceeds 200,000 evaluations. The local searches using the 3-*opt* (Δ_5) operator gives the worst solutions starting from 8,000 evaluations until 200,000 evaluations but turns to return best solutions when l^{EVAL} is upper to 4,000,000. The time consuming of the tested LS when using Δ_1, Δ_2, Δ_3, Δ_4 and Δ_6 is rather negligent as it is approximately equal to zero. On the other hand, it is slightly higher when using Δ_5.

The results summarized in Tables 4.6, 4.7 and 4.8 consolidates our expectations. Indeed, Table 4.6 shows us that the local search using the Δ_1 operator outperforms the others when the number of evaluations is set to 8000. In the opposite case, when l^{EVAL} is set to 20,000 the Δ_2 operator outperforms the others. The Δ_5 operator becomes the best one when the number of evaluations is set to 10,000,000.

Due to this observations, we decided to implement a local search based on the Δ_6 operator from the initial solution phase (WLS procedure). The Δ_2, Δ_6, and Δ_6 operators are exploited under a first improvement strategy on the VND procedure. As it is usual to explore neighborhoods in the increasing order with respect to their performances (Todosijević et al., 2016), we investigated the three neighborhood structures in this order: $\mathscr{N}_2(S)$, $\mathscr{N}_6(S)$ and $\mathscr{N}_4(S)$.

4.6.4 Parameters calibrations

In this sub-section, we examined the influence of the parameters I^{MS-DSH} and L_{max} within GVNS (how many time we call the shaking procedure). To do this, different values of I^{MS-DSH} and L_{max} are tested, from 1 to 30 for I^{MS-DSH} and from 5 to 25 for L_{max}.

The obtained results are presented in Tables 4.9 and 4.10 for twenty instances of size $n = 90$ for each benchmark class. These instances are of large sizes and could reveal the performance of MS-DSH for a given values of I^{MS-DSH} and L_{max}.

For each value of I^{MS-DSH} and L_{max} the average solution value found and also the average time in seconds spent upon reaching these solutions are presented. As GVNS is a stochastic heuristic, consequently we run this latter 10 times for each instance. The coefficient of variation is less than 0.1%, for that reason, the standard deviation is not reported.

L_{max}	$Av_{value}(B_1)$	$Aver_{value}(B_2)$	$Av_{value}(B_3)$	$Aver_{value}(B_4)$	$Aver_{time}(B_1)$	$Aver_{time}(B_2)$	$Aver_{time}(B_3)$	$Aver_{time}(B_4)$
5	30981188.14	31984517.90	29443131.15	30277027.30	0.63	0.67	0.65	0.63
10	30902881.12	32046247.23	29440960.87	30221723.01	0.99	0.90	0.65	0.63
15	30857030.55	32003400.57	29420660.16	30186067.47	1.07	1.05	0.81	0.97
20	30869505.50	31971156.68	29411506.64	30151778.10	1.12	1.10	0.81	0.86
25	30902494.60	32123555.44	29345290.66	30185935.12	0.94	0.90	0.89	0.90

Table 4.9: Impact of L_{max} on MS-GVNS

From the results presented in Table 4.9, it follows that GVNS is not really sensitive to the value of L_{max}. Indeed, the average solution value found as well as the average time consumed for different values of L_{max} are very close. Due to this, L_{max} is set to 5.

I^{MS-DSH}	$Av_{value}(B_1)$	$Av_{value}(B_2)$	$Av_{value}(B_3)$	$Av_{value}(B_4)$	$Av_{time}(B_1)$	$Av_{time}(B_2)$	$Av_{time}(B_3)$	$Av_{time}(B_4)$
1	31013266.14	32188493.55	29460833.21	30218614.095	0.87	0.84	0.84	0.84
5	30992887.40	32146010.39	29446035.35	30281780.94	3.28	3.22	3.20	3.22
15	30720185.05	29156800.75	29349112.44	30208196.29	9.69	9.53	13.38	14.26
25	30720132.30	29156200.33	29345640.73	30208050.35	15.80	15.30	17.28	18.87
30	30720132.30	29156200.33	29344640.73	30208050.35	21.42	21.98	24.01	26.16

Table 4.10: Impact of I^{MS-DSH} on MS-DSH

From the results presented in Table 4.10, it follows that the proposed MS-DSH is sensitive to the value of I^{MS-DSH} parameter. Indeed, MS-DSH returns best solution values when I^{MS-DSH} is greater than 25. Unsurprisingly, when I^{MS-DSH} is set to 25, MS-DSH consumes the least amount of time compared with $I^{MS-DSH} > 25$: namely, 15.80, 15.30, 17.28 and 18.87 seconds in respectively B_1, B_2, B_3 and B_4. Obviously, we conclude that using LCG generator in MS-LSH significantly improves the results. This can be explained by the fact that DSH is a local search which is quickly trapped in local optima. On the contrary, when using LCG, it allows to escape from local optima by exploring other regions of the search space.

Empirical experiments show as that it is not interesting to set $I^{GVNS-DSH}$ to more than 3 iterations. In this perspectives we fixed $I^{GVNS-DSH}$ to 3. Consequently we conclude that unsurprisingly it is not necessary to use an initial solution generator such as proceeding in MS-GVNS. This can be explained by the fact that the search space is widely explored by GVNS especially due to its shaking procedure. Otherwise, it is observed that using Lehmer Code generator in MS-LSH significantly improves the results. This can be explained by the fact that DSH is a local search which is quickly trapped in a local optimum and when calling LCG it allows to escape from that local optimum by exploring other regions of the search space.

4.6.5 MS-DSH and MS-GVNS performances

In this section we compared the obtained results of the MS-GVNS with the MILP for the small sized instances and with a lower bound LB_{RP}, obtained by relaxing the integrity constraints $x_{i,j}$ in the first mixed integer programming model, for the large sized instances. To measure the performance of both MS-DSH and MS-GVNS, the two notions of PED and PER are used. Thereby, PED_{MS-DSH} and $PED_{MS-GVNS}$, represent the improvement of MS-DSH, respectively MS-GVNS, compared with the MILP while PER_{MS-DSH} and $PER_{MS-GVNS}$ represent the improvement of MS-DSH and MS-GVNS compared with the lower bound LB_{RP}. PED and PER are computed as shown by the two equations 4.19, 4.20.

$$PED\,(\%) = [(V_{heuristic} - V_{MILP})/V_{MILP}] \times 100. \tag{4.19}$$

$$PER(\%) = [(V_{heuristic} - LB_{RP})/LB_{RP}] \times 100. \tag{4.20}$$

Table 4.11 provides computational comparisons between MS-GVNS and MS-DSH using the average percentage error deviation (PED) metric for instances belonging to each class B_1, B_2, B_3 and B_4. The average

computational time spent while reaching best solutions when using MS-DSH, MS-GVNS and CPLEX solver is reported in the three last columns of Table 4.11.

size	PED_{MS-DSH}	$PED_{MS-GVNS}$	Aver-time(MS-DSH)	Aver-time(MS-GVNS)	Aver-time(MILP)
n=8	0.00	0.00	0.03	0.00	0.25
	0.00	0.00	0.02	0.00	0.31
	0.00	0.00	0.01	0.00	0.28
	0.00	0.00	0.01	0.00	0.35
n=10	0.00	0.00	0.04	0.00	4.29
	0.00	0.00	0.04	0.00	14.09
	0.00	0.00	0.06	0.00	17.70
	0.00	0.00	0.05	0.00	21.67
n=11	5.29	0.00	0.07	0.00	38.50
	0.00	0.00	0.05	0.00	114.41
	0.00	0.00	0.07	0.00	84.54
	2.26	0.00	0.06	0.00	159.03
n=15	6.09	5.05	0.10	0.00	1800.00
	8.17	7.56	0.09	0.01	1800.00
	9.72	9.13	0.06	0.00	1800.00
	5.22	3.51	0.08	0.00	1800.00
n=20	7.35	6.53	0.15	0.00	1800.00
	8.77	8.34	0.11	0.00	1800.00
	6.97	7.18	0.11	0.00	1800.00
	9.26	9.06	0.06	0.00	1800.00
n=25	1.72	1.43	0.18	0.01	1800.00
	3.33	2.27	0.12	0.01	1800.00
	2.71	3.34	0.08	0.03	1800.00
	1.97	1.76	0.06	0.05	1800.00

Table 4.11: MS-DSH and MS-GVNS performances for small sized instances

We conclude that MS-DSH succeeds to reach optimal solutions when $n = 8$, $n = 10$ for all the tested instances belonging to B_1, B_2, B_3 and B_4 and only for B_3 and B_4 when $n = 11$. On other hand, MS-GVNS reaches optimal solutions for instance size $n = 8$, $n = 10$ and $n = 11$. It is easy to notice that MS-GVNS produces better results than MS-DSH .

The MS-DSH and MS-GVNS failed to solve 1-PMStWCT when instance sizes are greater than 11, even if CPLEX solver does reach optimal solution for these instances and was stopped after 1800 seconds.

Table 4.12 gathers the average percentage error ratio (PER) metric of both MS-GVNS and MS-DSH. We can observe that the PER is slightly higher when the size of instances n increases. This is obvious since, as n increases, 1-PMStWCT becomes more difficult to solve due the to combinatorial explosion of the search space. We observe that MS-DSH reaches reasonably good solutions in a minimum amount of time. Hence it does not exceed 18 seconds for instances with size $n = 90$. Otherwise, MS-GVNS returns better solutions than MS-DSH

for practicly all instance sizes n.

size	PER_{MS-DSH}	$PER_{MS-GVNS}$	Aver-time(MS-DSH)	Aver-time(MS-GVNS)
n=50	7.73	6.71	0.07	0.21
	13.52	12.28	0.06	0.29
	0.63	0.41	0.05	0.4
	3.38	3.37	0.05	0.35
n=60	8.45	6.88	0.11	3.72
	12.03	13.34	0.15	3.64
	0.96	0.97	0.10	1.02
	2.97	2.84	0.13	1.27
n=70	6.55	6.28	0.31	4.49
	12.97	11.96	0.28	6.08
	1.39	1.37	0.24	6.44
	2.89	3.39	0.34	4.58
n=80	8.43	8.30	0.38	8.59
	13.03	13.34	0.32	11.63
	4.57	4.80	0.33	10.03
	3.00	2.79	0.40	8.57
n=90	8.80	8.50	0.75	16.53
	12.47	13.08	0.62	17.62
	1.59	1.18	0.65	15.65
	3.51	3.57	0.68	15.56

Table 4.12: MS-DSH and MS-GVNS performances for large sized instances

4.7 Conclusion

In this chapter we presented the results obtained when addressing a scheduling problem that combines periodic maintenance and sequence dependent setup-times. The complexity of the problem leads us to design heuristic algorithms to solve it. The two proposed algorithms are composed from two phases, greedy phase and an improvement phase. In this regard, MS-GVNS combines a greedy heuristic and a local search heuristic while, MS-GVNS uses a GVNS metaheuristic instead of a local search.

The computational results showed that the performance of the heuristics was satisfactory in obtaining reasonably good solutions in very short times. However, MS-GVNS gives better solutions than MS-DSH. Therefore, this heuristic can be applied to large-sized instances, where standard commercial solvers fail to find an optimal solution within reasonable computational times.

Chapter 5

A bicriteria two parallel machine scheduling problem with maintenance and jobs rejection

We consider in this chapter a bicriteria scheduling problem on two parallel, non identical machines with a periodic preventive maintenance policy. The two objectives considered involve minimization of jobs rejection cost and weighted sum of completion times. They are handled through a lexicographic approach, due to a natural hierarchy among the two objectives in the type of applications that we consider. The contributions of this chapter are first to develop a mixed integer linear program model for the problem and, secondly, to introduce two metaheuristics based on tabu search. Computational results on randomly generated instances are reported to show the potential savings.

Contents

5.1 Introduction

Scheduling problems have been extensively studied in the literature under the assumption that all jobs have to be processed. However, in many practical cases, one may wish or may be forced to postpone the processing of some jobs, although at some cost. Accordingly, a decision has to be made about jobs that will be accepted and those that will be rejected, so as to produce a good schedule. Nowadays, this situation is observed in several companies with a weekly planning (e.g., pharmaceutical industries, luxury watches, fast moving consumer goods, ship loading). Typically, the rejected jobs will get a higher weight (priority) the next week. The parallel machine scheduling problem has been extensively studied due to its practical applications in various manufacturing systems such as printed circuit board manufacturing, group technology cells, injection molding processes, etc. However, few studies have been done in the context of parallel machine scheduling with jobs rejection.

Preventive maintenance is performed when the machines are idle and, consequently, represents a source of machine unavailability.

In this regard, we address a scheduling problem with two parallel and non-identical machines, denoted as 2-PMJRWCT (2 Parallel Machines problem with Periodic Maintenance, Jobs Rejection and Weighted sum of Completion Times). In this problem, the two machines must undergo periodic preventive maintenance over the scheduling horizon. Solution quality is measured with two criteria. The first one is jobs rejection cost and the second is weighted sum of job completion times. In the latter case, the weights can stand for the holding or inventory cost per time unit of the corresponding jobs as well as their priority level (importance, urgency). A strategy based on Lexicographic Optimization (LO) is proposed to deal with this multi-objective problem. In LO, the decision maker establishes beforehand a priority order among the optimization objectives, where each higher level objective is infinitely more important than any lower level objective. LO is common practice, as reported in (Zykina, 2004); (Ehrgott, 2005), (Thevenin et al., 2017); (Prats et al., 2010) and (Solnon et al., 2008), since it is a convenient approach to address multiobjective problems.

To solve the 2-PMJRWCT, we propose a greedy constructive algorithm and two metaheuristics based on tabu search. We also propose a Mixed Integer Linear Program (MILP) to find the optimum on small-sized instances. It is used here to compare the solutions produced by our algorithms with optimal solutions on instances defined over a scheduling horizon of one day.

The remainder of this chapter is organized as follows. A literature review dealing with order acceptance and scheduling, jobs rejection, periodic maintenance and multi-availability constraints is proposed in Section 5.2. Then, a MILP for the 2-PMJRWCT is presented in Section 5.3. The greedy constructive heuristic and the two metaheuristics based on tabu search are described in Sections 5.4 and 5.5, respectively, while Section 5.6 reports computational results. Finally, Section 5.7 ends the chapter with a conclusion and some perspectives for the future.

5.2 Literature review

Based on the three-field notation $\alpha \mid \beta \mid \gamma$ known as the Graham triplet (Graham et al., 1979), 2-PMJRWCT can be denoted as $P2 \mid pm \mid \sum_{j=1}^{n} u_j, \sum_{j=1}^{n} w_j C_j$. The first field ($\alpha$) means that there are two parallel machines. The second field (β) indicates that a periodic preventive maintenance (pm) must be performed on each machine. Finally, the last field (γ) represents the objective functions. To the best of our knowledge, 2-PMJRWCT has never been studied in the literature. Nonetheless, Subsections 5.2.1 to 5.2.4 will review works that are related to this problem. Subsections 5.2.1 and 5.2.2 are dedicated to the order-acceptance-and-scheduling literature and to the scheduling-problem-with-job-rejection literature, respectively. Both literatures cover actually the same problem, namely job scheduling when the production capacity is not able to schedule all the jobs. This situation leads to the rejection (resp. acceptance) of some of them, which is penalized (resp. rewarded) in the objective function. Subsections 5.2.3 and 5.2.4 focus on the maintenance and on the lexicographic optimization aspects in the context of job scheduling.

5.2.1 Order acceptance and scheduling

A taxonomy and a general review on order acceptance and scheduling (OAS) can be found in (Slotnick, 2011). This problem is to jointly decide about job acceptance and the scheduling of accepted jobs. Different problem characteristics and problem-solving methodologies, starting from this basic scheme, have been proposed in the literature. In the following, papers dealing with a single machine and different objective functions are reviewed, followed by a discussion on problems with two or more machines.

Single machine

Ceyda et al. (2010) consider the single machine scheduling problem where job acceptance depends on the release date, due date, deadline, processing time, sequence-dependent setup time and revenue. The main objective is the maximization of the total revenue. The authors propose a MILP and also develop three heuristic algorithms to solve their problem. Based on the same objective function, Bahriye et al. (2012) propose a tabu search to solve a problem that considers sequence-dependent setup times and tardiness penalties. Nobibon and Leus (2011) generalize two existing problems defined in a single machine environment. The first one is the order acceptance and scheduling problem with weighted-tardiness penalties, already reported in (Slotnick and Morton, 2007). The generalized problem reduces to the latter when the pool of firm planned orders is empty and all jobs can potentially be rejected. The second problem is the total-weighted-tardiness scheduling problem reported in (Potts and Van Wassenhove, 1985). To solve their generalized problem, the authors propose a MILP and two exact branch-and-bound algorithms. They report solving instances with up to 50 jobs in less than two hours. In (Thevenin et al., 2016), the authors address a production scheduling problem in a single-machine environment with earliness and tardiness penalties, sequence-dependent setup times and costs. The objective function includes setup costs, jobs rejection penalties and weighted tardiness penalties. The authors propose various methods to solve this problem, ranging from a basic greedy algorithm to sophisticated metaheuristics (e.g., tabu search, adaptive memory algorithm). In another work by the same authors (Thevenin et al., 2015),

sequence-dependent setup times and setup costs between jobs of different families, release dates, deadlines and jobs rejection are taken into account. They propose and compare a constructive heuristic, local search methods, and population-based algorithms. Recent papers dealing with OAS in a single machine environment take into account machine availability constraints, as in (Zhong et al., 2014). Here, the authors propose a pseudo-polynomial algorithm for fixed time intervals between two consecutive PMs.

Multiple machines

In (Ou and Zhong, 2017), the authors study the OAS problem for n jobs on m parallel machines where the number of rejected jobs should not exceed a given limit L. The objective is to minimize the completion time of the last scheduled job plus the total cost of rejected jobs. For the special case of a single machine, they present an exact algorithm of complexity $O(n \cdot log(n))$. For m machines, they first propose a heuristic of complexity $O(n \cdot log(n))$ with a worst-case bound of $2 - \frac{1}{m}$. They also develop a heuristic based on LP-relaxation and bin-packing techniques. The OAS with two machines in a flow shop is considered in (Xiuli and TCE, 2013). The authors present a heuristic and a branch-and-bound algorithm based on dominance rules and relaxation techniques. Their objective is to maximize the total net profit of accepted jobs, where the latter is the revenue minus the weighted tardiness. In (Wang et al., 2015), the authors solve a scheduling problem with two parallel machines with two heuristics and an exact algorithm, using some properties of optimal solutions to maximize the total profit. In another environment with parallel machines, Jiang et al. (2017) study the OAS problem with batch delivery in a supply chain consisting of a manufacturer and a customer. The objective is to minimize the weighted sum of the maximum lead times of accepted jobs and the total cost of deliveries. To solve the problem, two approximation algorithms are proposed. Finally, Emami et al. (2016) report a MILP model and a Lagrangian relaxation algorithm to solve an OAS problem with the objective of maximizing the total profit.

5.2.2 Scheduling problem with jobs rejection

The scheduling problem with jobs rejection is also reported in the literature, although it can be related to the OAS problem (since rejecting a job means not accepting it and vice-versa). The scheduling problem with jobs rejection was studied in different machine environments, as indicated in a recent survey (Shabtay et al., 2013), although mostly for single machine problems.

In (Li and Chen, 2017), the authors consider the scheduling problem with jobs rejection and a maintenance activity that becomes less effective over time. The main objective is to determine the timing of the maintenance activity and the sequence of accepted jobs to minimize the scheduling cost of accepted jobs plus the total cost of rejected jobs. The authors provide polynomial time algorithms for this problem. Shabtay et al. (2012) propose a bicriteria analysis of a large class of single machine scheduling problems with a common property, namely, the consideration of rejection costs plus other additional criteria (makespan, sum and variations of completion times, earliness and tardiness costs).

Since scheduling with rejection is essentially studied in bicriteria contexts (Shabtay et al., 2013), concepts from the theory of bicriteria scheduling are commonly used when dealing with such problems. Below, we

review papers addressing the weighted sum of completion times and the total cost of rejected jobs. Cao et al. (2006) first prove that the problem for a single machine is NP-hard. A few years later, Zhang et al. (2009) develop a pseudo-polynomial algorithm and a Fully Polynomial Time Approximation Scheme (FPTAS) for multiple parallel machines. Also, Engels et al. (2003) propose general techniques such as linear-programming relaxations. In (Moghaddam et al., 2012), the authors study a single machine scheduling problem with jobs rejection, while considering again minimization of the weighted sum of completion times plus the total cost of rejected jobs. They propose a mathematical formulation and three different bi-objective simulated annealing algorithms to estimate the Pareto-optimal front for large sized instances. The authors in (Zhong et al., 2017) study a scheduling problem on two parallel machines with release times and jobs rejection. The objective is to minimize the makespan of accepted jobs plus the total cost of rejected jobs. They develop a $(1.5 + \varepsilon)$-approximation algorithm to solve the problem. Ou et al. (2015) consider m parallel machines in a context where jobs rejection is allowed. The objective is to minimize the makespan plus the total cost of rejected jobs. They develop a heuristic of complexity $O(n \cdot log(n) + n/\varepsilon)$ to solve the problem with a worst-case bound of $1.5 + \varepsilon$. With the same goal, Zhong and Ou (2017) present a 2-approximation algorithm with a complexity of $O(n \cdot log(n))$ by making use of specific data structures. The authors also propose a PTAS to solve the problem. In (Ma and Yuan, 2016), the authors consider that the information about each job, including processing time, release date, weight and rejection cost, is not known in advance. They develop a technique named Greedy-Interval-Rejection to produce good solutions. Finally, the authors in (Agnetis and Mosheiov, 2017) consider the minimization of the makespan in a flow shop with position-dependent job processing times and jobs rejection. A polynomial time procedure is proposed to solve this problem.

5.2.3 Periodic maintenance and multi-availability constraints

The authors in (Kaabi and Harrath, 2014) have written a comprehensive survey about scheduling in parallel machine environments in the presence of availability constraints (which can be induced, in particular, by maintenance activities). Sun and Li (2010) consider two problems. In the first problem, they minimize the makespan on two parallel machines when maintenance activities are performed periodically. In the second problem, maintenance activities are determined jointly with job scheduling, while minimizing the sum of the job completion times. They introduce an algorithm of complexity $O(n^2)$ and show that the classical Shortest Processing Time algorithm (SPT) is efficient for the second problem with a worst-case bound less than or equal to $1 + 2 \cdot \sigma$, where $\sigma = t/T$, T is the maximum continuous working time for each machine and t is the time required to perform each maintenance activity. Li et al. (2017) investigate a parallel-machine scheduling problem where each machine must undergo periodic maintenance. The authors propose two mathematical programming models and two heuristic approaches to address instances of large-sized. In (Qi et al., 2015), the authors investigate a scheduling problem on a single machine with maintenance, in which the starting time of the maintenance is given in advance but its duration depends on the previous machine load.

5.2.4 Multiobjective scheduling problem using lexicographic optimization

LO was widely used in control engineering and scheduling applications (Aggelogiannaki and Sarimveis, 2006; Kerrigan and Maciejowski, 2002; Ocampo-Martinez et al., 2008; Respen et al., 2016; T'kindt and Billaut, 2006). In a work closely related to ours, the authors in (Thevenin et al., 2017) model a parallel machine scheduling problem with job incompatibility through an extension of the graph coloring problem. Different objectives like makespan, number of job preemptions and total time spent by the jobs in the production shop are considered and addressed through LO. A mathematical model, two greedy constructive algorithms, two tabu search methods and an adaptive memory algorithm are proposed to solve the problem.

5.3 Mathematical model

In the following, we first introduce some notation and a brief description of our problem. This is followed by the MILP.

5.3.1 Formal description of 2-PMJRWCT

Let J be a set of n independent jobs to be scheduled on two parallel, non identical machines M_i (where $i \in I = \{1, 2\}$) over a planning horizon of five days (i.e., 7200 minutes). Accordingly, we define $\tilde{d} = 7200$ minutes as the common deadline for all jobs in set J. If a job cannot be feasibly scheduled during the current week, it is then postponed to the next week and a rejection cost is incurred. A feasible solution of 2-PMJRWCT is illustrated in Figure 5.1, where S (defined as the set of accepted jobs) is made of two sequences: S_1 associated with machine M_1 and S_2 associated with machine M_2. $\overline{S}$ is the set of rejected jobs that are postponed to the next week. Each machine M_i must undergo a PM at intervals that cannot exceed T_i minutes. In other words, the interval between the end time of a given PM and the start time of the next PM cannot exceed T_i minutes. The jobs scheduled between the two PMs define a block, with B_k^i the k^{th} block on machine M_i scheduled between the $(k-1)^{\text{th}}$ and k^{th} PMs. The scheduling of a PM activity on each machine M_i is flexible and can actually occur before T_i minutes have elapsed, if it is not possible to avoid it or if it is beneficial to do so. Accordingly, the time length of a block is also variable, although it can never exceed T_i. The duration of a PM activity on machine M_i is denoted by δ_i. As illustrated in Figure 5.1, there is no idle time in the schedule between two consecutive jobs or between a job and a PM.

Each job $j \in J$ is characterized by a known processing time p_j, a rejection cost u_j and a weight $w_j = h_j + b_j$ which is the sum of its inventory cost h_j and its priority level b_j. Preemption is not allowed and any interrupted job due to maintenance is redone or postponed to the next week. The two machines are considered non identical in the sense that machine M_2 is assumed to be older than M_1. Accordingly, PMs must be done more frequently on M_2 and the maximum time interval T_2 between two consecutive PMs is smaller than T_1. A feasible solution to 2-PMJRWCT is represented by the pair $(S, \overline{S})$. It is evaluated first through objective f_1, which is the total rejection cost of the jobs in $\overline{S}$, and second through objective f_2, which is the weighted sum of the completion times of the jobs in S. With regard to f_2, the WSPT (Weighted Shortest Processing Time) rule is particularly

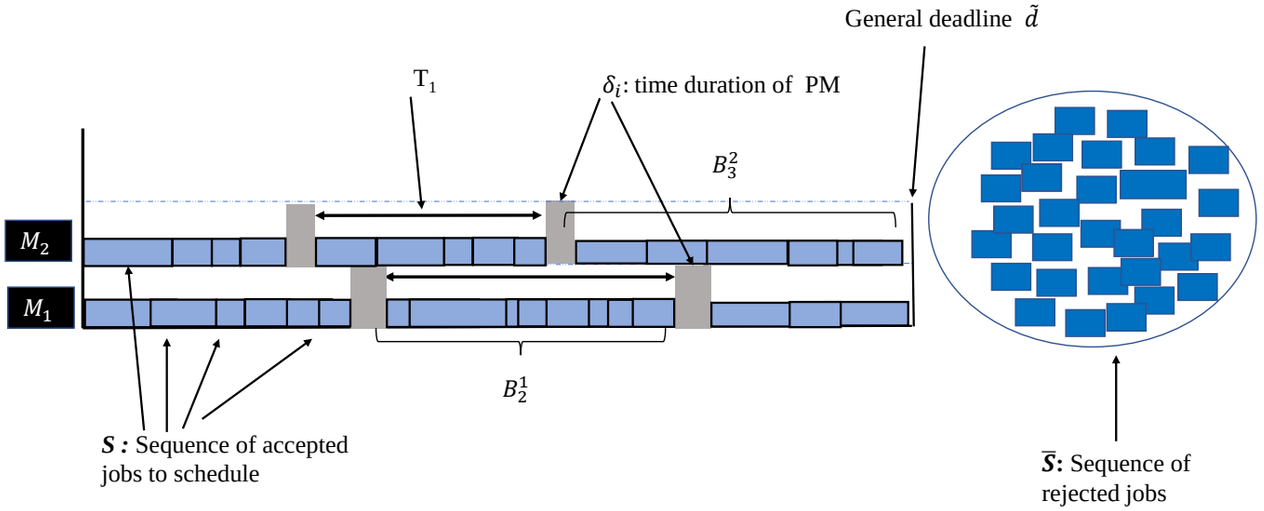

Figure 5.1: Feasible solution of 2-PMJRWCT

important, because it optimally solves the $1 \mid\mid \sum_{j=1}^{n} w_j C_j$ scheduling problem, which minimizes the weighted sum of completion times on a single machine without side constraints. The WSPT rule states that the jobs should be scheduled in increasing order of their ratio p_j/w_j. This rule will be exploited in our algorithms, although in a heuristic way since we have two machines with some operational constraints.

5.3.2 Mathematical model

The mathematical programming formulation of 2-PMJRWCT is presented below. It involves five different types of decision variables.

$$
\begin{aligned}
c_j &: \text{completion time of job } j \in J \\
m_i^k &: \text{starting time of the } k^{\text{th}} \text{ PM on machine } M_i \\
x_{lj}^i &= \begin{cases} 1 & \text{if job } l \text{ is scheduled before job } j \text{ on machine } M_i \\ 0 & \text{otherwise} \end{cases} \\
z_j^i &= \begin{cases} 1 & \text{if job } j \text{ is scheduled on machine } M_i \\ 0 & \text{if job } j \text{ is rejected} \end{cases} \\
y_{jk}^i &= \begin{cases} 1 & \text{if job } j \text{ is scheduled in block } k \text{ on machine } M_i \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

For the sake of the MILP formulation, two dummy jobs 0 and $n+1$ are added to the model with completion

times $C_0 = C_{n+1} = 0$. Accordingly, we define the set $J^+ = J \cup \{0, n+1\}$. We also introduce a block 0 on each machine with $m_1^0 = m_2^0 = 0$. Note finally that M is an arbitrary large number.

$$\min (f_1) = \sum_{j=1}^{n} u_j(1 - (z_j^1 + z_j^2)) \tag{5.1}$$

$$s.t. \quad m_{k-1}^i \leq m_k^i \leq m_{k-1}^i + T_i \quad \forall k = 1, \ldots, n, \ \forall i \in I \tag{5.2}$$

$$m_k^i \geq (\tilde{d} - \delta_i) \Rightarrow (m_k^i = 0) \quad \forall k \in J, \ \forall i \in I \tag{5.3}$$

$$c_j \leq m_b^i + M(1 - y_{jb}^i) \quad \forall i \in I, \forall j \in J, \ b = 2, \ldots, n \tag{5.4}$$

$$c_j - p_j y_{jb}^i \geq m_{b-1}^i + \delta_i - M(1 - y_{jb}^i) \quad \forall i \in I, \forall j \in J, \ b = 2, \ldots, n \tag{5.5}$$

$$\sum_{j=1, k \neq j}^{n+1} x_{kj}^i = z_k^i \quad \forall i \in I, \ k = 0, \ldots, n \tag{5.6}$$

$$\sum_{k=0, k \neq j}^{n} x_{kj}^i = z_j^i \quad \forall i \in I, \ j = 1, \ldots, n+1 \tag{5.7}$$

$$x_{kj}^1 + x_{kj}^2 \leq 1 \quad k = 0, \ldots, n \quad j = 1, \ldots n+1 \tag{5.8}$$

$$c_j \geq p_j(z_j^1 + z_j^2) \quad \forall j \in J, \ \forall i \in I \tag{5.9}$$

$$c_j \leq \tilde{d}(z_j^1 + z_j^2) \quad \forall j \in J, \ \forall i \in I \tag{5.10}$$

$$c_k \leq c_j - p_j x_{kj}^i + \tilde{d}(1 - x_{kj}^i) \quad \forall i \in I, \ k = 0, \ldots, n \quad j = 1, \ldots, n+1 \tag{5.11}$$

$$\sum_{j=1}^{n} p_j y_{jb}^i \leq T_i \quad \forall b \in J, \ \forall i \in I \tag{5.12}$$

$$\sum_{b=1}^{n} y_{jb}^i = z_j^i \quad \forall j \in J, \ \forall i \in I \tag{5.13}$$

$$y_{jb}^1 + y_{jb}^2 \leq 1 \quad \forall (j, b) \in J \times J \tag{5.14}$$

$$z_j^1 + z_j^2 \leq 1 \quad \forall j \in J \tag{5.15}$$

$$z_k^i + z_j^i \geq 2(x_{kj}^i + x_{jk}^i) \quad \forall (k, j) \in J \times J, \ \forall i \in I \tag{5.16}$$

$$C_0 = C_{n+1} = m_0^1 = m_0^2 = 0 \tag{5.17}$$

$$x_{kj}^i, \ y_{jb}^i, \ z_j^i \in \{0, 1\} \tag{5.18}$$

Equations (5.1) and (5.19) are the first and second objective functions considered in this work. Constraints (5.2) allow to compute the starting time of the k^{th} PM on each machine M_i. When $\tilde{d}$ is reached, constraints (5.3) set to zero the starting time of the k^{th} PM on each machine M_i. Constraints (5.4) and (5.5) forbid a scheduled job and a PM activity to overlap. Constraints (5.6) indicate that every job assigned to machine M_i, including the dummy job 0, should have a successor. Constraints (5.7) state that if a job j is assigned to machine M_i, at least one job should immediately follow, which includes the dummy job $n+1$. Constraints (5.8)

state that two jobs scheduled consecutively should be assigned to the same machine. Constraints (5.9) and (5.10) define bounds on the completion time of each scheduled job (if a job is rejected, its completion time is set to 0). Constraints (5.11) indicate that two jobs scheduled on the same machine cannot overlap. Constraints (5.12) state that the sum of processing times of all jobs between two consecutive maintenance activities must be less than or equal to T_i. Since the number of blocks on each machine M_i is at most the number of jobs assigned to M_i, constraints (5.13) computes the number of accepted jobs in each block B_k^i. Constraints (5.14) enforce each accepted job j to be scheduled either on M_1 or M_2 but not both. Similarly, constraints (5.15) allow accepted jobs to be scheduled in a block B_k^i of either machine M_1 or M_2 but not both. Constraints (5.16) state that if two jobs l and j are scheduled on the same machine, then l is scheduled either before or after j. Constraints (5.17) set the completion times of dummy jobs 0 and $n+1$, and the starting time of the first PM on each machine to 0. Finally, constraints (5.18) define the binary variables.

Through the lexicographic approach, 2-PMJRWCT can be solved optimally in two steps with an exact solver. First, objective f_1 is minimized while ignoring f_2 (i.e., the above-presented model is solved). Let $f_1^\star$ value of f_1. In the second step, constraint $f_1 \leq f_1^\star$ is added to the model and the latter is solved with objective f_2 only. In other words, the model below is solved.

$$\min\,(f_2) = \quad \sum_{j=1}^{n} w_j c_j \tag{5.19}$$

$$s.t. \qquad \text{Constraints } (5.2 - 5.18) \tag{5.20}$$

$$\sum_{j=1}^{n} u_j(1 - (z_j^1 + z_j^2)) \leq f_1^\star \qquad \forall\, i \in \{1, 2\} \tag{5.21}$$

The solution obtained at the end of the second step is one optimal solution of 2-PMJRWCT. We observed that CPLEX solver could only solve small instances, thus supporting the use of heuristics and metaheuristics to address instances of larger sizes. In the following, our problem-solving methods are presented, starting with a greedy heuristic to generate a first feasible schedule, which is then improved with tabu search-based metaheuristics.

5.4 Greedy heuristic (GrH) for the 2-PMJRWCT

The greedy heuristic GrH calls a construction procedure which is aimed at producing a feasible schedule of good quality from a given set of jobs. In particular, GrH calls the construction procedure within a loop where the set of jobs is gradually reduced until all jobs can be scheduled, as it is explained below. In each proposed procedure of this work, ties are broken randomly if no other information is provided.

5.4.1 Main procedure

We can see from the description in Algorithm 17 that GrH starts by using the greedy construction procedure (presented in Algorithm 18) with the whole set of jobs J. The latter then returns a feasible solution (i.e., a set S with $|S|$ scheduled jobs and a set $\overline{S}$ of rejected jobs). Forgetting about the schedule previously obtained, we just select the $|S|$ jobs in J with the largest u_j to obtain a subset J_s (step 3a). The construction procedure is then called again, but with J_s (step 3b). If the feasible schedule obtained does not contain all jobs in J_s, we reset the value of $|S|$ to the (smaller) number of scheduled jobs in the new solution, we select the $|S|$ jobs in J with the largest u_j to obtain a new subset J_s (step 3a again), and the construction procedure is called with the latter (step 3b again). This is repeated until all jobs in J_s can be feasibly scheduled. The aim of this step 3 is to schedule as many jobs as possible with the largest rejection costs, since f_1 is the main objective.

Next (step 4), we try to insert the jobs that are not in the solution, by inserting them at the end of the schedule of machine M_1 or M_2. We consider these jobs one by one in decreasing order of rejection costs. First, we check if the current job j can be inserted without exceeding the deadline $\tilde{d}$. If this is not possible for the two machines, job j is skipped. Otherwise, we check if the completion time C_j of j occurs after the due time for the next PM on the single machine or on the two machines that still qualify. If it does, the job is skipped. Otherwise, we schedule job j on the machine that leads to the smallest C_j.

Algorithm 17: GrH. Input: J. Output: $(S, \overline{S})$.

---

1. Set $S \longleftarrow$ Construction(J)
2. Initialization: set $J_s \longleftarrow J$
3. Repeat until all jobs in J_s are scheduled:
   (a) Set J_s as the set of $|S|$ jobs $j \in J_s$ with the largest u_j
   (b) Set $S \longleftarrow$ Construction(J_s)
4. For each job $j \in \overline{S}$ (taken in decreasing order of u_j)
   (a) Try to schedule job j on M_1 or M_2 (favoring the minimization of C_j)
   (b) If j has been scheduled in the previous step, update S and $\overline{S}$

5.4.2 Construction procedure

The construction procedure produces a sequence of scheduled jobs on both machines M_1 and M_2 in a greedy way. Starting with empty sequences on both machines and a set Q of jobs provided in input, a new job is selected and inserted, at each iteration, at the end of the schedule of M_1 or M_2, according to some heuristic rules, until there are no more jobs or until the schedules of both machines are full. The construction procedure is described in Algorithm 18.

The selection of the next job is done in two steps. First (steps 2a to 2c), we consider the set Q_1 of the q_1 (parameter $< n$) jobs with the highest w_j/p_j ratio, which is a good heuristic rule with regard to objective f_2. Next, we compute the completion time C_j^i of each $j \in Q_1$ if it is scheduled on each machine M_i (while programming a PM before job j if necessary). If the deadline $\tilde{d}$ is exceeded, we simply set $C_j^i = \infty$. Each feasible job (i.e., satisfying the deadline) is then tentatively assigned (but not scheduled) to the machine leading to the smallest completion time. In a second step (steps 2d to 2f), we select the subset $Q_2 \subset Q_1$ containing the q_2 (parameter $< q_1$) feasible jobs with the smallest completion times. For each job $j \in Q_2$, we compute the slack time with the due time of the next PM, and we finally select the job $j^\star$ with the smallest slack time. The latter is then inserted (as well as its possible preliminary PM) at the end of the schedule of its assigned machine. It should be noted that all jobs are considered in the first and second steps when the number of remaining jobs is smaller than q_1 and q_2, respectively.

Algorithm 18: Construction. Input: Q. Output: S.

---

1. Initialization: set $S \longleftarrow \emptyset$
2. While $Q \neq \emptyset$, do
   (a) Select the subset $Q_1 \subset Q$ (of size q_1) of jobs with the highest ratio w_j/p_j
   (b) For each job $j \in Q_1$, do
      i. For each $i \in \{1, 2\}$, do: compute the completion time C_j^i of job j if it is scheduled on machine M_i, while programming a PM before j if necessary (set $C_j^i = \infty$ if $\tilde{d}$ is exceeded)
      ii. Assign j to machine M_{i_j} such that $i_j = arg\min_i(C_j^i)$
      iii. Define j as unfeasible if $C_j^i = \infty$ on its assigned machine
   (c) STOP the algorithm if all the jobs are unfeasible
   (d) Select the subset $Q_2 \subset Q_1$ (of size q_2) of feasible jobs with the smallest completion times (select all the feasible jobs of Q_1 if there are less than q_2 feasible jobs remaining in Q_1)
   (e) Select the job $j^\star \in Q_2$ with the smallest slack time (with respect to the beginning of the next PM on its assigned machine), and insert it (along with its possible above-programmed PM) at the end of the schedule of machine $M_{i_{j^\star}}$
   (f) Set $Q \longleftarrow Q \backslash \{j^\star\}$, $S \longleftarrow S \cup \{j^\star\}$

5.5 Tabu Search-based metaheuristics

Two metaheuristics based on tabu search (TS) called Multi-Tabu Search (*MultTS*) and Consistent Tabu Search (*ConsTS*), are proposed in this section. *MultTS* explores only feasible solutions by exploiting different types of neighborhoods, while *ConsTS* admits moves leading to infeasible solutions, although an infeasible solution is immediately repaired to restore feasibility (consistency). These two metaheuristics will now be described in the following sub-sections.

5.5.1 Multi-Tabu Search

A feasible solution is represented by a pair (S, $\overline{S}$), where S is the sequence of scheduled jobs made of the two subsequences S_1 and S_2 on machines M_1 and M_2, respectively, whereas $\overline{S}$ is the unordered set of rejected jobs.

MultTS improves the initial starting solution produced by the greedy heuristic *GrH*, while always maintaining feasibility. As shown in Figure 5.2 and Algorithm 19, *MultTS* has four different phases. Phases 1, 2 and 3 exploit the intensification capabilities of tabu search by using different types of neighborhood structures, while Phase 4 is aimed at diversification.

In Algorithm 19, (S, $\overline{S}$) stands for the current solution, while (S^*, $\overline{S^*}$) is the best known solution. The **while** loop stops the algorithm after $IterMax^{MultTS}$ global iterations and returns the best solution found (S^*, $\overline{S}^*$). Each iteration involves three consecutive TS phases, each with a given neighborhood structure. Phase 1 optimizes objective f_1 and the set $\overline{S}$ of rejected jobs obtained at the end of this phase is directly fed to Phase 4. Meanwhile, the sequence S of scheduled jobs obtained at the end of Phase 1 is modified in Phases 2 and 3 with the aim of optimizing f_2. In these two phases, no job from S can be rejected, thus only the sequence of jobs on the two machines is modified. Phase 4 is aimed at diversifying the search by rescheduling the jobs to obtain a new partition between scheduled and rejected jobs. The main components of *MultTS* will now be described in the following paragraphs.

Algorithm 19: MultTS. Input: J. Output: ($S^\star$, $\overline{S}^\star$).

---

1. Initialization: set (S, $\overline{S}$) $\longleftarrow GrH(J)$
2. For ($t = 1$ to $IterMax^{MultTS}$), do:
   (a) **Phase 1:** set (S, $\overline{S}$) $\longleftarrow Tabu((S, \overline{S})$; $SWAP^1$; $INSERT^1)$
   (b) **Phase 2:** set (S, $\overline{S}$) $\longleftarrow Tabu((S, \overline{S})$; $SWAP^2)$
   (c) **Phase 3:** set (S, $\overline{S}$) $\longleftarrow Tabu((S, \overline{S})$; $SWAP^3)$
   (d) **Phase 4:** if $\big((NB_1(S) > NB_1^\star)$ and $(NB_2(S) > NB_2^\star)\big)$, set $(S, \overline{S}) \longleftarrow Diversification(J)$

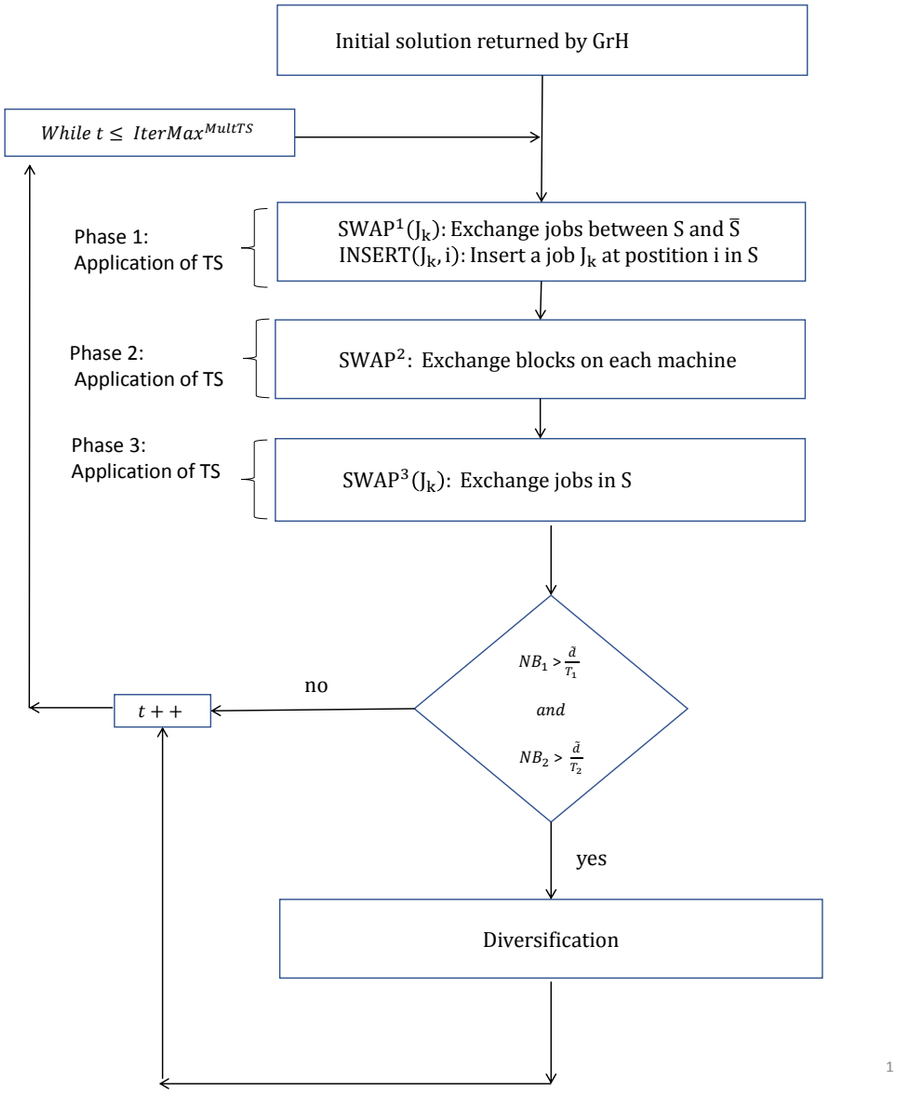

Figure 5.2: Algorithmic flow of MultTS

Neighborhood structures

MultTS exploits a variety of neighorhood structures in Phases 1, 2 and 3 to improve the current solution, as it is explained below.

Phase 1. The tabu search in Phase 1 optimizes only f_1 (rejection cost) using a neighborhood structure based on SWAP[1] and INSERT. In SWAP[1], every pair of jobs j and j' with $j \in S$ and $j' \in \overline{S}$ are considered for exchange. That is, a scheduled job is rejected and replaced by a previously rejected job. After each such potential exchange, the jobs $j \in \overline{S}$, sorted in decreasing order of rejection cost u_j, are considered one by one for insertion in the schedule, with the goal of inserting as many jobs as possible while keeping solution feasibility. Indeed, when a swap is applied between a job j in S and a job j' in $\overline{S}$, the processing time $p_{j'}$ can well be greater than p_j, which may lead to exceeding the deadline $\tilde{d}$. Conversely, when $p_{j'}$ is smaller than p_j, some free space is created in the schedule and this flexibility can then be exploited by INSERT.

Phase 2. The tabu searches in Phase 2 and 3 are aimed at improving the scheduling of accepted jobs, as identified in Phase 1, with regard to objective f_2. The neighborhood structure called SWAP[2] exchanges every pair of blocks scheduled on the same machine. After some preliminary tests, we discovered that swapping blocks between the two machines was not beneficial because the blocks are not of the same size (i.e., $T_2 < T_1$). Accordingly, the schedule of machine M_2 often exceeded the deadline after such a move, as illustrated in Figure 5.3 for the exchange of blocks B_1^2 and B_2^1.

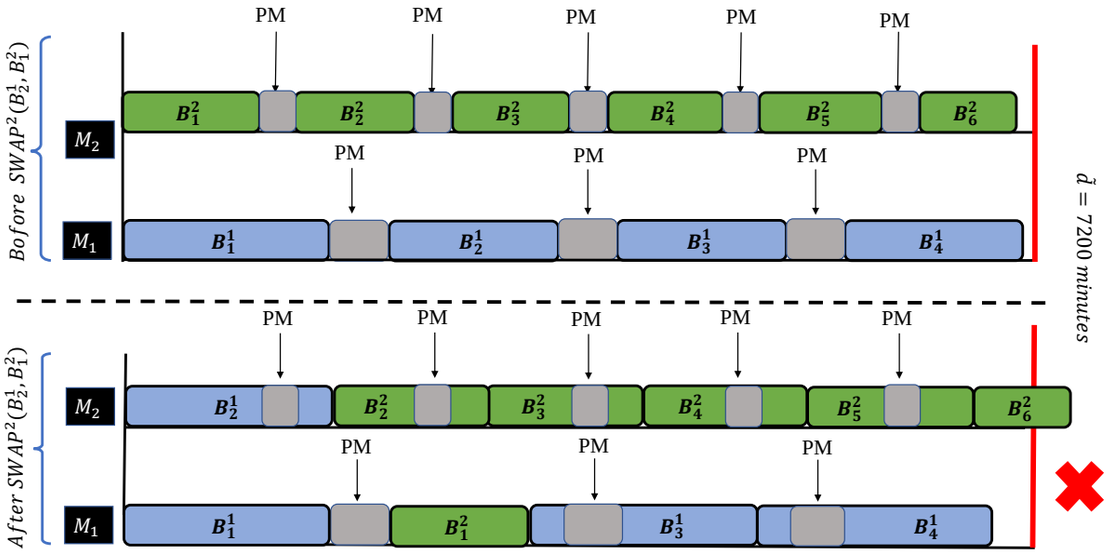

Figure 5.3: Infeasible solution after swapping two blocks between M_1 and M_2

Phase 3.

The neighborhood structure in Phase 3 is based on SWAP[3] moves where every pair of jobs, scheduled on the same machine or not, exchange their positions. The goal here is to obtain a better scheduling of the jobs within the blocks with regard to objective f_2.

It should be noted that a fraction of each neighborhood structure can be explored in Phases 1, 2 and 3 depending on the value of parameter Pr. Values 1/4, 1/2, 3/4 and 1 are considered, where $Pr = 1$ means that the whole neighborhood is explored and $Pr = 1/4, 1/2, 3/4$ mean that only the corresponding fraction of the whole neighborhood is explored. In the latter cases, the neighborhood solutions considered are randomly chosen.

Phase 4. Diversification

The purpose of Phase 4 is to provide a form of diversification in the search process. First, we consider the number of blocks NB_i scheduled on each machine M_i, where a block is the sequence of jobs between the start of the schedule and the first PM, between two consecutive PMs, or between the last PM and the end of the schedule. This number is compared with NB_i^*, which is the smallest possible number of blocks on machine M_i. We recall that preventive maintenance should ideally be done after the maximum allowed time T_i. Thus, NB_i^* is computed with the formula $\lceil \frac{\tilde{d}}{T_i + \delta_i} \rceil$. When $NB_1 > NB_1^*$ and $NB_2 > NB_2^*$, Algorithm 20 is used to rebuild a new solution. To this end, the set of jobs J is partitioned into scheduled (S) and rejected ($\overline{S}$) jobs with the aim of producing NB_i^* blocks over the scheduling horizon on each machine M_i. This is done by iteratively solving the following MILP to create each block first on machine M_2 and then on machine M_1 (note that $\overline{S}$ is equal to J at the start, and is then appropriately updated from one iteration to the next):

$$z_j = \begin{cases} 1 & \text{if job } j \text{ is scheduled} \\ 0 & \text{if job } j \text{ is rejected} \end{cases}$$

$$\text{maximize} \quad \sum_{j \in \overline{S}} z_j \tag{5.22}$$

$$\text{s.t.} \quad \sum_{j \in \overline{S}} p_j z_j \leq T_i \tag{5.23}$$

Equation (5.22) is the objective function, which maximizes the number of jobs in a block. Constraint (5.23) prevents exceeding the maximum time length T_i of a block. It should be noted that a solution is reached in less then 30 seconds on instances with 200 jobs.

Once done, the jobs in S are scheduled to minimize objective f_2. This is done by first sequencing the blocks on each machine M_i in decreasing order of their weight, which is the summation over the weights of all jobs in the block. Then, the jobs within each block are sequenced according to the WSPT rule.

Solution update

Each modification to the current solution in Phases 1, 2 and 3 needs to be correctly evaluated. It implies that jobs may have to be shifted to the right or to the left (in the latter case, to fill idle time between two consecutive

Algorithm 20: Diversification. Input: J. Output: $(S, \overline{S})$.

1. Solve the MILP to construct the largest possible blocks (ideally, perfect blocks of length T_i) on the machine with smallest T_i.
2. Solve the MILP on the other machine.
3. Put all rejected jobs in $\overline{S}$.
4. Sequence the blocks scheduled on each machine M_i in decreasing order of weight.
5. Sequence the jobs in each block according to the WSPT rule.
6. Return $(S, \overline{S})$.

jobs). However, this is done only from the point of insertion of a new job to the end of the schedule, since nothing changes before the insertion point.

Tabu status

There is no tabu list as such in our implementation, but rather a tabu tenure which is associated with each move m leading from the current solution to the next one. A tabu tenure means that it is forbidden, for a certain number of iterations, to perform an inverse move m^{-1} that would cancel out the modifications of m. For example, if a move that exchanges jobs j and l is performed at iteration ITER, then the reverse exchange will be declared tabu until iteration ITER + tabu tenure. Note also that a tabu tenure is associated with an exchange of blocks in Phase 2, rather than an exchange of jobs. The tabu tenure is a randomly chosen number in the interval $[\alpha, \beta]$. After some preliminary experiments, this interval was fixed at [5, 10] in Phases 1 and 3 and [3, 7] in Phase 2.

Aspiration criterion

The tabu status of a move is revoked it leads to a solution which is better than the best known solution. There is no risk of cycling in this case, since this new best known solution has clearly not been previously visited.

Stopping criterion

The stopping criterion for each Phase l, with $l = 1, 2, 3$, corresponds to a maximum number of iterations $Itermax_l^{MultTS}$. The value for each parameter was determined through parameter sensitivity analysis, as explained in Section 5.6.

5.5.2 Consistent tabu search

This tabu search is inspired from the work in (Zufferey and Vasquez, 2015) where a satellite range scheduling problem is addressed. As opposed to *MultTS*, infeasible neighborhood solutions are considered but are immediately repaired to restore feasibility. There are two main phases in *ConsTS*, each based on a tabu search which is aimed at optimizing one of the two objectives. Figure 5.5.2 illustrates the main steps of this algorithm. An initial solution (S, $\overline{S}$) is first generated using the greedy heuristic GrH. Then, objective function f_1 is optimized in Phase 1 through insertion moves. This phase is repeated as long as an improvement is observed within $IterMax_1^{ConsTS}$ iterations. Otherwise, objective function f_2 is optimized in Phase 2 using a swap neighborhood structure similar to the one used in Phase 3 of *MultTS*. This phase is stopped after a maximum number of iterations $IterMax_2^{ConsTS}$. As opposed to Phase 1, this phase is not repeated as long as an improvement is observed, because f_2 is only a secondary objective. The two phases are then repeated until $IterMax^{ConsTS}$ global iterations have been performed. Since the aspiration criterion is the same as in *MultTS*, we will focus on the neighborhood structures and tabu status in the following.

Neighborhood structures

Phase 1

Here, the first objective f_1 (rejection cost) is optimized using the neighborhood structure INSERT^2, where every rejected job is considered for insertion at every position in the schedule of machines M_1 and M_2. It is important to note that the insertion is enforced even if the deadline $\tilde{d}$ is exceeded. In such a case, the tentative solution is immediately repaired by removing jobs. Assuming that the (infeasible) insertion of job j takes place in block k, we first define the subset of candidate jobs for removal, L, as all jobs other than j that are scheduled between the end time of PM_{k-1} (or $t = 0$ if $k = 1$) and the end of the schedule on machine M_i, see Figure 5.5. Subset L is chosen with the aim of limiting the impact of a removal on the solution structure, while facilitating the evaluation.

Then, we follow the steps:

While $\tilde{d}$ is exceeded

1. select job s from L with smallest u_s.
2. break ties by considering job s with smallest w_s/p_s
3. break ties by randomly choosing job s
4. reject job s and update (S, $\overline{S}$).

Phase 2

Phase 2 is aimed at minimizing the second objective f_2, using the neighborhood structure $SWAP^4$. The latter is similar to SWAP^3 in *MultTS*, excepted that a swap between two jobs j and j' in positions k and k', $k < k'$, is considered only when $w_{j'}/p_{j'}$ is larger than w_j/p_j (c.f., WSPT rule).

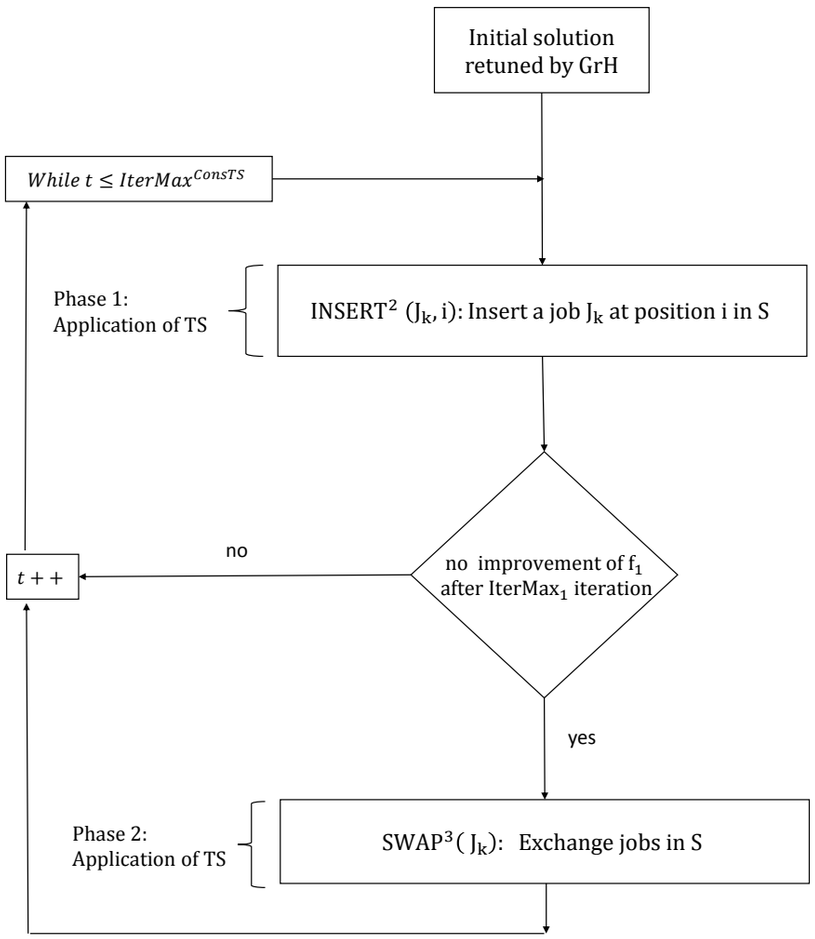

Figure 5.4: Algorithmic flow of *ConsTS*

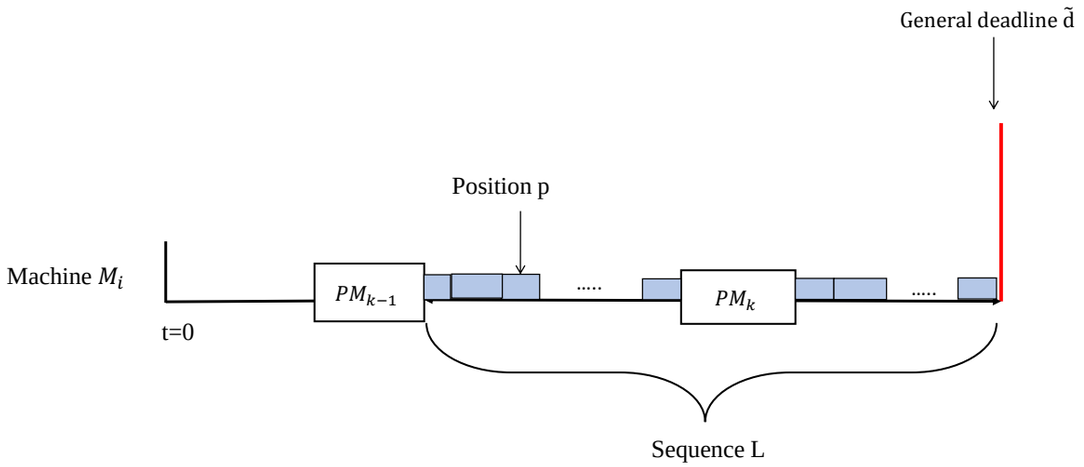

Figure 5.5: INSERT^2

Tabu status

Like *MultTS*, a tabu tenure is associated with each move. In Phase 1, only the rejected job is tabu for a certain number of iterations, while it is the reverse swap in Phase 2. In both cases, the number of iterations is randomly chosen in the interval [5, 10].

5.6 Computational results

This section reports computational results obtained with the proposed algorithms. The generation of test instances is first discussed in Sub-section 5.6.1. Then, experiments conducted to determine the best value for different parameters are described in Sub-section 5.6.2. Finally, comparisons between *MultTS*, *ConsTS* and CPLEX solver are reported and discussed in Sub-section 5.6.4.

All algorithms are coded in Java. The MIP solver is CPLEX 12.7 with Concert Technology for the Java interface. All computational tasks were performed on an i7 Intel Core at 2.50 GHz with 16 GB of RAM.

Since objective f_1 (minimizing the sum of rejection costs) is the main objective and subsumes f_2 (minimizing the weighted sum of job completion times), we will sometimes report only the values of f_1 in the following results for brevity purposes. It should also be noted that $MultTS$ and $ConsTS$ are stochastic algorithms and, consequently, they are run 10 times on each instance with only the best run being recorded (note, however, that the relative standard deviation is smaller than 0.2%).

5.6.1 Test instances

Since there are no available benchmark instances in the literature for our problem, we carried out experiments based on randomly generated data.

The job processing time p_j in minutes is uniformly distributed in the interval [30, 120] with an average of 75 minutes. Thus, it is possible to schedule at most 96 jobs per machine over a scheduling horizon of 5 days (7200 minutes), if we do not consider the maintenance activities. For example, when $n = 200$, a minimum of 8 jobs will be rejected. Accordingly, we generated instances of size, $n = 200, 210, 220$. For each size, 10 instance were generated, for a total of 30 instances. Periodic preventive maintenance is performed on the two machines after a maximum of T_i minutes of use with $T_1 = 480$ and $T_2 = 360$. The time required to perform a maintenance is set to 4 % of T_i, which translates into $\delta_1 = 20$ minutes and $\delta_2 = 15$ minutes.

The weight $w_j = b_j + h_j$ is uniformly distributed in the interval [20, 60]. That is, the priority b_j is randomly selected in the set $\{10, 20, 30\}$, while h_j is uniformly distributed over the interval [10, 30]. Finally, u_j is uniformly distributed over the interval $[b_j p_j/2,\ 2b_j p_j]$, since the rejection cost of a job depends on its priority and processing time.

5.6.2 Parameters calibration

In this Sub-section, we examine the impact of different parameters on the proposed algorithms. To this end, different values were tested for each parameter, as it is explained below.

Parameters q_1 and q_2

GrH was run with $q_1 = .05n, .1n, .2n, .3n, .4n, .5n, n$ while q_2 was set to $\frac{q_1}{2}$. Table 5.1 summarizes the average value of f_1 over the ten instances for each size n and each q_1 value. It shows that GrH produces better solutions with smaller values of q_1. For a better view of the results, the entries in Table 5.2 provide the number of instances for which a given value of q_1 produces the best solution. At the end, the value $.2n$ was selected.

	q_1=.05n	q_1=.1n	q_1=.2n	q_1=.3n	q_1=.4n	q_1=.5n	q_1=n
n=200	11270.5	**11159.7**	11462.8	11593	15200.2	16530.1	21020.6
n=210	20970.5	21127.5	**16530.1**	18013.2	19292.3	20534.8	33035
n=220	**31037.5**	31063.1	31365.5	33502	35590.3	38437.1	47697.5

Table 5.1: Average objective values of GrH based on f_1 for different values of q_1

	$q_1 = .05n$	$q_1 = .1n$	$q_1 = .2n$	$q_1 = .3n$	$q_1 = .4n$	$q_1 = .5n$	$q_1 = n$
n=200	5/10	4/10	2/10	2/10	0/10	0/10	0/10
n=210	4/10	1/10	6/10	2/10	0/10	0/10	0/10
n=220	4/10	3/10	5/10	3/10	0/10	0/10	0/10

Table 5.2: Number of best solutions found by GrH based on f_1 for different values of q_1

From now on, the results will be reported as improvements (gaps) in percent from solutions produced by GrH, that is:

$$GAP = 100 \times ((GrH - TS)/GrH) \tag{5.24}$$

where TS is either $MultTS$ or $ConsTS$ and the improvement is calculated either with regard to objective f_1 or f_2, depending on the context.

Stopping criterion

In $MultTS$, Phase l is stopped after $IterMax_l^{MultTS}$ iterations with $l \in \{1, 2, 3\}$. First, we tested $IterMax_1^{MultTS} = \frac{n}{10}, \frac{n}{5}, n, 2n, 3n, 4n, 5n$. The average values of objective f_1 for these values are summarized in Table 5.3 with the corresponding computation times in seconds. Since Phases 2 and 3 are aimed at optimizing f_2, Table 5.4 shows the average values of f_2 at the end of Phase 3, after fixing $IterMax_1^{MultTS}$ to $2n$, while entries of columns 5, 6 and 7 report the average computation time in seconds.

We observe that the value of f_1 is rather stable after $IterMax_1^{MultTS}$= $2n$ iterations. It should also be noted that the computation time is around one minute for the largest number of iterations $5n$. With regard to $IterMax_3^{MultTS}$, the entries in the second column of Table 5.4 for instances of size $n = 200$ shows that improvements to f_2 are obtained with increasing values of $IterMax_3^{MultTS}$, although the gap between $IterMax_3^{MultTS} = 3n, 4n$ and $5n$ is quite small. On the other hand, entries of column 3 and 4 show that f_2 remains practically constant for $IterMax_3^{MultTS} = 3n, 4n$ and $5n$. In all cases, the computation time is less than one minute, even for $5n$ iterations. Based on these results, $IterMax_3^{MultTS}$ was set to $3n$. Note also that $IterMax_2^{MultTS}$ was set to $n/5$ given the relatively small-sized of the corresponding neighborhood, where blocks are moved rather than individual jobs.

	f_1 value			Computation time		
	$n = 200$	$n = 210$	$n = 220$	$n = 200$	$n = 210$	$n = 220$
n/10	25.66	19.62	18.41	0.81	1.26	3.79
n/5	25.78	19.66	18.48	0.67	2.10	22.47
n	25.71	19.77	18.50	2.96	13.50	28.27
$2n$	25.71	19.77	18.62	9.77	23.65	41.59
$3n$	25.71	19.76	18.62	13.29	35.78	41.90
$4n$	25.71	19.77	18.57	37.74	36.96	49.46
$5n$	25.71	19.77	18.57	65.84	51.02	64.66

Table 5.3: Average value of f_1 with regard to $IterMax_1$ and instance size n

Since the two neighborhoods explored in ConsTS are similar to the ones in Phases 1 and 3 of $MultTS$, $IterMax_1^{ConsTS}$ and $IterMax_2^{ConsTS}$ were set to $2n$ and $3n$, respectively.

The global iteration counter of $MultTS$ and $ConsTS$ was lunched to $IterMax^{MultTS}$ such that the computational time is set to 60 minutes.

	f_2 value			Computation time		
	$n = 200$	$n = 210$	$n = 220$	$n = 200$	$n = 210$	$n = 220$
$n/10$	38.57	37.68	40.53	9.54	12.3	10.9
$n/5$	38.63	37.75	40.57	14.96	12.64	9.75
n	38.65	37.84	40.57	18.36	21.22	17.38
$2n$	38.70	37.87	40.64	30.21	32.46	22.91
$3n$	38.71	37.90	40.67	28.05	30.28	38.45
$4n$	38.73	37.90	40.67	45.17	26.67	28.18
$5n$	38.75	37.90	40.67	58.16	45.53	47.88

Table 5.4: Average value of f_2 with regard to $IterMax_3$ and instance size n

Neighborhood exploration parameter Pr^N

We ran Phases 1 and 3 of $MultTS$ with $Pr^N = 1$, 3/4, 1/2, 1/4, where Pr^N is the fraction of the neighborhood explored. For these experiments, the number of iterations of $MultTS$ in Phases 1, 2 and 3 was set to $2n$, $n/5$ and $3n$, respectively. Tables 5.5 and 5.6 summarize the results, while considering that Phase 1 optimizes f_1 and Phase 3 optimizes f_2. From these results, we can easily see that high values of Pr^N are associated with better objective values, although at a computational cost. For $n = 220$ in Table 5.5, for example, the average computation time with $Pr^N = 1$ is 27.60 seconds, while it is only 7.18 seconds with $Pr^N = 1/4$. On the other hand, the average solution value obtained with $Pr^N = 1$ is substantially better than $Pr^N = 1/4$.

		$Pr^N = 1$	$Pr^N = 3/4$	$Pr^N = 1/2$	$Pr^N = 1/4$
n=200	f_1	25.68	25.50	25.29	24.77
	comp. time	9.77	3.87	1.46	0.97
n=210	f_1	19.71	19.54	19.55	18.98
	comp. time	23.66	10.80	6.68	2.70
n=220	f_1	18.56	18.23	18.21	18.12
	comp. time	27.60	16.37	8.30	7.18

Table 5.5: Comparison of f_1 values for different problem sizes n and Pr^N values

Based on these results, and given the benefits obtained from a complete exploration of each neighborhood, Pr^N was set to 1 in Phases 1 and 3. Furthermore, the whole neighborhood is also explored in Phase 2 of $MultTS$, due to its smaller size. Based on the results obtained with $MultTS$, Pr^N was also set to 1 in Phases 1 and 2 of $ConsTS$.

5.6.3 Results on large test instances

This Sub-section analyzes the results obtained with $MultTS$ and $ConsTS$.

		$Pr^N = 1$	$Pr^N = 3/4$	$Pr^N = 1/2$	$Pr^N = 1/4$
n=200	f_2	38.73	38.79	38.65	38.64
	comp. time	28.05	28.97	22.71	11.74
n=210	f_2	37,87	38,08	38,01	38,00
	comp. time	30.28	31.16	19.95	11.17
n=220	f_2	40.64	40.73	40.62	40.56
	comp. time	38.46	30.76	17.45	13.18

Table 5.6: Comparison of f_2 values for different problem sizes n and Pr^N values

First, we analyze the impact of using the diversification step in *MultTS*. Second, we compare the two implementations of *MultTS* with a tabu search versus a descent in Phase 2. Finally, we report and discuss the final solutions produced by *MultTS* and *ConsTS*.

Impact of diversification (Phase 4 of MultTS)

Table 5.7 reports the improvement in objectives f_1 and f_2 obtained from the different phases of *MultTS* for each instance size n. As we can see, there is more improvement in objective f_1 when *MultTS* starts from the initial solution constructed by GrH in Phase 1. Since Phases 2 and 3 are not concerned with f_1, the results of Phase 4 (diversification) are then reported. The last line shows the best solution obtained after one hour of computational time.

	f_1			f_2		
MultTS components	$n = 200$	$n = 210$	$n = 220$	$n = 200$	$n = 210$	$n = 220$
Phase 1	25.25	17.04	18.21	-	-	-
Phase 2	-	-	-	15.52	13.43	17.97
Phase 3	-	-	-	38.52	37.50	40.65
Phase 4	-174.42	-150.54	-120.63	29.05	27.48	30.62

Table 5.7: Impact of different phases of *MultTS*

Results comparison between TS and DA in Phase 2 of *MultTS*

Table 5.8 compares the final results obtained when using the DA algorithm once and TS algorithm another time in Phase 2 of *MultTS* for instance size $n = 200$, 210 and 220. We remind that the neighborhood structure used in Phase 2 consists on swapping blocks. From the computed GAPs reported in the Table 5.8, we conclude that the results obtained remain the same when DA or TS are used. GAP_{DA} and GAP_{TS} used here for the comparison are computed on the basis of the Equation 5.24. Thus, v_1 refers to f_2 returned by GrH in both in GAP_{DA} and GAP_{TS}, when v_2 refers, in the case of GAP_{TS}, to the best value of the target criterion f_2 when

TS is applied, and to the best value of the objective function f_2 returned when this time it is DA which is implemented in Phase 2 of *MultTS*.

	$n = 200$	$n = 210$	$n = 220$
GAP_{DA}	38.438	38.756	37.655
GAP_{TS}	38.439	38.655	37.659

Table 5.8: Results comparison between the application of TS or DA in phase 2 of *MultTS*

5.6.4 Comparison between MultTS and ConsTS

Table 5.9 presents a comparison between objectives f_1 and f_2 of *MultTS* and *ConsTS* for instances with $n =$ 200, 210 and 220 jobs. The GAP metric is used to calculate the improvement achieved on the objective functions f_1 and f_2. We remind that GAP is computed as explained in Equation 5.24. Table| 5.9 mentioned above is composed of two parts. Hence, columns two, three and four are devoted to the first criterion f_1 while the remaining columns five, six and seven to f_2. The third line shows the GAP between v_1 which represents here f_1 respectively f_2 values returned by GrH and v_2 which is the better objective function value f_1 respectively f_2 found by *MultTS*, while the last line, v_2 represent f_1 or f_2 value returned, this time, by *ConsTS*. We can easily notice that *MultTS* produces better solutions with regard to objective f_1, while *ConsTS* does a better job at optimizing objective f_2. Both *MultTS* and *ConsTS* obviously outperforms GrH which is used to provide initial solutions for the TS. Table 5.11 show the number of times each method found the best solution on the 10 test instances with the same size, namely n=200, 210 and 220. Clearly, *MultTS* improves over *ConsTS* on the first objective f_1. For example, with n=210, *MultTS* finds the best solution on 7 instances out of 10. On the other hand, *ConsTS* is always better with regard to f_2. As columns 5, 7 and 9 of Table5.11 shows so well, for each instance size n, *ConsTS* records 10 best solutions out of 10 and *multTS* does not find any. Table 5.10

	f_1			f_2		
	$n = 200$	$n = 210$	$n = 220$	$n = 200$	$n = 210$	$n = 220$
GAP_{MultTS}	25.7	20.2	18.9	38.4	37.3	36.3
GAP_{ConTS}	22.9	16.5	11.0	40.9	38.7	37.7

Table 5.9: Comparison between MultTS and ConsTS

summarizes the average number of rejected jobs recorded for each instance size, namely, n =200, 210 and 220 and for each method *MultTS* and *ConsTS*. Entries of the second line named by $\overline{S}_{GrH}$ display the average number of rejected jobs returned by the greedy heuristic GrH that is used to compare with, while $\overline{S}_{MulTS}$ and $\overline{S}_{ConsTS}$ refer respectively to the average number of rejected jobs when applying *MultTS*, respectively *ConsTS*. We note that, when using *MultTS*, the rejected jobs are reduced in average about 4.4 jobs when n=200, 3.3 when n=210 and about 10.2 when n=220. On the other hand, the average number of rejected jobs is less important when

$ConsTS$ is applied. Indeed, this number is equal to 2.3 when n=200, 1.9 when n=210 and 4.9 when n =220. Consequently, even if minimizing f_1 does not quite correspond to minimize the number of rejected jobs but to minimize the cost of rejecting jobs, we can state that these results confirm the fact that $MulTS$ is much better than $ConsTS$ with regard to f_1.

	n=200	n=210	n=220
GrH	24.9	39.5	51.5
$MultTS$	20.5	36.2	41.3
$ConsTS$	22.6	37.6	46.6

Table 5.10: Average number of rejected jobs

		MulTS	ConsTS
n=200	f_1	10/10	1/10
	f_2	1/10	9/10
n=210	f_1	8/10	2/10
	f_2	0/10	10/10
n=220	f_1	9/10	1/10
	f_2	0/10	10/10

Table 5.11: Performance of MulTS and ConsTS with regard to the number of best solutions found for n=200, 210 and 220 jobs

Tables 5.12 and 5.13 summarize the average run times that is consumed by each phase that constitutes *MultTS* and *ConsTS* when the stopping criteria $IterMax^{MultTS}$ and $IterMax^{ConsTS}$ are set to one hour and for each instance size n =200, 210, and 220.

We first notice that the more the size of the instances increases the more the run times of each phase also increases in both *MultTS* and *ConsTS*.

From Table 5.12 we note that during one hour, Phase 3 of *MultTS* is the phase that takes the most run times (almost 57 minutes) and phase 2 the one that takes the least run times. This remains obvious because the neighborhood structure used in Phase 3 is based on the interchange (swap) of all jobs belonging to the set S. The evaluation function being generally the one that consumes the most time has certainly been much more solicitous during this phase. Contrary to this, in phase 2, where we proceed to only interchange blocks of jobs instead of jobs and therefore moving inside the solution space does not take much time.

From Table 5.13, we note that phase 1 takes longer than phase 2 in *ConsTS*. This can be explained by the fact that in Phase 2 the number of calls to the evaluation function is reduced. Indeed, jobs are interchanged only if the *WSPT* rule is respected (see Section 5.5.2) whereas in Phase 1, the neighborhood structure consists to insert each job of $\overline{S}$ in all the possible positions in S.

	n=200	n=210	n=220
Phase 1	116.37	112.48	118.55
Phase 2	7.032	6.97	7.27
Phase 3	3445.11	3433.04	3395.6
Phase 4	26.33	77.41	85.22

Table 5.12: Average time duration (seconds) for each phase of *MultTS*

	n=200	n=210	n=220
Phase 1	2586.09	2400.14	2013.84
Phase 2	1031.75	1203.11	1588.30

Table 5.13: Average time duration (seconds) for each phase of *ConsTS*

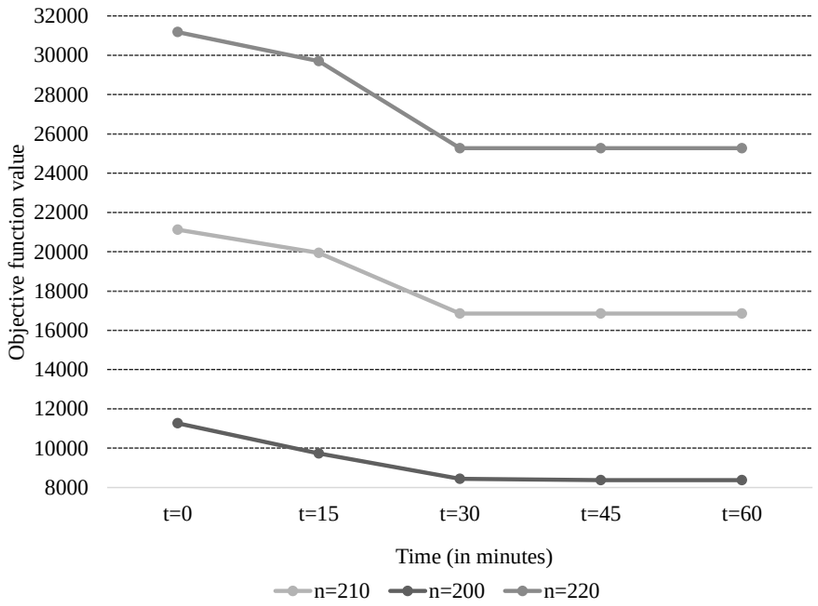

Figure 5.6: f_1 values variation regarding to $IterMax^{MultTS}$ and instance size set to n jobs

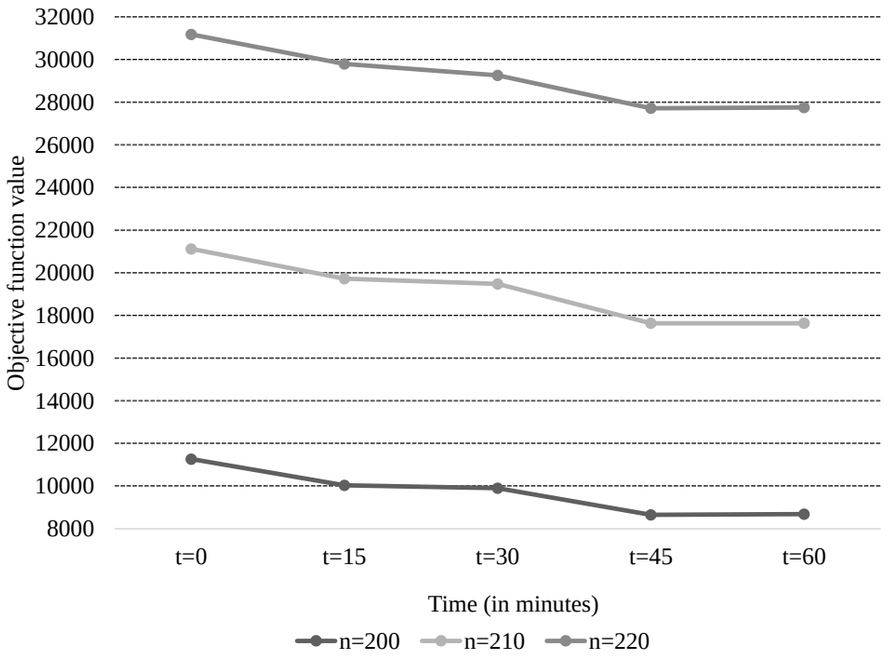

Figure 5.7: f_1 values variation regarding to $IterMax^{ConsTS}$ and instance size set to n jobs

Figures 5.6 and 5.7 show respectively the evolution of the best objective function value f_1 with respect to the execution times parameterized by the stopping criterion $IterMax^{MulTS}$ respectively $IterMax^{ConsTS}$. When $t = 0$, it is the value returned by the greedy GrH which is recorded. We conclude that, best value found considering f_1 is slightly increasing with the progress of t and became constant when $t = 30$ minutes for MultTS and when $t = 45$ minutes for ConsTS. From the results above we can conclude that it is better to set $IterMax^{MulTS}$ to 30 minutes and $IterMax^{ConsTS}$ to 45 minutes instead of one hour.

5.7 Conclusion

In this work, we considered a parallel machine scheduling problem with two non identical machines over a weekly planning horizon, while considering periodic preventive maintenance. Two objectives were considered: jobs rejection cost and weighted sum of job completion times. Lexicographic Optimization was used to address this multiobjective problem. We first introduced a MILP formulation for the problem. Then, we developed a greedy heuristic and two tabu search-based metaheuristics, denoted GrH, $MultTS$ and $ConsTS$, respectively. Computational experiments were conducted on randomly generated data. They showed that $MultTS$ outperform $ConsTS$ when jobs rejection cost is minimized, which is the main objective, while $ConsTS$ performed better than $MulTS$ when the weighted sum of job completion times is minimized, which is the secondary objective.

Chapter 6

Conclusion and prospects

This chapter summarizes the thesis by presenting the research contributions and proposing some future works and extensions of this research.

As a guideline, this thesis addresses scheduling problems with unavailability constraints due to preventive maintenance on the basis of academic problems resulting from simplified industrial problems. This leads to some variations around the common theme of maintenance-production scheduling problems.

Firstly, the type and number of machines differ from one problem to another. To this aim, we studied two cases: single-machine scheduling and multiple-machines scheduling. Secondly, in addition to the number of machines in the workshop, we focused on the production environment. Indeed, two other cases are taken into account: scheduling considering preventive maintenance and sequence-dependent setup times impacting each consecutive pair of tasks that should be performed on a same machine. Finally, amongst the bunch of existing performance criteria to measure the quality of a schedule, we studied two criteria: the weighted sum of completion times and the jobs rejection cost.

On the basis of the four abovementioned cases, we studied and solved three scheduling problems detailed in chapters 3, 4 and 5. To this aim, this manuscript gathers our investigations on several heuristics and meta-heuristics, as well as mathematical models based on integer linear programming which however systematically failed to solve the large-sized instances. As three pillars of our research, the three scheduling problems lead us to propose several contributions and future works which are summarized below.

In chapter 3, we considered the case of a multitask machine where only minor negligible breakdowns appear and where the cost of repairing big failures is very high. The machine must undergo, with much lower cost, preventive maintenance (PM) to minimize the risk of failures. PM is carried out to meet, for example, a regulatory requirement and is applied at regular intervals. Without this maintenance, repair costs can be very high. In this regard, the goal of this chapter is to minimize the weighted sum of completion times on a single machine subject to periodic preventive maintenance. First, we provided a Mixed Integer Linear Programming formulation to solve this problem optimally. Then we proved the NP-hardness of the studied problem. We have also provided some properties that have been used to build several efficient heuristics. Finally, several lower bounds were also proposed and special cases of the problem were discussed. All the abovementionned methods were tested on large randomly generated data. The lower bounds helped us to discuss the performance of the heuristics when the size of instances is greater than 20. The result showed that the MILP formulation is efficient

on instances with size ≤ 20 and the proposed heuristics are efficient algorithms to solve large-sized instances.

In a second part of chapter 3, we proposed a GVNS algorithm to solve this NP-hard problem. The presented algorithm was tested and compared with a MILP model for small instance sizes, and with four lower bounds for instance sizes greater than 20, three of them based on job splitting and one based on the relaxation of one integrity constraint. The computational results showed that the proposed GVNS is very efficient both in terms of quality of the objective function values and computational time.

In future work, it would be interesting to propose more lower bounds and generate more test instances. It would also be interesting to compare the performance of GVNS with other metaheuristics for solving the considered problem after generalizing it by adding more realistic constraints such as release times, failures, etc. Another extension of this research would be to consider the interval between two consecutive maintenance activities as a decision variable of the problem.

In chapter 4, we presented the results obtained when addressing a scheduling problem that combined periodic maintenance and sequence-dependent setup times. The complexity of the problem forced to design heuristic algorithms to solve it. In this regard, we compared two alternative integer programming formulations of the studied problem, considering both setup times and periodic maintenance with the aim of minimizing the weighted sum of completion times. The first formulation is inspired from the bin packing problem formulation as the periodic maintenance defines batches. In the second formulation, the maintenance activity is considered as a dummy job. As the first MILP formulation uses a significantly smaller number of constraints, it slightly outperforms the second MILP formulation for the given test sets. But in the two cases, only the smallest instances with 8, 10 jobs and some instances with 15 jobs were solved within the time limit of 30 minutes.

Due to the complexity of the problem, we proposed two heuristics named MS-DSH and MS-GVNS, composed of a greedy phase and an improvement phase. In this regard, MS-GVNS combines a greedy heuristic and a local search heuristic while MS-GVNS uses a GVNS metaheuristic instead of a local search. The computational results showed that the performance of the heuristics was satisfactory obtaining reasonably good solutions in very short times. Unsurprisingly, MS-GVNS gives better solutions than MS-DSH. Therefore, this heuristic can be applied to large-sized instances of similar problems, where the standard commercial solvers fail to find an optimal solution within reasonable times.

As an extension of this work, we believe that the study of other variants of the problem with several machines and/or several criteria seems to us closer to the industrial reality. A last important point to explore also lies in taking into account random breakdowns that may occur over time.

In Chapter 5, we considered a two non identical parallel machine scheduling problem with a weekly planning horizon and a flexible preventive maintenance. Two criteria are studied: The jobs rejection cost and the weighted sum of completion times. Lexicographic Optimization (LO) technique is used to solve this problem. We first introduced an integer linear formulation for the problem, but this latter is not efficient to solve the large-sized instances.

Consequently, a greedy algorithm and two methods based on a Tabu Search heuristic denoted respectively by GrA, $MultTS$ and $ConsTS$ are proposed to solve the problem. Computational experiments are conduced

on the basis of randomly generated data and showed that *MultTS* and *ConsTS* efficiently outperform GrA, *MultTS* outperform *ConsTS* when minimizing the jobs rejection cost while *ConsTS* is better than $MulTS$ when minimizing the weighted sum of completion times.

Future works will investigate different extensions of the considered problems (e.g., several machines, other objectives, failures, ...).

As common prospects, in case of considering stochastic aspects, an integration pattern of optimization and simulation should be used to solve the studied problems. It would be also interesting to study the online version of the studied problems since in practice, decisions should be made without knowing the complete instance (i.e. jobs become known only at their release date).

Publications resulting from This Research

Journal Paper

- Krim H., Benmansour R., Duvivier D., Artiba A. (2018). A variable neighborhood search algorithm for solving the single machine scheduling problem with periodic maintenance. RAIRO-Operations Research journal

Submitted journal paper

- Krim H., Benmansour R., Duvivier D., Ait-Kadi D., Hanafi S., (2018).Minimizing the weighted sum completion times on a single machine with periodic maintenance. Computational Optimization and Applications.

Working papers

- Krim H., Zufferey N.,Potvin J-Y., Benmansour R., Duvivier D. (2019). Two parallel machine scheduling problem with periodic maintenance, jobs rejection and weighted sum completion times.
- Krim H., Benmansour R., Duvivier D. (2019). Variable neighborhood search for a single machine scheduling problem with sequence-dependent setup times and periodic maintenance.

International conference with reviewing committees and published proceedings

- Krim H., Benmansour R., Duvivier D., Artiba A. (2019). Heuristic for single machine scheduling problem with sequence dependent setup-times and periodic maintenance. International Conference on Industrial Engineering and Systems Management (IEEE IESM 2019), Shanghai, China, pp.(accepted), september.
- Krim H., Benmansour R., Duvivier D. (2018). On the single machine scheduling problem with sequence-dependent setup times and periodic maintenance. Proceedings of the fifth edition of the IEEE International Conference on Control, Decision and Information Technologies (CODIT'18), Thessaloniki, Greece, april.

National conferences with reviewing committees and published proceedings

- Krim H., Benmansour R., Duvivier D. (2017). Scheduling with sequence-dependent setup times and periodic maintenance on a single machine to minimize the total weighted completion time. Proceedings de la conférence ROADEF 2017, Metz, february.
- Krim H., Benmansour R., Duvivier D. (2016). Minimizing the weighted completion time on a single machine with periodic maintenance. ROADEF 2016, ConfÃl'rence ROADEF 2016, Compiégne, february.

Posters and other communications

- Krim H., Benmansour R., Duvivier D. (2017). Résumé des travaux de recherche menés dans le cadre d'ELSAT 2020. Groupe de travail du projet OLOGMAESTRO (ELSAT/OS2/P2), LGI2A, Béthune, France, october.

- Krim H., Benmansour R., Duvivier D. (2016).Optimisation et simulation pour l'ordonnancement et la maintenance dans un environnement incertain. Poster présenté à la Matinée des Chercheurs 2016 (MDC'16), UVHC, LAMIH, Valenciennes, may .

Bibliography

Adams, T. M. (2008). *Traffic Operations Asset Management Systems (TOAMS): Study findings*. Midwest Regional University Transportation Center, University of Wisconsin, Madison.

Aggelogiannaki, E. and Sarimveis, H. (2006). Multiobjective constrained mpc with simultaneous closed-loop identification. *International Journal of Adaptive Control and Signal Processing*, 20(4):145–173.

Agnetis, A. and Mosheiov, G. (2017). Scheduling with job-rejection and position-dependent processing times on proportionate flowshops. *Optimization Letters*, 11(4):885–892.

Allahverdi, A. (2015). The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, 246(2):345–378.

Allahverdi, A., Gupta, J. N. D., and Aldowaisan, T. (1999). A review of scheduling research involving setup considerations. In *Omega – The International Journal of Management Science*, pages 219–239. Elsevier Science Ltd.

Ángel-Bello, F., Álvarez, A., Pacheco, J., and Martínez, I. (2011). A single machine scheduling problem with availability constraints and sequence-dependent setup costs. *Applied Mathematical Modelling*, 35(4):2041–2050.

Artiba, A., Dhaevers, V., Duvivier, D., and Elmaghraby, S. E. (2011). A multi-model approach for production planning and scheduling in an industrial environment, chapter 19. In Kempf, K. G., Keskinocak, P., and Uzsoy, R., editors, *Planning Production and Inventories in the Extended Enterprise*, volume 152 of *International Series in Operations Research & Management Science*, pages 489–530. Springer New York. ISBN 9781441981905, Kluwer International Series in Operations Research & Management Science, DOI:10.1007/978-1-4419-8191-2_19.

Arts, R., Knapp, G. M., and Mann Jr, L. (1998). Some aspects of measuring maintenance performance in the process industry. *Journal of Quality in Maintenance Engineering*, 4(1):6–11.

Bahriye, C., Ceyda, O., and Sibel, S. F. (2012). A tabu search algorithm for order acceptance and scheduling. *Computers & Operations Research*, 39(6):1197–1205.

Banks, J. (1998). *Handbook of simulation: principles, methodology, advances, applications, and practice*. John Wiley & Sons.

Baptiste, P., Giard, V., Haït, A., and Soumis, F. (2005). *Gestion de production et ressources humaines. Méthodes de planification dans les systèmes productifs*. Presses internationales Polytechnique, Montréal. ISBN 9782553011450.

Barlow, R. E. and Proshan, F. (1976). *Theory Of Modeling and Simulation*. Wiley Interscience.

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W., and Vance, P. H. (1998). Branch-and-price: Column generation for solving huge integer programs. *Operations research*, 46(3):316–329.

Baykasoğlu, A. and Ozsoydan, F. B. (2018). Dynamic scheduling of parallel heat treatment furnaces: A case study at a manufacturing system. *Journal of manufacturing systems*, 46:152–162.

Bellman, R. (1966). Dynamic programming. *Science*, 153(3731):34–37.

Bellman, R. et al. (1954). The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6):503–515.

Belouadah, H., Posner, M. E., and Potts, C. N. (1992). Scheduling with release dates on a single machine to minimize total weighted completion time. *Discrete applied mathematics*, 36(3):213–231.

Ben Daya, M., Duffuaa, S. O., Raouf, A., Knezevic, J., and Ait Kadi, D., editors (2009). *Handbook of Maintenance Management and Engineering*. Springer, London. ISBN 9781848824713, doi:10.1007/978-1-84882-472-0.

Benmansour, R., Allaoui, H., Artiba, A., and Hanafi, S. (2014). Minimizing the weighted sum of maximum earliness and maximum tardiness costs on a single machine with periodic preventive maintenance. *Computers & Operations Research*, 47:106–113.

Benmansour, R., Allaoui, H., Artiba, A., Iassinovskin, S., and Pellerin, R. (2011). Simulation-based approach to joint production and preventive maintenance scheduling on a failure-prone machine. *Journal of Quality in Maintenance Engineering*, 17(3):254–267.

Benmansour, R., Artiba, A., Duvivier, D., Ramat, E., and Ait-Kadi, D. (2012). Scheduling of production and maintenance activities under reliability constraint. In *9th International Conference on Modeling, Optimization & SIMulation*, Bordeaux - France.

Billaut, J. and T´Kindt, V. (2002). *Multicriteria scheduling – Theory, Models and Algorithms*. Springer. ISBN 9783540282303.

Billaut, J.-C., Moukrim, A., and Sanlaville, E., editors (2005). *Flexibilité et robustesse en ordonnancement*. Hermes Science, Lavoisier, Paris, France. ISBN 2746210282.

Boschian, V., Rezg, N., and Chelbi, A. (2009). Contribution of simulation to the optimization of maintenance strategies for a randomly failing production system. *European Journal of Operational Research*, 197:1142–1149.

Bovet, D. P., Crescenzi, P., and Bovet, D. (1994). *Introduction to the Theory of Complexity*. Citeseer.

Breit, J. (2007). Improved approximation for non-preemptive single machine flow-time scheduling with an availability constraint. *European Journal of Operational Research*, 183(2):516–524.

Brusco, M. J. and Stahl, S. (2006). *Branch-and-bound applications in combinatorial data analysis*. Springer Science & Business Media.

Cao, Z., Wang, Z., Zhang, Y., and Liu, S. (2006). On several scheduling problems with rejection or discretely compressible processing times. *Theory and Applications of Models of Computation*, pages 90–98.

Carlier, J. and Chrétienne, P. (1988). *Problèmes d'ordonnancement, modélisation, complexité, algorithmes*. Masson. ISBN 2225812756.

Ceyda, O., Sibel, S. F., and Bilgintürk, Y. Z. (2010). Order acceptance and scheduling decisions in make-to-order systems. *International Journal of Production Economics*, 125(1):200–211.

Chekuri, C., Johnson, R., Motwani, R., Natarajan, B., Ran, B., and Schlansker, M. (1996). Profile-driven instruction level parallel scheduling with application to super blocks. In *Microarchitecture, 1996. MICRO-29. Proceedings of the 29th Annual IEEE/ACM International Symposium on*, pages 58–67. IEEE.

Chen, W. (2009). Scheduling with dependent setups and maintenance in a textile company. *Computers & Industrial Engineering*, 57(3):867–873.

Chen, W.-J. (2006). Minimizing total flow time in the single-machine scheduling problem with periodic maintenance. *Journal of the Operational Research Society*, 57(4):410–415.

Chen, W.-J. (2007). An efficient algorithm for scheduling jobs on a machine with periodic maintenance. *The International Journal of Advanced Manufacturing Technology*, 34(11-12):1173–1182.

Chen, Y., Zhang, A., and Tan, Z. (2011). Single machine scheduling with semi-resumable machine availability constraints. *Applied Mathematics-A Journal of Chinese Universities*, 26(2):177–186.

Chen, Z.-L. and Powell, W. B. (1999). Solving parallel machine scheduling problems by column generation. *INFORMS Journal on Computing*, 11(1):78–94.

Cheng, T. and Sin, C. (1990). A state of the art review of parallel machine scheduling research. *European Journal of Operational Research*, 47(3):271–292.

Chrétienne, P., Coffman Jr., E., Lenstra, J., and Liu, Z., editors (1995). *Scheduling theory and its applications*. John Wiley & Sons, Chichester.

Collette, Y. and Siarry, P. (2002). *Optimisation multiobjectif*. Algorithmes: Eyrolles. Eyrolles. ISBN 9782212111682.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to algorithms*. MIT press.

Desaulniers, G., Desrosiers, J., and Solomon, M. M. (2005). *Column generation*. Springer science and Business Media, Inc. ISBN-10 978-0-387-25485-2.

Desrosiers, J. and Lübbecke, M. E. (2005). A primer in column generation. In *Column generation*, pages 1–32. Springer.

Dorigo, M. and Di Caro, G. (1999). Ant colony optimization: a new meta-heuristic. In *Proceedings of the 1999 congress on evolutionary computation*, volume 2, pages 1470–1477.

Duvivier, D. (2015). *Contributions au couplage de l'optimisation et de la simulation*. Thèse d'habilitation à diriger des recherches, Université de Valenciennes et du Hainaut-Cambrésis, LAMIH UMR CNRS 8201, Valenciennes, France.

Ebrahimy Zade, A. and Fakhrzad, M. B. (2013). A dynamic genetic algorithm for solving a single machine scheduling problem with periodic maintenance. *ISRN Industrial Engineering*.

Ehrgott, M. (2005). *Multicriteria optimization*, volume 491. Springer Science & Business Media.

Emami, S., Sabbagh, M., and Moslehi, G. (2016). A lagrangian relaxation algorithm for order acceptance and scheduling problem: a globalised robust optimisation approach. *International Journal of Computer Integrated Manufacturing*, 29(5):535–560.

Engels, D. W., Karger, D. R., Kolliopoulos, S. G., Sengupta, S., Uma, R., and Wein, J. (2003). Techniques for scheduling with rejection. *Journal of Algorithms*, 49(1):175–191.

Esquirol, P. and Lopez, P. (1999). *L'ordonnancement*. Economica. ISBN 2717837981.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability; A Guide to the theory of NP-Completeness*. Freeman and Company. ISBN 0716710455.

Gendreau, M., Guertin, F., Potvin, J.-Y., and Taillard, E. (1999). Parallel tabu search for real-time vehicle routing and dispatching. *Transportation science*, 33(4):381–390.

Gendreau, M., Hertz, A., and Laporte, G. (1994). A tabu search heuristic for the vehicle routing problem. *Management science*, 40(10):1276–1290.

Gendreau, M. and Potvin, J.-Y. (2010). *Handbook of metaheuristics*, volume 2. Springer.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5):533–549.

Glover, F. (1989). Tabu search?part i. *ORSA Journal on computing*, 1(3):190–206.

Glover, F. W. and Kochenberger, G. A. (2006). *Handbook of metaheuristics*, volume 57. Springer Science & Business Media.

GOThA (1993). Les problèmes d'ordonnancement. *RAIRO Recherche Opérationnelle/Operations Research*, 27(1):77–150. http://homepages.laas.fr/lopez/publis/PAPERS/gotha93.pdf.

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

Gravel, M., Price, W. L., and Gagné, C. (2000). Scheduling jobs in an alcan aluminium foundry using a genetic algorithm. *International Journal of Production Research*, 38(13):3031–3041.

Guo, P., Chen, W., and Wang, Y. (2013). A general variable neighborhood search for single-machine total tardiness scheduling problem with step-deteriorating jobs. *arXiv preprint arXiv:1301.7134*.

Guo, P., Weidinger, F., and Boysen, N. (2018). Parallel machine scheduling with job synchronization to enable efficient material flows in hub terminals. *Omega*.

Gupta, J. and Ho, J. (1999). A new heuristic algorithm for the one-dimensional bin-packing problem. *Production Planning & Control*, 10(3):598–603.

Hansen, P., Jaumard, B., Mladenovic, N., and Parreira, A. (2000). Variable neighborhood search for weighted maximum satisfiability problem. *Les Cahiers du GERAD*, 62.

Hansen, P., Mladenović, N., Brimberg, J., and Pérez, J. A. M. (2010a). Variable neighborhood search. In *Handbook of metaheuristics*, pages 61–86. Springer.

Hansen, P., Mladenović, N., and Pérez, J. A. M. (2010b). Variable neighbourhood search: methods and applications. *Annals of Operations Research*, 175(1):367–407.

Hansen, P., Mladenović, N., and Perez-Britos, D. (2001). Variable neighborhood decomposition search. *Journal of Heuristics*, 7(4):335–350.

He, J., Li, Q., and Xu, D. (2016). Scheduling two parallel machines with machine-dependent availabilities. *Computers & Operations Research*, 72:31–42.

Helsgaun, K. (2006). *An effective implementation of K-opt moves for the Lin-Kernighan TSP heuristic*. PhD thesis, Roskilde University. Department of Computer Science.

Holland, J. H. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press.

Hsu, C.-J., Low, C., and Su, C.-T. (2010). A single-machine scheduling problem with maintenance activities to minimize makespan. *Applied Mathematics and Computation*, 215(11):3929–3935.

Ilić, A., Urošević, D., Brimberg, J., and Mladenović, N. (2010). A general variable neighborhood search for solving the uncapacitated single allocation p-hub median problem. *European Journal of Operational Research*, 206(2):289–300.

James, T., Rego, C., and Glover, F. (2009). Multistart tabu search and diversification strategies for the quadratic assignment problem. *IEEE TRANSACTIONS ON SYSTEMS, Man, And Cybernetics-part a: systems and humans*, 39(3):579–596.

Jarboui, B., Siarry, P., and Teghem, J. (2013). *Métaheuristiques pour l'ordonnancement monocritère des ateliers de production*. Lavoisier. ISBN 9782746289260.

Ji, M., He, Y., and Cheng, T. E. (2007). Single machine scheduling with periodic maintenance to minimize makespan. *Computers & operations research*, 34(6):1764–1770.

Jiang, D., Tan, J., and Li, B. (2017). Order acceptance and scheduling with batch delivery. *Computers & Industrial Engineering*, 107:100–104.

Johnson, D. S. (1985). The np-completeness column: an ongoing guide. *Journal of Algorithms*, 6(3):434–451.

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

Kacem, I. (2008). Approximation algorithm for the weighted flow-time minimization on a single machine with a fixed non-availability interval. *Computers & Industrial Engineering*, 54(3):401–410.

Kacem, I. (2009). Approximation algorithms for the makespan minimization with positive tails on a single machine with a fixed non-availability interval. *Journal of Combinatorial Optimization*, 17(2):117–133.

Kacem, I. and Chu, C. (2008a). Minimizing the weighted flow time on a single machine with the resumable availability constraint: worst case of the wspt heuristic. *International Journal of Computer Integrated Manufacturing*, 21(4):388–395.

Kacem, I. and Chu, C. (2008b). Worst-case analysis of the wspt and mwspt rules for single machine scheduling with one planned setup period. *European Journal of Operational Research*, 187(3):1080–1089.

Kacem, I., Chu, C., and Souissi, A. (2008). Single machine scheduling with an availability constraint to minimize the weighted sum of the completion times. *Computers & operations research*, 35(3):827–844.

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

Kacem, I. and Paschos, V. T. (2013). Weighted completion time minimization on a single machine with a fixed non-availability interval: Differential approximability. *Discrete Optimization*, 10(1):61–68.

Kawaguchi, S. and Fukuyama, Y. (2016). Reactive tabu search for job-shop scheduling problems. In *Computer Science & Education (ICCSE), 2016 11th International Conference on*, pages 97–102. IEEE.

Kerrigan, E. C. and Maciejowski, J. M. (2002). Designing model predictive controllers with prioritised constraints and objectives. In *Computer Aided Control System Design, 2002. Proceedings. 2002 IEEE International Symposium on*, pages 33–38. IEEE.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, 220(4598):671–680.

Kirlik, G. and Oguz, C. (2012). A variable neighborhood search for minimizing total weighted tardiness with sequence dependent setup times on a single machine. *Computers & Operations Research*, 39(7):1506–1520.

Laisant, C.-A. (1888). Sur la numération factorielle, application aux permutations. *Bulletin de la Société Mathématique de France*, 16:176–183.

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

Lee, C. and Liman, S. (1992). Single machine flowtime scheduling with scheduled maintenance. *Acta Informatica*, 29(4):375–382.

Lee, J.-Y. and Kim, Y.-D. (2012). Minimizing the number of tardy jobs in a single-machine scheduling problem with periodic maintenance. *Computers & Operations Research*, 39(9):2196–2205.

Lee, K.-F., Masso, A., and Rudd, D. (1970). Branch and bound synthesis of integrated process designs. *Industrial & Engineering Chemistry Fundamentals*, 9(1):48–58.

Lee, Y. H. and Pinedo, M. (1997). Scheduling jobs on parallel machines with sequence-dependent setup times. *European Journal of Operational Research*, 100(3):464–474.

Lehmer, D. H. (1960). Teaching combinatorial tricks to a computer. In *Proc. Sympos. Appl. Math. Combinatorial Analysis*, volume 10, pages 179–193.

Lei, H., Laporte, G., and Guo, B. (2012). A generalized variable neighborhood search heuristic for the capacitated vehicle routing problem with stochastic service times. *TOP*, 20(1):99–118.

Lenstra, J. K., Shmoys, D. B., and Tardos, E. (1990). Approximation algorithms for scheduling unrelated parallel machines. *Mathematical programming*, 46(1-3):259–271.

Leung, J. Y. (2004). Approximation algorithms for minimizing average weighted completion time. In *Handbook of Scheduling*, pages 220–249. Chapman and Hall/CRC.

Li, G., Liu, M., Sethi, S. P., and Xu, D. (2017). Parallel machine scheduling with machine-dependent maintenance periodic recycles. *International Journal of Production Economics*, 186:1–7.

Li, S.-S. and Chen, R.-X. (2017). Scheduling with rejection and a deteriorating maintenance activity on a single machine. *Asia-Pacific Journal of Operational Research*, 34(02):1750010.

Liao, C.-J. and Chen, W.-J. (2003). Single machine scheduling with periodic maintenance and nonresumable jobs. *Computers & Operations Research*, 30(9):1335–1347.

Lin, S. (1965). Computer solutions of the traveling salesman problem. *The Bell System Technical Journal*, 44(10):2245–2269.

Low, C., Hsu, C.-J., and Su, C.-T. (2010). A modified particle swarm optimization algorithm for a single-machine scheduling problem with periodic maintenance. *Expert Systems with Applications*, 37(9):6429–6434.

Luo, J., Wu, Y., and Mendes, A. B. (2016). Modelling of integrated vehicle scheduling and container storage problems in unloading process at an automated container terminal. *Computers & Industrial Engineering*, 94:32–44.

Ma, R. and Yuan, J.-J. (2016). Online scheduling with rejection to minimize the total weighted completion time plus the total rejection cost on parallel machines. *Journal of the Operations Research Society of China*, 4(1):111–119.

Ma, Y., Chu, C., and Zuo, C. (2010). A survey of scheduling with deterministic machine availability constraints. *Computers & Industrial Engineering*, 58(2):199–211.

Martí, R., Moreno-Vega, J. M., and Duarte, A. (2010). Advanced multi-start methods. In *Handbook of metaheuristics*, pages 265–281. Springer.

Mathlouti, I., Gendreau, M., and Potvin, J.-Y. (2018). *A Metaheuristic Based on Tabu Search for Solving a Technician Routing and Scheduling Problem*. CIRRELT, Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport= Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation.

McCall, J. J. (1965). Maintenance policies for stochastically failing equipment: a survey. *Management science*, 11(5):493–524.

Mjirda, A., Todosijevic, R., Hanafi, S., Hansen, P., and Mladenovic, N. (2017). Sequential variable neighborhood descent variants: an empirical study on the traveling salesman problem. *ITOR*, 24(3):615–633.

Mladenovic, N. (1995). A variable neighborhood algorithm: A new metaheuristic for combinatorial optimization. In *Optimization Days*, volume 12.

Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers & operations research*, 24(11):1097–1100.

Moghaddam, A., Amodeo, L., Yalaoui, F., and Karimi, B. (2012). Single machine scheduling with rejection: Minimizing total weighted completion time and rejection cost. *International Journal of Applied Evolutionary Computation (IJAEC)*, 3(2):42–61.

Moreno-Vega, J. M. and Melián, B. (2008). Introduction to the special issue on variable neighborhood search. *Journal of Heuristics*, 14(5):403.

Naderi, B., Zandieh, M., and Ghomi, S. F. (2009). Scheduling sequence-dependent setup time job shops with preventive maintenance. *The International Journal of Advanced Manufacturing Technology*, 43(1-2):170–181.

Nesello, V., Subramanian, A., Battarra, M., and Laporte, G. (2017). Exact solution of the single-machine scheduling problem with periodic maintenances and sequence-dependent setup times. *European Journal of Operational Research*.

Nobibon, F. T. and Leus, R. (2011). Exact algorithms for a generalization of the order acceptance and scheduling problem in a single-machine environment. *Computers & Operations Research*, 38(1):367–378.

Ocampo-Martinez, C., Ingimundarson, A., Puig, V., and Quevedo, J. (2008). Objective prioritization using lexicographic minimizers for mpc of sewer networks. *IEEE Transactions on Control Systems Technology*, 16(1):113–121.

Ou, J. and Zhong, X. (2017). Order acceptance and scheduling with consideration of service level. *Annals of Operations Research*, 248(1-2):429–447.

Ou, J., Zhong, X., and Wang, G. (2015). An improved heuristic for parallel machine scheduling with rejection. *European Journal of Operational Research*, 241(3):653–661.

Papadimitriou, C. H. and Steiglitz, K. (1998). *Combinatorial optimization: algorithms and complexity*. Courier Corporation.

Pearn, W., Chung, S., and Lai, C. (2007). Scheduling integrated circuit assembly operations on die bonder. *IEEE Transactions on electronics packaging manufacturing*, 30(2):97–105.

Pierskalla, W. P. and Voelker, J. A. (1976). A survey of maintenance models: the control and surveillance of deteriorating systems. *Naval Research Logistics Quarterly*, 23(3):353–388.

Pinedo, M. (1995). *Scheduling - Theory, Algorithms, and Systems*. Prentice Hall. ISBN 0137067577.

Pinedo, M. and Chao, X. (1999). *Operations Scheduling with applications in manufacturing and services*. Irwin/McGraw-Hill. ISBN 0072897791.

Pinedo, M. L. (2016). *Scheduling: theory, algorithms, and systems*. Springer.

Posner, M. E. (1985). Minimizing weighted completion times with deadlines. *Operations Research*, 33(3):562–574.

Potts, C. N. and Van Wassenhove, L. N. (1985). A branch and bound algorithm for the total weighted tardiness problem. *Operations research*, 33(2):363–377.

Pozzetti, A., Bilegan, I.-C., Duvivier, D., Artiba, A., Forato, S., Matrat, B., and D'arco, S. (2014). Decision support system for offers of fleet availability services. In *Proceedings of the Ninth Intl. Conference on Systems ICONS'14*, pages 150–153, Nice, France. ISBN 9781612083193.

Prats, X., Puig, V., Quevedo, J., and Nejjari, F. (2010). Lexicographic optimisation for optimal departure aircraft trajectories. *Aerospace Science and Technology*, 14(1):26–37.

Prouff, E. and Schaumont, P. R. (2012). *Cryptographic Hardware and Embedded Systems*, volume 7428. Springer.

Qi, X., Chen, T., and Tu, F. (1999). Scheduling the maintenance on a single machine. *Journal of the Operational Research Society*, pages 1071–1078.

Qi, Y., Wan, L., and Yan, Z. (2015). Scheduling jobs with maintenance subject to load-dependent duration on a single machine. *Mathematical Problems in Engineering*, 2015.

Reeves, C. R., editor (1993). *Modern Heuristic Techniques for Combinatorial Problems*. John Wiley & Sons, Inc., New York, NY, USA.

Respen, J., Zufferey, N., and Amaldi, E. (2016). Metaheuristics for a job scheduling problem with smoothing costs relevant for the car industry. *Networks*, 67 (3):246 – 261.

Rivera Gómez, H., Gharbi, A., and Kenné, J. P. (2013). Joint production and major maintenance planning policy of a manufacturing system with deteriorating quality. *International Journal of Production Economics*, 146(2):575–587.

Rojanasoonthon, S., Bard, J., and Reddy, S. D. (2003). Algorithms for parallel machine scheduling: a case study of the tracking and data relay satellite system. *Journal of the Operational Research Society*, 54(8):806–821.

Roux, O., Duvivier, D., Quesnel, G., and Ramat, E. (2009). Optimization of preventive maintenance through a combined maintenance-production simulation model. In *Proceedings of the International Conference on Industrial Engineering and Systems Management (IESM?09)*, Montreal, Canada. ISBN 9782960053227.

Roux, O., Duvivier, D., Quesnel, G., and Ramat, E. (2013). Optimization of preventive maintenance through a combined maintenance-production simulation model. *International journal of production economics*, 143(1):3–12.

Ruiz, R., García-Díaz, J. C., and Maroto, C. (2007). Considering scheduling and preventive maintenance in the flowshop sequencing problem. *Computers & Operations Research*, 34(11):3314–3330.

Sadfi, C., Kacem, I., and Liu, W. (2009). Lower bounds for total weighted completion scheduling problem with availability constraints. In *Computers & Industrial Engineering*, pages 159–163. IEEE.

Sadfi, C., Penz, B., and Rapine, C. (2002). A dynamic programming algorithm for the single machine total completion time scheduling problem with availability constraints. In *Eighth international workshop on project management and scheduling, (2002), Valence, Spain*.

Sadfi, C., Penz, B., Rapine, C., Błażewicz, J., and Formanowicz, P. (2005). An improved approximation algorithm for the single machine total completion time scheduling problem with availability constraints. *European journal of operational research*, 161(1):3–10.

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

Scholz, D. (2011). *Deterministic global optimization: geometric branch-and-bound methods and their applications*, volume 63. Springer Science & Business Media.

Schulz, A. S. (1996). Scheduling to minimize total weighted completion time: Performance guarantees of lp-based heuristics and lower bounds. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 301–315. Springer.

Shabtay, D., Gaspar, N., and Kaspi, M. (2013). A survey on offline scheduling with rejection. *Journal of Scheduling*, 16(1):3–28.

Shabtay, D., Gaspar, N., and Yedidsion, L. (2012). A bicriteria approach to scheduling a single machine with job rejection and positional penalties. *Journal of Combinatorial Optimization*, 23(4):395–424.

Shmoys, D. B., Stein, C., and Wein, J. (1994). Improved approximation algorithms for shop scheduling problems. *SIAM Journal of Computing*, 23:617–632.

Sierksma, G. and Zwols, Y. (2015). *Linear and integer optimization: theory and practice*. Chapman and Hall/CRC.

Singh, R. and Arvinderjit, S. (2013). Maintenance planning and control of hand tools unit in india: a case study. *International Journal of Indian Culture and Business Management*, 7(1):109–132.

Slotnick, S. A. (2011). Order acceptance and scheduling: A taxonomy and review. *European Journal of Operational Research*, 212(1):1–11.

Slotnick, S. A. and Morton, T. E. (2007). Order acceptance with weighted tardiness. *Computers & Operations Research*, 34(10):3029–3042.

Smith, W. (1956). Various optimizers for single stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66.

Solnon, C., Cung, V. D., Nguyen, A., and Artigues, C. (2008). The car sequencing problem: Overview of state-of-the-art methods and industrial case-study of the roadef?2005 challenge problem. *European Journal of Operational Research*, 191(3):912–927.

Standard, B. (1984). British standard glossary of maintenance management terms in terotechnology. *British Standard Institution, London*.

Sun, K. and Li, H. (2010). Scheduling problems with multiple maintenance activities and non-preemptive jobs on two identical parallel machines. *International Journal of Production Economics*, 124(1):151–158.

Talbi, E.-G. (2009). *Metaheuristics: From Design to Implementation*. Wiley Publishing.

Thevenin, S., Zufferey, N., and Potvin, J.-Y. (2017). Makespan minimisation for a parallel machine scheduling problem with preemption and job incompatibility. *International Journal of Production Research*, 55(6):1588–1606.

Thevenin, S., Zufferey, N., and Widmer, M. (2015). Metaheuristics for a scheduling problem with rejection and tardiness penalties. *Journal of Scheduling*, 18(1):89–105.

Thevenin, S., Zufferey, N., and Widmer, M. (2016). Order acceptance and scheduling with earliness and tardiness penalties. *Journal of Heuristics*, 22(6):849–890.

T'kindt, V. and Billaut, J.-C. (2001). Multicriteria scheduling problems: a survey. *RAIRO-Operations Research*, 35(2):143–163.

T'kindt, V. and Billaut, J.-C. (2006). *Multicriteria scheduling: theory, models and algorithms*. Springer Science & Business Media.

Todosijević, R. (2015). *Contributions théoriques et pratiques pour la recherche dispersée, recherche à voisinage variable et matheuristique pour les programmes en nombres entiers mixtes*. PhD thesis, Valenciennes.

Todosijević, R., Benmansour, R., Hanafi, S., Mladenović, N., and Artiba, A. (2016). Nested general variable neighborhood search for the periodic maintenance problem. *European Journal of Operational Research*, 252(2):385–396.

Veerapen, N. (2012). *Contrôle autonome d'opérateurs pour la recherche locale*. PhD thesis, Université d'Angers.

Vitanyi, P. M. and Li, M. (1997). *An introduction to Kolmogorov complexity and its applications*, volume 34. Springer Heidelberg.

Vivek, P., Saravanan, R., Chandrasekaran, M., and Pugazhenthi, R. (2017). Critical machine based scheduling-a review. In *IOP Conference Series: Materials Science and Engineering*, volume 183, pages 1–7. IOP Publishing.

Wang, G., Sun, H., and Chu, C. (2005). Preemptive scheduling with availability constraints to minimize total weighted completion times. *Annals of Operations Research*, 133(1-4):183–192.

Wang, H. (2002). A survey of maintenance policies of deteriorating systems. *European Journal of Operational Research*, 139:469–489.

Wang, X., Huang, G., Hu, X., and Cheng, T. E. (2015). Order acceptance and scheduling on two identical parallel machines. *Journal of the Operational Research Society*, 66(10):1755–1767.

Williamson, D. P. and Shmoys, D. B. (2011). *The design of approximation algorithms*. Cambridge university press.

Xiuli, W. and TCE, X. X. C. (2013). Order acceptance and scheduling in a two-machine flowshop. *International Journal of Production Economics*, 141(1):366–376.

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

Yang, S., Ma, Y., Xu, D., and Yang, J. (2011). Minimizing total completion time on a single machine with a flexible maintenance activity. *Computers & Operations Research*, 38(4):755–770.

Zhang, G., Zhang, L., Song, X., Wang, Y., and Zhou, C. (2018). A variable neighborhood search based genetic algorithm for flexible job shop scheduling problem. *Cluster Computing*, pages 1–12.

Zhang, S.-X., Cao, Z., and Zhang, Y. (2009). Scheduling with rejection to minimize the total weighted completion time. *International Symposium on Operations Research and Its Applications*, 9:111–114.

Zhong, X. and Ou, J. (2017). Parallel machine scheduling with restricted job rejection. *Theoretical Computer Science*.

Zhong, X., Ou, J., and Wang, G. (2014). Order acceptance and scheduling with machine availability constraints. *European journal of operational research*, 232(3):435–441.

Zhong, X., Pan, Z., and Jiang, D. (2017). Scheduling with release times and rejection on two parallel machines. *Journal of Combinatorial Optimization*, 33(3):934–944.

Zufferey, N. and Vasquez, M. (2015). A generalized consistent neighborhood search for satellite range scheduling problems. *RAIRO-Operations Research*, 49(1):99–121.

Zykina, A. V. (2004). A lexicographic optimization algorithm. *Automation and Remote Control*, 65(3):363–368.