UNIVERSITÉ FRANÇOIS RABELAIS DE TOURS

École Doctorale : MIPTIS

LABORATOIRE D'INFORMATIQUE (EA 6300)

THÈSE présentée par :

Quang Chieu TA

soutenue le : 12 février 2015

pour obtenir le grade de : Docteur de l'université François - Rabelais de Tours

Discipline/ Spécialité : INFORMATIQUE

Matheuristic algorithms for minimizing total tardiness in flow shop scheduling problems

THÈSE DIRIGÉE PAR :

BILLAUT Jean-Charles	Professeur, Université François Rabelais de Tours

CO-DIRECTEUR :

BOUQUARD Jean-Louis	MCF, HDR, Université François Rabelais de Tours

RAPPORTEURS :

ARTIGUES Christian	Directeur de Recherche, LAAS-CNRS, Toulouse
PINSON Eric	Professeur, Université Catholique de l'Ouest

JURY :

ARTIGUES Christian	Directeur de Recherche, LAAS-CNRS, Toulouse
BILLAUT Jean-Charles	Professeur, Université François Rabelais de Tours
BOUQUARD Jean-Louis	MCF, HDR, Université François Rabelais de Tours
CARLIER Jacques	Professeur, Université de Technologie de Compiegne
DELLA CROCE Federico	Professeur, Politecnico di Torino, Italie
PINSON Eric	Professeur, Université Catholique de l'Ouest

Remerciements

Je ne voudrais pas terminer ce travail sans remercier, ici, toutes celles et tous ceux, et ils sont nombreux, qui m'ont aidé à mener à terme cette thèse.

Je veux, tout d'abord et tout particulièrement, remercier et exprimer ma profonde gratitude à mon directeur de thèse, le Professeur Jean-Charles BILLAUT pour toute son aide tout au long de ces trois années de thèse. Ses qualités scientifiques exceptionnelles, mêlées à une gentillesse extraordinaire, font de Jean-Charles la clef de voûte de cette réalisation. Ses conseils avisés ont été très nombreux tout au long de ce travail, et m'ont permis de découvrir les fabuleux plaisirs de la recherche sous ses apparences les plus diverses. Je n'oublierai jamais son soutien et sa disponibilité dans les moments de doute. Je lui suis reconnaissant pour tous ces moments de partage qui ont agrémenté mon parcours.

Je remercie beaucoup le Docteur Jean-Louis BOUQUARD pour avoir co-encadré cette thèse tout au long de son élaboration. Il a toujours été disponible, à l'écoute de mes nombreuses questions, et s'est toujours intéressé à l'avancement de mes travaux. Les nombreuses discussions que nous avons eues, ainsi que ses conseils, sont pour beaucoup dans le résultat final de ce travail.

C'est également avec plaisir que je remercie mes rapporteurs pour le temps qu'ils ont accordé à la lecture de cette thèse et à l'élaboration de leur rapport: Je remercie le Docteur Christian ARTIGUES et le Professeur Eric PINSON d'avoir accepté cette charge. C'est avec joie que je les remercie également pour leurs multiples conseils ainsi que pour l'intérêt qu'il ont porté à mes travaux.

Je remercie le Professeur Federico DELLA CROCE pour ses conseils avisés tout au long de l'élaboration de cette thèse et pour l'honneur qu'il me fait d'être dans mon jury de thèse. Mes remerciements vont au Professeur Jacques CARLIER pour avoir accepté d'être présent dans mon jury de thèse et pour sa participation scientifique ainsi que le temps qu'il a consacré à ma recherche.

Je remercie tout particulièrement l'accueil du laboratoire d'Informatique, de l'école doctorale MIPTIS de l'université François Rabelais, ainsi que ses responsables qui m'ont permis de m'intégrer rapidement et de réaliser mes projets.

Je remercie le Professeur Vincent T'KINDT qui m'a tout d'abord permis d'intégrer l'équipe Ordonnancement et Conduite. Je le remercie également pour ses multiples conseils ainsi que pour l'intérêt qu'il a porté à ma recherche.

Je n'oublie évidemment pas mes collègues du LI avec lesquels j'ai partagé tous ces moments de doutes et de plaisirs. Tous ces instants autour d'un sandwich ou d'un café ont

été autant de moments de détente indispensables pour une complète expression scientifique.

Je remercie au Monsieur Nicolas RAGOT qui m'a beaucoup aidé tout au long de première année en France.

Je voudrais remercier tout particulièrement la famille de Madame Madeleine NICOLET qui non seulement m'a aidé moi, mais aussi ma famille, depuis que nous vivons en France. Elle et sa famille ont toujours été à nos côtés quelles que soient les situations qui se présentaient. De mon tout mon coeur, je lui suis sincèrement reconnaissant et je la considère comme ma mère.

Je n'oublie pas non plus la famille de Madame Nga PAYEN. Elle et son mari Jean-Paul nous ont beaucoup soutenus dans le premier temps en France, surtout pour notre fille. A travers ce papier, je souhaitais également remercier toute sa famille.

Je remercie le Ministère de l'Education et de la Formation du Vietnam à travers le Développement International de l'Education de m'avoir accordé une bourse me permettant d'effectuer mes études de doctorat en France.

C'est également avec plaisir que je remercie les différents organismes qui ont financé ce travail, à savoir l'Agence National de la Recherche (ANR), le Laboratoire d'Informatique de Tours (LI), l'équipe Ordonnancement et Conduite (OC).

Je tiens à remercier l'Ambassade du Vietnam en France, Campus France et le CROUS de Tours qui m'ont aidé dans toute la partie administrative, me permettant ainsi de pouvoir venir étudier en France.

Je remercie l'Université des mines et de géologie de Hanoi pour m'avoir permis de venir en France pour faire mes études.

Je n'oublie pas, dans ces remerciements, l'association Touraine-Vietnam (Jean-Jacques, Gérard, Monique, Françoise,...), Mesdames Mireille GEINGNENAUD, Joële VOUGOUIN de l'Espace Toulouse Lautrec de Tours qui m'avez donné les cours de français, votre aide. Je remercie également l'association des étudiants Vietnamiens de Tours et Blois.

Un grand merci à tous mes amis pour leur amitié, leur présence et leur soutien.

Je tiens à remercier tout particulièrement, et du fond du coeur, toute ma famille en pensant plus spécialement à mes parents, ma petite famille, notamment ma femme et deux jolies petites filles pour tout leur soutien et leurs encouragements tout au long de ma thèse.

12 février 2015, Tours - France

Quang Chieu TA

Résumé

Nous considèrons dans cette thèse un problème d'ordonnancement de flow shop de permutation où un ensemble de travaux doit être ordonnancé sur un ensemble de machines. Les travaux doivent être ordonnancés sur les machines dans le même ordre. L'objectif est de minimiser le retard total. Ce problème se note habituellement $Fm|prmu|\sum T_j$ et il est connu pour être *NP-difficile* au sens ordinaire pour $m = 1$ et au sens fort pour $m \geq 2$ [Lenstra et al., 1977], [Du and Leung, 1990].

Nous proposons des algorithmes heuristiques classiques et des matheuristiques pour ce problème. Les matheuristiques sont un nouveau type d'algorithmes approchés qui ont été proposés pour résoudre des problèmes d'optimisation combinatoire. Les méthodes importent de la résolution exacte au sein des approches (méta) heuristiques. Ce type de méthode de résolution a reçu un grand intérêt en raison de leurs très bonnes performances pour résoudre des problèmes difficiles. Nous présentons d'abord les concepts de base d'un problème d'ordonnancement. Nous donnons aussi une brève introduction à la théorie de l'ordonnancement et nous présentons un panel de méthodes de résolution.

Deuxièmement, nous présentons un état de l'art des méthodes de résolution appliqueés à notre problème.

Troisièmement, nous proposons et décrivons une formulation de programmation linéaire du problème et un algorithme de branch-and-bound. Des conditions de dominance sont utilisées pour couper des noeuds. Nous évaluons ces méthodes avec des jeux de données aléatoires pour des instances de taille petite et moyenne avec deux machines. Nous développons aussi une nouvel borne basée sur une relaxation partielle de l'intégrité des variables du MILP. Cette borne inférieure a de bonnes performances pour de petits instances, mais elle n'est pas efficace pour de grandes instences compte tenu de la taille du MILP et du nombre de variables.

Et puis, de nombreux algorithmes (*NEH, EDD, Recherche en faisceau, Récupération recherche en faisceau, un algorithme génétique et algorithme Tabou*) sont proposés pour résoudre le problème. De nombreux opérateurs de voisinage sont appliqués pour ces méthodes. Ces algorithmes sont évalués pour tester leurs performances avec 108 instances de litérature allant jusqu'à 350 travaux et 50 machines.

Ensuite, nous proposons de nouveaux algorithmes matheuristiques dans le Chapitre 5. Les matheuristics sont basées sur l'insertion de solution partielles exactes dans des algorithmes de voisinage. Plusieurs versions de ces algorithmes sont proposées. Dans ces méthodes, un solveur de MILP est appelé de façon itérative.

Enfin, nous considérons un problème où un flow shop de permutation à m-machine et

un problème de vehicle routing sont intégrés, avec objectif la minimisation de la somme des retards. Nous proposons un codage direct d'une solution et une méthode de voisinage. Les résultats montrent que l'algorithme Tabou améliore grandement la solution initiale donnée par EDD et où chaque voyage ne délivre qu'un travail.

Mots clés : Matheuristique, Ordonnancement, flow shop, retard total, algorithme tabou, algorithme génétique, récupération recherche en faisceau, recherche en faisceau, vehicle routing

Abstract

We consider in this thesis a permutation flow shop scheduling problem where a set of jobs have to be scheduled on a set of machines. The jobs have to be processed on the machines in the same order. The objective is to minimize the total tardiness. This problem, classically denoted by $Fm|prmu|\sum T_j$, is known to be *NP-hard* in the ordinary sense for $m = 1$ and in the strong sense for $m \geq 2$ [Lenstra et al., 1977], [Du and Leung, 1990].

We propose classical heuristic algorithms and matheuristic algorithms for this problem. The matheuristic methods are a new type of approximated algorithms that have been proposed for solving combinatorial optimization problems. These methods embed exact resolution into (meta)heuristic approaches. This type of resolution method has received a great interest because of their very good performances for solving some difficult problems. First, we present the basic concepts and components of a scheduling problem and the aspects related to these components. We also give a brief introduction to the theory of scheduling and present an overview of resolution methods.

Second, we provide the description of the state-of-the-art methods for the problem.

Third, we propose and describe mixed integer linear programming formulations and branch-and-bound algorithms for solving the problem. The dominance conditions are used to prune nodes. We evaluate the methods with a data set small to medium random instances for two machines. We also develop a new hybrid lower bound algorithm based on a partial relaxation of the integrity of variables of the MILP model. This lower bound has good performances for small instances, but is not usable for large instances, due to the size of the MILP and to the number of binary variables introduced.

And then, many algorithms (*NEH, EDD, Beam Search, Recovering Beam Search, Genetic algorithm and Tabu Search algorithm*) are proposed for solving the problem. Many neighborhood operators are applied for the methods. We evaluate the algorithms in order to test their performance with benchmark instances that are proposed, with 108 problems of up to 350 jobs and 50 machines.

Then, we propose the new matheuristic algorithms for solving the problem in the Chapter 5. The matheuristics are based on the insertion of exact partial solution into a neighborhood search algorithm. Several versions of these algorithms are derived. In these methods, the solver for the MILP model is called iteratively.

Finally, we consider a problem where a m-machine permutation flow shop scheduling problem and a vehicle routing problem are integrated and the objective is to minimize the total tardiness. We introduce a direct coding for a complete solution and a Tabu search for finding a sequence and trips. The results show that the TS greatly improves the initial

solution given by EDD heuristic where each trip serves only one job at a time.

Keywords : Matheuristic algorithm, scheduling, flow shop, total tardiness, tabu search, genetic algorithm, recovering beam search, beam search, vehicle routing

Contents

List of Tables

List of Figures

Introduction

"Scheduling is a decision-making process that is used on a regular basis in many manufacturing and services industries. It deals with the allocation of resources to tasks over given time periods and its goal is to optimize one or more objectives" [Pinedo, 2008]. Typically, the result of the allocation is specified in the schedule. For a given set of tasks to schedule and a given set of resources, a schedule is *feasible* if it does not violate any accompanying constraint. Sometimes, finding a single feasible schedule is sufficient. However, under many circumstances, the goal is to find the best possible schedule from among all feasible schedules, to achieve certain objective(s), such as the minimization of the completion time of the last task. Another objective may be the minimization of the number of tasks completed after their respective due dates or the minimization of the total tardiness, etc.

In a scheduling process, the type and amount of each resource is supposed to be known. Boundary of scheduling problem can be efficiently determined if the resources are specified. In addition, each task is described by its resource requirement, its duration, the earliest time at which it may start and the time at which it is due to complete. Any technological constraint (precedence relations for instance) should also be described. Information about resources and tasks define a scheduling problem and its resolution is fairly a complex matter. Resources are usually machines (in a workshop) or computers (in a computing environment) or staff, supposed to perform only one task at a time (also called disjunctive resources). The tasks are called jobs which may be operations in a production process, or executions of computer programs. Sometimes, jobs may consist of several elementary tasks called operations. The environment of the scheduling problem is called a "shop environment", and more precisely a job shop, a flow shop or an open shop environment, depending on the routing of the elementary tasks in the shop. When all the jobs visit the machines of the shop in the same order, the shop is called a *flow shop*. The scheduling problem associated to this environment is called a *flow shop scheduling problem*. A simplified version of this problem exists when it is assumed that the sequence of these jobs in each machine is the same. This type of flow shop is called a *permutation flow shop*. This thesis deals with permutation flow shop scheduling problems.

This type of problem has been widely studied in the literature. The first paper on flow shop scheduling is due to Johnson in 1954 [Johnson, 1954]. Since then, thousands of papers have been published on this category of problems for proposing exact and approximated resolution methods. Furthermore, since several years, a new type of approximated algorithms has been proposed for solving combinatorial optimization problems. These algorithms embed exact resolution into (meta)heuristic approaches. This type of resolution

method has received a great interest because of their very good performances for solving some difficult problems [Maniezzo et al., 2010], [Talbi, 2013]. These methods are called *matheuristics*. An essential feature is the exploitation in some part of the algorithms of features derived from the resolution of the mathematical model of the problem under study. In this thesis, we propose classical heuristic algorithms and matheuristic algorithms for minimizing the total tardiness in a permutation flowshop.

The outline of the thesis is the following. In Chapter 1, we introduce the basic concepts and components of a scheduling problem and the aspects related to these components, a brief introduction to the theory of scheduling and resolution methods. In Chapter 2, the description of the state-of-the-art methods for the problem is provided. We propose and describe mixed integer linear programming formulations and branch-and-bound algorithms for the problem in Chapter 3. The dominance conditions are presented in this chapter and a new hybrid lower bound algorithm for improvement lower bound. The new lower bound is based on a partial relaxation of the integrity of variables of the MILP model. Many heuristic and metaheuristic algorithms are proposed for solving the problem in Chapter 4. Many neighborhood operators are applied in the methods. New matheuristic algorithms are developed for solving the problem in Chapter 5. Several versions of these algorithms have been also derived, depending on how the initial sequence is obtained. In Chapter 6, we have considered a problem where a m-machine permutation flow shop scheduling problem and a vehicle routing problem are integrated to minimize the total tardiness. We have presented a direct coding for a complete solution and a neighborhood method for finding a sequence and trips. We propose a Tabu search algorithm for this problem.

Chapter 1

Problem Formulation

This chapter is the introduction to the basic concepts and components of a scheduling problem (tasks, resources and objective functions) and the aspects related to these components. The chapter is composed of three sections. The first section gives a brief introduction to the theory of scheduling. The second section presents an overview of resolution methods, some of them being used in the rest of the thesis. The third section presents the outline of the thesis.

1.1 Brief introduction to the theory of scheduling

We present in this section definitions, criteria and complexity of some scheduling problems.

1.1.1 Definitions

There are many definitions of a scheduling problem in several books in the literature. We can find the following definitions: "Scheduling is a decision-making process that is used on a regular basis in many manufacturing and services industries. It deals with the allocation of resources to tasks over given time periods and its goal is to optimize one or more objectives"[Pinedo, 2008]. Pinedo also showed the importance of the sequencing and scheduling problems: "scheduling,..., plays an important role in most manufacturing and production systems as well as in most information processing environments. It is also important in transportation and distribution settings and in other types of service industries."

Another definition has been put forward by researchers. Scheduling is defined by Leung as follows: "Scheduling is concerned with the allocation of scarce resources to activities with the objective of optimizing one or more performance measure." [Leung, 2004]

Morton et al. also defined scheduling as follows: "... scheduling is the process of organizing, choosing, and timing resource usage to carry out all the activities necessary to produce the desired outputs at the desired times, while satisfying a large number of time and relationship constraints among the activities and the resources." [Morton and Pentico, 1993]

From the above definitions, we can say that scheduling is a decision-making process that is concerned with the allocation of limited resources or machines over time to carry out a collection of jobs or tasks in which one or several objectives have to be optimized, and satisfying a set of constraints.

Notation of scheduling problems

A scheduling problem is characterized by three elements: a machine environment, a set of constraints and an objective function. These elements are those of the classical three-field notation of [Graham et al., 1979], also called "$\alpha|\beta|\gamma$" notation. The field α describes the machine environment. This field is decomposed into two sub-fields α_1 and α_2 where α_1 denotes the machine organization and α_2 the number of machines. The machine organization can be:

- a single machine, in this case α_1 is empty and $\alpha_2 =' 1'$,
- parallel machines, in this case $\alpha_1 \in \{P, Q, R\}$ to denote respectively identical, uniform or unrelated parallel machines and α_2 is generally equal to 2 if there are two machines, or to m if the number of machines is fixed, or α_2 is empty if the number of machines is unknown,
- a shop organization, in this case $\alpha_1 \in \{F, J, O\}$ where
  - F denotes a flow shop organization where all the jobs have the same routing, i.e. they visit the machines in the same order,
  - J denotes a job shop organization where each job has its own routing in the shop,
  - and O denoted an open shop organization where the routing of the jobs in the shop has to be decided.

The field β describes the constraints of the problem. In this field, it is possible to indicate that jobs have a release date, i.e. a date before which they cannot be started, that jobs are subject to precedence relations, that the sequence of jobs on the machines are all the same (for a flow shop environment, denoted by "perm"), etc. The field γ gives the notation of the objective function that is considered (see Section 1.1.2).

In the following, we treat a permutation flow shop problem with m machines and the objective is to minimize the total tardiness. The notation of this problem is $Fm|perm|\sum T_j$, where $\sum T_j$ is the notation for the total tardiness.

Notations in this thesis

- $N = \{J_1, \ldots, J_n\}$ is the set of n jobs that have to be processed on a set $M = \{M_1, \ldots, M_m\}$ of m machines.
- $p_{i,j} \geq 0$ denotes the fixed *processing time* of job J_j, $J_j \in N$, on machine M_i, $M_i \in M$.
- d_j, $J_j \in N$ is the *due date* for the job J_j.

- C_j is the *completion time* of job J_j, i.e. the completion time of J_j on the last machine of the shop.
- $T_j = \max(0, C_j - d_j)$ is the *tardiness* of job J_j.

1.1.2 Criteria

In the sequencing and scheduling literature, according to [Kan, 1976] and [French, 1982], the criteria for scheduling problems are classified as follows: completion times oriented, due dates oriented, or inventory and machine utilization oriented. There are other criteria obtained by a combination of two or more other criteria.

Another classification of criteria is possible by considering "*minimax*" critera, which represent the maximum value of a set of functions to be minimized, and "minisum" criteria, which represent a sum of functions to be minimized. A summary of the criteria is presented below.

1.1.2.1 The "minimax" criteria

The criteria only based on the completion times:

- $C_{max} = \max_{j=1,\ldots,n} C_j$, is the maximum completion time of jobs. C_{max} is also called the "*makespan*".
- $F_{max} = \max_{j=1,\ldots,n} F_j$ with $F_j = C_j - r_j$ is the maximum time spent in the shop, or even yet, the duration of resting. r_j denotes the release date of the job J_j (supposed to correspond to the arrival time of the job in the shop).

The criteria based on the due dates of jobs are:

- $L_{max} = \max_{j=1,\ldots,n} L_j$, the maximum lateness, with $L_j = C_j - d_j$.
- $T_{max} = \max_{j=1,\ldots,n} T_j$, the maximum tardiness (T_j already defined).
- $E_{max} = \max_{j=1,\ldots,n} E_j$, the maximum earliness, with $E_j = \max(0, d_j - C_j)$.

1.1.2.2 The "minisum" criteria

"Minisum" criteria are usually more difficult to optimize than "minimax" criteria. These criteria are the following.

- $\sum C_j$ denotes $\displaystyle\sum_{j=1}^{n} C_j$ which is the total completion time of jobs.
- $\sum T_j$ denotes $\displaystyle\sum_{j=1}^{n} T_j$ which is the total tardiness of jobs.

- $\sum U_j$ denotes $\sum_{j=1}^{n} U_j$ which is the number of late jobs with $U_j = 0$ if $C_j \leq d_j$, and $U_j = 1$ otherwise.

1.1.3 Complexity of some scheduling problems

"Complexity theory provides a mathematical framework in which computational problems are studied so that they can be classified as "easy" or "hard" [Brucker, 2007]. This classification is useful to see if an efficient algorithm may exist, especially in terms of computation time, for solving a particular problem. A problem belongs to a class of complexity, which informs us of the complexity of the "best algorithm" which is able to solve it. Hence, if a given problem is shown to belong to the class of "easy" problems, it means that it is possible to exhibit a polynomial time algorithm to solve it. Usually this is good news but unfortunately, this does not often happen for complex problems. Accordingly, if a problem belongs to the class of hard problems, it cannot be solved in polynomial time, i.e. the required CPU time to solve it becomes "exponential" [T'kindt and Billaut, 2006].

We refer to [Garey and Johnson, 1979] and to [Papadimitriou, 1994] for the definition of complexity classes and more details on computational complexity.

We give now the complexity of some flowshop scheduling problems. More problems can be found at www.mathematik.uni-osnabrueck.de/research/OR/class.

- $F2||C_{max}$ is the well known two-machine flow shop problem which is polynomially solvable [Johnson, 1954] by the famous Johnson's algorithm,
- $F3||C_{max}$ is *NP-complete* in the strong sense [Garey et al., 1976],
- $F2||\sum C_j$ is *NP-complete* in the strong sense [Garey et al., 1976],
- $F2||\sum T_j$ is *NP-complete* in the ordinary sense [Lenstra et al., 1977],
- $1||\sum T_j$ [Du and Leung, 1990] showed that the problem is *NP-hard* and can be solved in pseudo-polynomial time (NP-complete in the ordinary sense or weakly NP-complete).

The problem that we consider in this thesis, the $Fm|perm|\sum T_j$ problem, is thus NP-complete in the strong sense. Notice also that if there is only one machine, the problem remains NP-complete, and that if the due dates are equal to zero, the problem remains NP-complete (equivalent to $Fm||\sum C_j$ problem). The basic simplifications of this problem are NP-complete problems, which makes this problem one of the more difficult flow shop scheduling problems.

1.2 Resolution methods

In this section, we describe several resolution methods for scheduling problems including exact and heuristics techniques. Many approaches have been proposed to obtain

optimal or near-optimal solutions. Each method has its own performances, measured by the mean quality of the solutions and by its computation time.

The most used exact methods are dedicated branch-and-bound algorithms, dynamic programming algorithms and solvers based on mixed integer linear programming formulations (MILP or MIP). These exact methods return optimal solutions but they can be computationally intensive, even for very small problem instances.

Approximation algorithms and heuristic algorithms are able to solve large problem instances and are able to produce "good" solutions within a reasonable amount of time. These methods can be combined in many ways with one another, forming hybrid techniques.

1.2.1 Exact methods

One of the exact techniques to obtain an optimal solution is the straightforward enumeration, where every possible solution is explored in order to find the optimal solution. However, due to the computational complexity, it is not practical even for problems of moderate sizes. We have therefore to search for more sophisticated resolution methods.

Branch-and-bound

Branch-and-Bound (B&B) is an algorithm for solving hard combinatorial optimization problems. The method was first proposed by [Land and Doig, 1960] for discrete programming and [Ignall and Schrage, 1965] used to find an optimal schedule for the $Fm||C_{\max}$ flow shop problem. The process of solving a problem using B&B algorithm can be described by a *search tree*. Each node of the search tree corresponds to a subset of feasible solutions to a problem. We assume in the following that the B&B algorithm is to find for a minimum value of a function $f(x)$, where x ranges over some set S of candidate solutions. A B&B algorithm has the following characteristics:

- A branching rule that defines partitions of a set of feasible solutions into subsets. From a set S of feasible solutions, the branching returns two or more smaller sets $S_1, S_2, ...$, whose union covers S. The minimum of $f(x)$ over S is the minimum of v_1, v_2,... where each v_i is the minimum of $f(x)$ within S_i. The recursive application of the branching leads to a tree structure (also called the search tree) whose nodes are the subsets of S.

- A lower bounding rule that provides a lower bound, $LB(S)$, on the value of the feasible solutions for any S. A good lower bound (with the highest possible value) may lead to the elimination of an important number of nodes of the search tree, but if its computational requirements are excessively large, it may become advantageous to use a weaker but more quickly computable lower bound.

- A search strategy, which selects the next node to explore. There are three basic search strategies: depth-first (the list of nodes is managed as a LIFO list), breadth-first (the list of nodes is managed as a FIFO list) and best-first (nodes are sorted according to a sorting rule, generally the lower bound value is used).

- An upper bounding method, UB, of the objective value. The objective value of any feasible solution will provide such an upper bound. If the lower bound $LB(S_i)$ of a subproblem S_i is greater than of equal to UB, i.e. $LB(S_i) \geq UB$, then this subproblem cannot yield a better solution and there is no need to continue the branching from this node. We say that we cut this node. To stop the branching process in many nodes, the upper bound UB has to be as small as possible. Therefore, at the beginning of the branch-and-bound some heuristic algorithm may be applied to find a good feasible solution with a small value of the objective function. When the considered node is a leaf of the tree, it corresponds to a feasible solution. If the value of this solution is better than UB then UB is updated and this solution is memorized.

The algorithm stops when the list of nodes to explore is empty.

Example: application of the B&B algorithm to the $F2||\sum C_j$ problem

We consider four jobs and two machines. The data are indicated in the following table.

j	1	2	3	4
$p_{1,j}$	4	2	1	3
$p_{2,j}$	2	5	5	3

The branching is the following. A node S represents a partial beginning sequence and the children nodes of S correspond to the sequence of S completed by an additional node. The bounding rules are the following:

- The initial UB is given by applying Johnson's rule and we obtain $\sum C_j = 47$ and the schedule is $\{J_3, J_2, J_4, J_1\}$.
- At each node, LB is calculated by completing the partial sequence with SPT rule on the second machine. The values of the lower bound are indicated at each node of the tree in Fig. 1.1.
- Finally, we found the optimal solution is $\{J_3, J_1, J_4, J_2\}$ and the value equals to 41.

The exploration strategy is depth-first.

For example, in the first node created in the search tree, job J_1 is scheduled in first position, therefore we note $S = \{J_1\}$. The total completion time of S is equal to $\sum C_j(S) = 6$. For computing the lower bound, one consider jobs J_2 to J_4 in SPT order on the second machine, i.e. in the order (J_4, J_2, J_3). The corresponding completion times are equal respectively to $(9, 14, 19)$. Then, the lower bound associated to $S = \{J_1\}$ is equal to $LB(S) = 6 + 9 + 14 + 19 = 48$. Because the upper bound is equal to 47, there is no chance that this node S can lead to a better solution and therefore, this node is pruned.

Dynamic Programming

Dynamic Programming (DP) was first proposed by Richard Bellman [Bellman, 1957]. It is a complete enumeration method. The main idea of DP is to decompose recursively

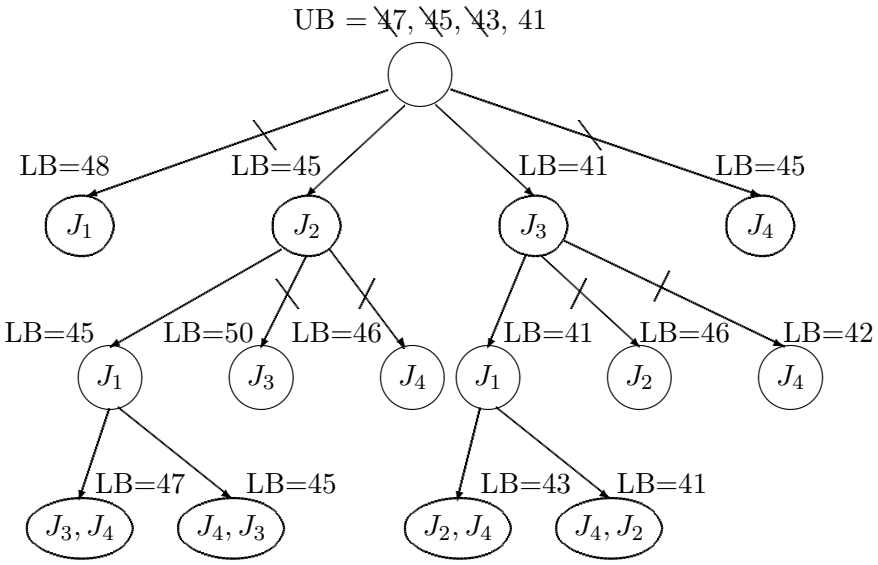


Figure 1.1: Search tree of B&B algorithm

the problem into a series of subproblems, in order to minimize the amount of computation to be done. The approach solves each subproblem and determines its contribution to the objective function. At each iteration, it determines the optimal solution for a subproblem. The solution for the original problem can be deduced from the solution of each subproblem.

A DP formulation is characterized by three types of expressions:

- some initial conditions,
- a recursive relation
- and a goal (i.e. an optimal value function).

Example: application of the DP algorithm to the $F2||\sum C_j$ problem

The resolution of the $F2||\sum C_j$ problem by a DP algorithm is originally proposed in [T'kindt et al., 2003]. We give here a short presentation of this formulation.

Each step of the DP algorithm corresponds to the number of scheduled jobs. At a given step i, we denote by e_i the set of unscheduled jobs and the decision corresponds to the job J_j in e_i that is put in first position in the sequence. We denote by t_1 and t_2 the times at which machine M_1 and machine M_2 become available after processing the jobs preceding e_i and we define $t = t_2 - t_1$. The completion time of J_j is the equal to

$$C_j(t_1, t_2) = \max(t_1 + p_{1,j}, t_2) + p_{2,j} = \max(p_{1,j}, t_2 - t_1) + p_{2,j} + t_1$$

Since for a given set of jobs e_i, t_1 is a constant for all jobs, it does not affect the selection of job J_j. Therefore, we assume that the start time of the first job at machine

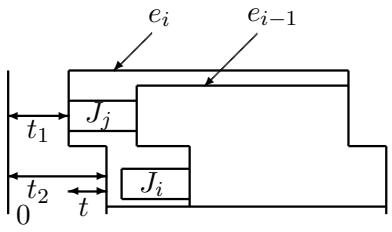


Figure 1.2: DP situation at stage i, state e_i and for decision J_i

M_1 is 0 and we only consider one parameter t in the following. Then, the temporary completion time of J_j is given by:

$$C_j(t) = \max(p_{1,j}, t) + p_{2,j}$$

We denote by $f_i(e_i, t)$ the minimum total completion time of the jobs of e_i if machine M_2 is busy during t time units before the first job of the sequence (i.e. J_j) can start. By selecting job J_j that minimizes the total completion time, we obtain the following recursive relation:

$$f_i(e_i, t) = \min_{J_j \in e_i} \Big[C_j(t) + p_{1,j}(|e_i| - 1) + f_{i-1}\big(e_i - \{J_j\}, C_j(t) - p_{1,j}\big) \Big] \quad \forall i,\ 1 \le i \le n,\ \forall t,\ 0 \le t \le UB$$

$$\Leftrightarrow f_i(e_i, t) = \min_{J_j \in e_i} \Big[f_{i-1}(e_i - \{J_j\}, t') + t' + ip_{1,j} \Big] \quad \forall i,\ 1 \le i \le n,\ \forall t,\ 0 \le t \le UB \text{ with } t' = C_j(t) - p_{1,j}$$

where

$$f_0(\emptyset, t) = 0,\ \forall t,\ 0 \le t \le UB$$

and UB is an upper bound for the makespan criterion.

Let consider the same instance as before. We have the first phase (see Table A.1) with one job. The tables to be done for $e_1 = \{J_1\}$, $e_1 = \{J_2\}$, $e_1 = \{J_3\}$, $e_1 = \{J_4\}$.

Then the second phase where two jobs are scheduled with $e_2 = \{J_1, J_3\}$ is the Table 1.2.

The third phase with $e_3 = \{J_1, J_3, J_4\}$, this table has to be completed with $J_j = J_1$, $J_j = J_3$ and $J_j = J_4$ is the Table 1.3.

At the end, we have the fourth phase with $e_4 = \{J_1, J_2, J_3, J_4\}$ and and only one possible value for $t = 0$. This table has to be completed with $J_j = J_2$ is the Table A.8.

We deduce from this table that the value of the optimal solution is equal to 41. By applying a backtrack algorithm, we found the the optimal solution is $\{J_3, J_1, J_4, J_2\}$.

The detail of all the phases is given in **Appendix A**.

Table 1.1: The first phase of example with one job

e_1	$\{J_1\}$			$\{J_2\}$			$\{J_3\}$			$\{J_4\}$		
J_j	J_1			J_2			J_3			J_4		
$ip_{1,j}$	4			2			1			3		
t	$C_j(t)$	t'	F_1	$C_j(t)$	t'	F_1	$C_j(t)$	t'	F_1	$C_j(t)$	t'	F_1
0	6	2	6	7	5	7	6	5	6	6	3	6
1	6	2	6	7	5	7	6	5	6	6	3	6
2	6	2	6	7	5	7	7	6	7	6	3	6
3	6	2	6	8	6	8	8	7	8	6	3	6
4	6	2	6	9	7	9	9	8	9	7	4	7
5	7	3	7	10	8	10	10	9	10	8	5	8
6	8	4	8	11	9	11	11	10	11	9	6	9
7	9	5	9	12	10	12	12	11	12	10	7	10
8	10	6	10	13	11	13	13	12	13	11	8	11
9	11	7	11	14	12	14	14	13	14	12	9	12
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 1.2: The second phase of example with two jobs that are scheduled

e_2	$\{J_1, J_3\}$							
J_j	J_1				J_3			
$ip_{1,j}$	8				2			
t	$C_j(t)$	t'	F_1	F_2	$C_j(t)$	t'	F_1	F_2
0	6	2	7	17	6	5	7	14^*
1	6	2	7	17	6	5	7	14^*
2	6	2	7	17	7	6	8	16^*
3	6	2	7	17^*	8	7	9	18
4	2	2	7	17^*	9	8	10	20
5	7	3	8	19^*	10	9	11	22
6	8	4	9	21^*	11	10	12	24
7	9	5	10	23^*	12	11	13	26
8	10	6	11	25^*	13	12	14	28
9	11	7	12	27	14	13	15	30
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Mixed Integer Linear Programming

Mixed Integer Linear Programming (MILP or MIP) is a phase of modelization of a problem under the following form [Bixby et al., 2000] and then the resolution of this

Table 1.3: The third phase of example with three jobs that are scheduled

e_3	$\{J_1, J_3, J_4\}$											
J_j	J_1				J_3				J_4			
$ip_{1,j}$	12				3				9			
t	$C_j(t)$	t'	F_2	F_3	$C_j(t)$	t'	F_2	F_3	$C_j(t)$	t'	F_2	F_3
0	6	2	17	31	6	5	17	25*	6	3	17	29
1	6	2	17	31	6	5	17	25*	6	3	17	29
2	6	2	17	31	7	6	19	28*	6	3	17	29
3	6	2	17	31	8	7	21	31	6	3	17	29*
4	6	2	17	31	9	8	23	34	7	4	17	30*
5	7	3	17	32*	10	9	25	37	8	5	19	33
6	8	4	19	34*	11	10	27	40	9	6	21	36
7	9	5	21	37*	12	11	29	43	10	7	23	39
8	10	6	23	40*	13	12	31	46	11	8	25	42
9	11	7	25	43*	14	13	33	49	12	9	27	45
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 1.4: The final phase of example with four jobs that are scheduled

e_4	$\{J_1, J_2, J_3, J_4\}$															
J_j	J_1				J_2				J_3				J_4			
$ip_{1,j}$	16				8				4				12			
t	$C_j(t)$	t'	F_3	F_4	$C_j(t)$	t'	F_3	F_4	$C_j(t)$	t'	F_3	F_4	$C_j(t)$	t'	F_3	F_4
0	6	2	25	43	7	5	32	45	6	5	32	41*	6	33	33	48

problem by using a solver:

$$\begin{aligned} &\text{Minimize } c^T x \\ &\text{subject to } Ax \geq b \\ &\qquad\qquad l \leq x \leq u \\ &\qquad\qquad \text{some or all } x_j \text{ integral} \end{aligned}$$

where A is an $m \times n$ matrix, call the *constraint matrix*, x is a vector of n *variables*, c is the vector of coefficients of the *objective function*, and l and u are vectors of bounds. Thus, an MILP is a linear program (LP) plus an integrality restriction on some or all of the variables.

Example: application to the $F2||\sum C_j$ problem

The following example is an MILP model for the $F2||\sum C_j$ [Della Croce et al., 2011] problem. This model is based on positional variables . We define the binary variables $x_{j,k}$,

equal to 1 if job J_j is in position k, 0 otherwise; $C_{k,1}$ and $C_{k,2}$, denote the completion times of the job in position k on M_1 and M_2, and $C_k \geq 0$ the completion time of the job in position k. The MILP is the following.

$$\text{Minimize } \sum_{k=1}^{n} C_{k,2} \tag{1.1}$$

$$\text{subject to } \sum_{k=1}^{n} x_{j,k} = 1, \ \forall j \in \{1, ..., n\} \tag{1.2}$$

$$\sum_{j=1}^{n} x_{j,k} = 1, \ \forall k \in \{1, ..., n\} \tag{1.3}$$

$$C_{1,1} = \sum_{j=1}^{n} p_{1,j} x_{j,1} \tag{1.4}$$

$$C_{1,2} = C_{1,1} + \sum_{j=1}^{n} p_{2,j} x_{j,1} \tag{1.5}$$

$$C_{k,1} = C_{k-1,1} + \sum_{j=1}^{n} p_{1,j} x_{j,k}, \ \forall k \in \{2, ..., n\} \tag{1.6}$$

$$C_{k,2} \geq C_{k,1} + \sum_{j=1}^{n} p_{2,j} x_{j,k}, \ \forall k \in \{2, ..., n\} \tag{1.7}$$

$$C_{k,2} \geq C_{k-1,2} + \sum_{j=1}^{n} p_{2,j} x_{j,k}, \ \forall k \in \{2, ..., n\} \tag{1.8}$$

$$\text{variables } C_{k,1} \geq 0, C_{k,2} \geq 0, \ \forall k \in \{1, ..., n\} \tag{1.9}$$

$$x_{j,k} \in \{0, 1\}, \ \forall j \in \{1, ..., n\}, k \in \{1, ..., n\} \tag{1.10}$$

Constraints (1.2) and (1.3) ensure that there is one job per position and one position per job. Constraints (1.4) and (1.6) compute the completion times on machine M_1. Constraints (1.5), (1.7) and (1.8) determine the completion times on machine M_2. This model has n^2 binary variables and $2n$ continuous variables, and $5n - 1$ constraints.

1.2.2 Heuristic and metaheuristic methods

Heuristics

Heuristic methods are algorithms which are used to find solutions among all possible ones, but these algorithms do not guarantee that the best solution will be found. Therefore, they are considered as approximated algorithms. They usually find quickly and easily a solution close to the optimal one. On some instances, these algorithms may return optimal solutions.

An example is the NEH heuristic algorithm [Nawaz et al., 1983] for the $Fm||C_{\max}$ problem. The idea of the NEH algorithm is the following: the jobs are sorted in non-increasing

order of the sum of the processing times on machines, then a solution is obtained in a constructive way, adding at each step a new job in that order, inserting it at the best place, i.e. the one that results in the best partial solution. NEH algorithm is now described in details.

Algorithm 1 NEH algorithm

1: Calculate total P_j for each job, where $P_j = \sum_{i=1}^{m} p_{i,j}$, $\forall j = 1 \ldots, n$
2: Sort the jobs in non-increasing order of P_j
3: Select the first two jobs and choose the sequence S of these two jobs with minimum C_{max} value.
4: **for** $k = 3$ **to** n **do**
5: Test the insertion of the next job at any possible position in S from position 1 to position $k + 1$
6: Keep the insertion with minimum C_{max} value.
7: Update S
8: **end for**

Metaheuristics: Tabu search and Genetic algorithm

The formal definition of metaheuristics is based on a variety of definitions from different authors, the following definition seems to be most appropriate: "A metaheuristic is an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high quality solutions. It may manipulate a complete (or incomplete) single solution or a collection of solutions at each iteration. The subordinate heuristics may be high (or low) level procedures, or a simple local search, or just a construction method. The family of metaheuristics includes, but is not limited to, tabu search, ant systems, greedy randomized adaptive search, variable neighborhood search, evolutionary methods, genetic algorithms, simulated annealing, and their hybrids." [Vop et al., 1999], [Maniezzo et al., 2010]

Tabu search

Tabu Search (TS) has been initially proposed by Glover [Glover, 1989], [Glover, 1990]. TS is a metaheuristic local search algorithm that begins with an initial solution and successively moves to the best solution in the neighborhood of the current solution. The algorithm maintains a list of forbidden solutions, to prevent the algorithm from visiting solutions already examined (these solutions are called *tabu*). The general TS algorithm is described below.

Notations:

- S_0 is an initial solution,
- S is the current solution,

- f is the value of solution S,
- S^* is the best solution,
- f^* is the value of S^*,
- $N(S)$ is the whole neighborhood of S,
- $N'(S)$ is the neighborhood of S which is not tabu,
- T is the tabu list.

Algorithm 2 Tabu Search algorithm

1: $S = S_0$, $f^* = f(S_0)$, $S^* = S_0$, $T = \varnothing$.
2: **while** Termination criterion not satisfied **do**
3: Select S in argmin$[f(S')]$, where S' is a solution in $N'(S)$,
4: **if** $f(S) < f^*$ **then**
5: $f^* = f(S)$, $S^* = S$,
6: **end if**
7: Record the current move in the tabu list T (delete oldest entry if necessary).
8: **end while**

Example:

The example deals with the resolution of the two-machine flowshop scheduling problem with minimization of the total tardiness ($F2||\sum T_j$). We apply the technique for five iterations, we fix the length of the Tabu list (T) to 2. The data are given below.

j	1	2	3	4
$p_{1,j}$	1	2	6	3
$p_{2,j}$	5	4	2	3
d_j	11	10	9	7

- *Initialization*
  - Tabu list $T = \varnothing$, $S_0 = \{J_4, J_3, J_2, J_1\}$ is an initial solution obtained by sorting the jobs in EDD order.
  - $S = S_0$, $f^* = f(S_0) = 14$, $S^* = S_0$,
  - The neighborhood of S is defined by all the schedules that can be obtained with adjacent pairwise interchanges.
- *Iteration 1*
  - The three neighbors of S_0 are shown in Fig. 1.3. The values of $\sum T_j$ are indicated for each neighbor.

- The current best neighbor is $\{J_4, J_2, J_3, J_1\}$
- Let $S = \{J_4, J_2, J_3, J_1\}$ and $f(S) = 11$
- $f(S) < f^*$, set $f^* = 11$, $S^* = \{J_4, J_2, J_3, J_1\}$
- Move $= (J_3, J_2)$, update tabu list $T = \{(J_3, J_2)\}$

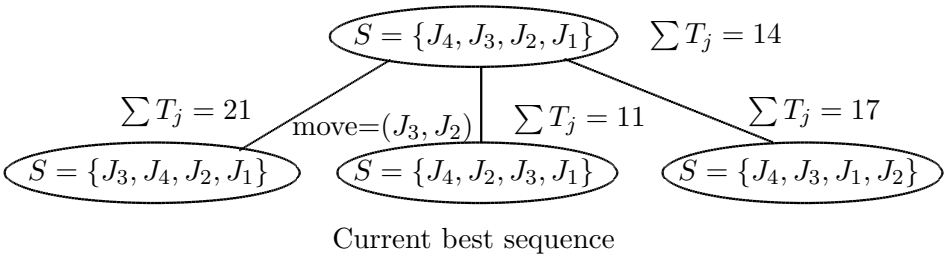


Figure 1.3: Iteration 1: Sequences of S by adjacent pairwise interchange

- *Iteration 2*
  - The computations of $\sum T_j$ of the neighbor sequences are reported in Fig. 1.4.
  - The current best sequence is $\{J_4, J_2, J_1, J_3\}$
  - Let $S = \{J_4, J_2, J_1, J_3\}$ and $f(S) = 12$
  - $f(S) > f^*$, f^* remains at value 11
  - Move $= (J_3, J_1)$ is added to the tabu list $T = \{(J_3, J_2), (J_3, J_1)\}$

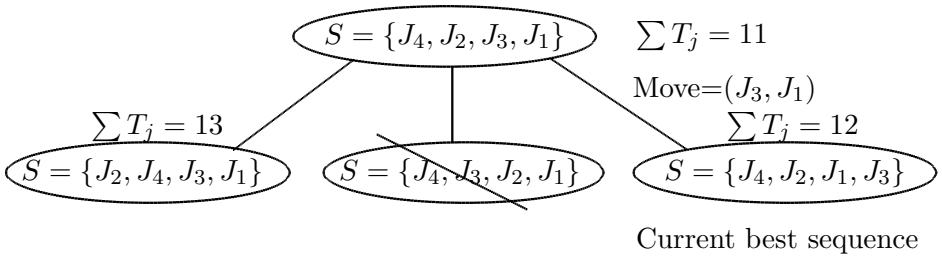


Figure 1.4: Iteration 2: Sequences of S by adjacent pairwise interchange

- *Iteration 3*
  - The computations of $\sum T_j$ of the neighbor sequences are reported in Fig. 1.5.
  - The current best sequence is $\{J_2, J_4, J_1, J_3\}$
  - Let $S = \{J_2, J_4, J_1, J_3\}$ and $f(S) = 12$
  - $f(S) > f^*$, f^* remains at value 11
  - Move $= (J_4, J_2)$. Since length of T is 2, we update tabu list, remove (J_3, J_2) from the list and add (J_4, J_2) to the list. The new tabu list is $T = \{(J_3, J_1), (J_4, J_2)\}$

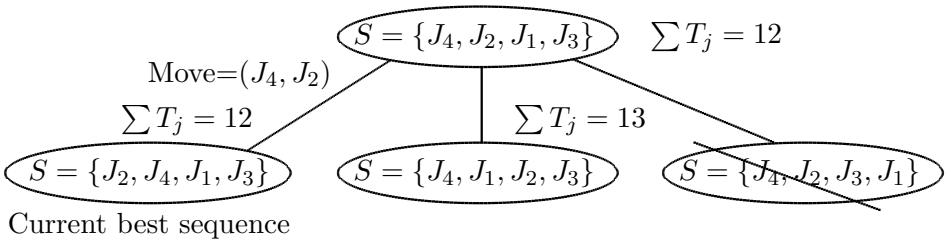


Figure 1.5: Iteration 3: Sequences of S by adjacent pairwise interchange

- *Iteration 4*
  - The computations of $\sum T_j$ of the neighbor sequences are reported in Fig. 1.6.
  - The current best sequence is $\{J_2, J_1, J_4, J_3\}$
  - Let $S = \{J_2, J_1, J_4, J_3\}$ and $f(S) = 14$
  - $f(S) > f^*$, f^* remains at value 11
  - Remove (J_3, J_1) from list and add (J_4, J_1) to list. New tabu list $T = \{(J_4, J_2), (J_4, J_1)\}$

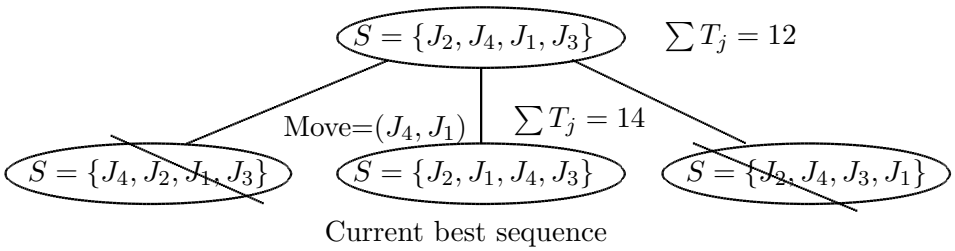


Figure 1.6: Iteration 4: Sequences of S by adjacent pairwise interchange

- *Iteration 5*
  - The computations of $\sum T_j$ of the neighbor sequences are reported in Fig. 1.7.
  - The current best sequence is $\{J_1, J_2, J_4, J_3\}$
  - Let $S = \{J_1, J_2, J_4, J_3\}$ and $f(S) = 12$
  - $f(S) > f^*$, f^* remains at value 11
  - Remove (J_4, J_2) from list and add (J_2, J_1) to list. New tabu list $T = \{(J_4, J_1), (J_2, J_1)\}$

At the end of iteration 5, the best overall sequence is $\{J_4, J_2, J_3, J_1\}$ and its value is equal to 11.

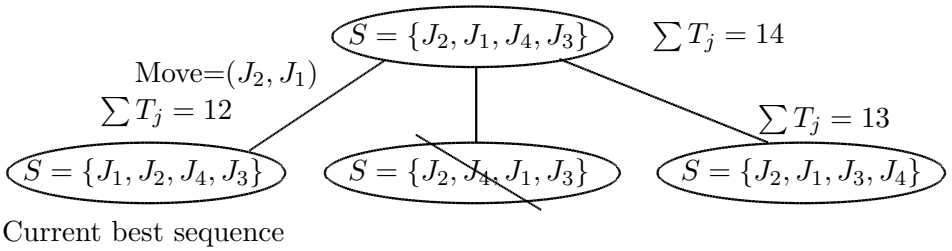


Figure 1.7: Iteration 5: Sequences of S by adjacent pairwise interchange

Genetic algorithms

Genetic algorithms (GA) have been originally proposed by Holland [Holland, 1975]. This is a general search technique where a population composed by individuals evolves following nature inspired mechanisms called "genetic operators". The population is composed by individuals that are evaluated by a fitness, which is often related to the objective function.

Starting from an initial population, new solutions are generated by selecting some "parents" randomly, but with a probability growing with fitness, and by applying genetic operators such as *selection*, *crossover* and *mutation*, which introduce random modifications. Some existing solutions are randomly selected for crossover, some solutions are selected for mutation, and a new population of the same size is obtained. The process is repeated until a given stopping criterion is reached, e.g. a time limit or when a sufficiently satisfactory solution has been found.

Genetic algorithms have been largely used for solving scheduling problems. According to [Goldberg, 1989], the main steps of a genetic algorithm are:

1. Generation of the *initial population* P_0,
2. Evaluation of the *fitness* of each individual,
3. *Selection* of the individual couples in population P_{k-1},
4. Application of the *crossover* operator: with a probability p_c, two individuals of P_{k-1} will be replaced by two new individuals in P_k,
5. Application of the *mutation* operator: with a probability p_m, each individual is modified by a mutation,
6. Replace population P_{k-1} by population P_k: P_k contains the $PopSize$ best individuals of $P_{k-1} \cup M_k \cup C_k$.
7. and repeat the process at step 2 until a stopping condition is satisfied.

Therefore, a genetic algorithm is designed by several parameters of high importance. First of all, there are several ways for coding a solution. Coding is important because it is used for the crossover operator and mutation operators. In scheduling, when solving the

problem is equivalent to find a sequence, it is generally convenient to consider that an individual is exactly a sequence. This is called in the literature "direct encoding" because an individual corresponds to a solution without ambiguity. For more complicated scheduling problems such as job shop or parallel machine problems, an individual may represent a list of jobs, but an algorithm has to be used to determine the corresponding solution. This is called in the literature "undirect encoding" because an individual does not correspond "immediately" to a solution. The other key points in a genetic algorithm are the crossover and the mutation operators. The literature contains a lot of definitions, strongly related to the coding definition. For classical scheduling problems, the most famous crossover operators are 1-point crossover up to k-point crossover. Mutation generally consists in changing arbitrarily an element of an individual. Fixing the probabilities p_c of crossover and p_m of mutation is not an easy task. It is generally done after some preliminary computational experiments on a subset of the data set. A survey of the applications of genetic algorithms to scheduling problems can be found in [Portmann and Vignier, 2008].

1.3 Outline of the thesis

To summarize, we consider in this thesis a permutation flow shop scheduling problem where a set $N = \{J_1, ..., J_n\}$ of n jobs have to be scheduled on a set $M = \{M_1, ..., M_m\}$ of m machines. Each job J_j has to be processed on machine M_i and then on machine M_{i+1} $(i = 1, ..., m-1)$. The objective is to minimize the total tardiness denoted by $\sum T_j = \sum_{j=1}^{n} T_j$. This problem, classically denoted by $Fm|prmu|\sum T_j$, is known to be *NP-hard* in the ordinary sense for $m = 1$ and in the strong sense for $m \geq 2$ [Lenstra et al., 1977], [Du and Leung, 1990].

The thesis is organized as follows: In the Chapter 2, the description of the state-of-the-art methods for solving the problem is provided. The exact methods and (meta)heuristic approaches in the literature for the flow shop scheduling problem with the minimization of the total tardiness are reviewed. We propose and describe mixed integer linear programming formulations and branch-and-bound algorithms for the problem in Chapter 3. Dominance conditions are used to prune nodes. We also develop a new hybrid lower bound algorithm for improving the basic lower bounds. The new lower bound is based on a partial relaxation of the integrity of variables of the MILP model. We evaluate the methods with random data sets with small to medium instances for the case of two machines. Many heuristic and metaheuristic algorithms are proposed for solving the problem in Chapter 4. Many neighborhood operators are applied on these methods. The performance of these algorithms is tested with benchmark instances. New matheuristic algorithms are developed for solving the problem in Chapter 5. Several versions of these algorithms are derived and evaluated. In Chapter 6, we consider a problem where a m-machine permutation flow shop scheduling problem and a vehicle routing problem are integrated, and the objective is to minimize the total tardiness. We present a direct coding for a complete solution and a neighborhood method for finding a sequence and trips. We propose a Tabu search algorithm for this problem, and the results show that the TS greatly improves the initial solution given by EDD heuristic and where each trip serves only one job at a time.

Chapter 2

Literature review

We present in this chapter the literature review concerning the flow shop scheduling problem with the minimization of the total tardiness. Exact methods and heuristic approaches are described. Notice that a state-of-the-art survey has recently been published on this topic [Vallada et al., 2008].

2.1 Introduction

Permutation flow shop sequencing problem with makespan minimization is a known *NP-hard* problem for m machines ($m \geq 3$). It is already strongly *NP-hard* for two machines for the total completion time minimization, it means that only exhaustive search guarantees to find an optimal permutation. But finding an optimal permutation can become prohibitively expensive, even for small instances [Pinedo, 1995]. In the following, we focus on *permutation flow shop*, i.e. flow shop where the sequence of jobs is imposed to be the same on all the machines. Such schedules are not dominant, but the restriction of this family of schedules reduces the search to only one permutation sequence. In this chapter, we review the most important existing methods, from the classical exact methods to the most recent and effective heuristics and metaheuristic methods for the minimization of the total tardiness. A new type of approximated algorithms, including exact resolution inside heuristic approaches, has received a great interest in the literature, because of their very good performances on some difficult problems [Maniezzo et al., 2010],[Talbi, 2013]. These methods are called *matheuristics* and these techniques are also reported below, for a particular scheduling problem.

2.2 Exact methods

Exact methods have been used for the minimization of the total tardiness for permutation flow shop scheduling problems. Due to the complexity of flow shop scheduling problems, it appears that exact methods are impracticable for instances of more than a few jobs and/or machines.

Sen et al [Sen et al., 1989] developed a branch-and-bound procedure that uses the

tardiness of the jobs in a partial sequence as a lowed bound, an ordering criterion, that is a dominance condition DC-SDG and a branch-and-bound algorithm that builds a sequence from the back to the front. This dominance condition is the following:

DC-SDG: Let $S = \sigma_1 J_i J_j \sigma_3$ be a sequence and $S' = \sigma_1 J_j J_i \sigma_3$ the same sequence with J_j before J_i. $\sum T_j(S) \leq \sum T_j(S')$ if $d_i \leq d_j$, $p_{2,i} - d_i \leq p_{2,j} - d_j$ and $p_{1,i} \leq \min(p_{2,i}, p_{1,j})$.

The results are compared to greedy algorithms implementing, the earliest due date (EDD), the shortest processing time (SPT) and the minimum slack (SLACK) dispatching rules. The experiments were carried out using a set of 640 problem instances generated using a scheme similar to the one proposed in [Fisher, 1976]. The processing times for both machines are randomly generated from a uniform distribution over the values 1 and 10. The due dates are also randomly generated from a uniform distribution between $P(1 - T - R/2)$ and $P(1 - T + R/2)$ where T and R are two parameters called "inherent tardiness factor" and "due date range" respectively. The value P is commonly a lower bound on the makespan, but in this case it is the sum of the processing times in machine M_2 plus the smallest processing time in machine M_1. Four problem sizes $n \in \{6, 8, 10, 12\}$ were proposed with $T \in \{0.25, 0.5, 0.75, 1\}$ and $R \in \{0.25, 0.5, 0.75, 1\}$. Therefore, 64 combinations are generated, each of which is repeated 10 times. The results show that the SPT heuristic with pairwise improvement phases performs very well for large T values and the number of nodes processed by the branch-and-bound to find an optimal solution tends to increase with increasing values of T and R.

Kim [Kim, 1993b] considers the two-machine flow shop scheduling problem with total tardiness minimization. He developed a lower bound based on the sum of two lower bounds computed from the set of jobs in the partial sequence and the set of jobs not included in it, respectively. Dominance rules DC-K1 and DC-K2 are also presented. The aim of these dominance conditions is to reduce the size of the search tree.

DC-K1: Let S be a sequence of type $S = \sigma J_i J_j \pi$ where J_i and J_j are two consecutive jobs, and σ and π two sub-sequences. Let $S' = \sigma J_j J_i \pi$ be the same sequence except for the positions of jobs J_i and J_j.

$$\sum T_j(S) \leq \sum T_j(S') \Leftarrow d_i \leq d_j \textit{ and } p_{1,i} \leq \min(p_{2,i}, p_{1,j}) \textit{ and } p_{2,i} - d_i \leq p_{2,j} - d_j$$

DC-K2: Let WC_{max} be the worst makespan that an active schedule may have. The inverse Johnson's sequence has such a makespan. Each job J_j with $d_j \geq WC_{max}$ is scheduled last.

To compare this algorithm with the algorithm given in [Sen et al., 1989], the author generated randomly 240 problem instances with the number of jobs ranges from 10 to 15 to test the performance of the method. Processing times were uniformly distributed between 1 and 30 and due dates of the jobs were computed using two parameters, R (due-date range) and T (tardiness factor) by using the method of [Potts and Van Wassenhove, 1982]. The due dates are generated from a discrete uniform distribution in $[P(1 - T - R/2), P(1 - T + R/2)]$, where P is the sum of the processing times of all operations divided by two (the number of machines). Several combinations of the parameters T, R and the number of jobs were considered, with $T \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$, $R \in \{0.8, 1, 1.2, 1.4, 1.6, 1.8\}$ and $n \in \{10, 11, 12, 13, 14, 15\}$. The author compares the proposed branch-and-bound using the lower bounds and dominance rules presented in the paper, to the branch-and-bound

proposed by [Sen et al., 1989] and a similar branch-and-bound including the dominance rule presented in [Sen et al., 1989]. The results show that the bounds and dominance rules presented in this paper are much more powerful than the algorithms of Sen et al.

In [Kim, 1995], the author developed a branch-and-bound for the m-machine total tardiness flow shop scheduling problem. The depth first exploration strategy is used, the lower bound is computed for each node and a procedure to check the existence of dominating sequences is applied at the root node in order to reduce the problem size. The algorithm was tested using 480 randomly generated instances where the processing time of the jobs are uniformly distributed between 1 and 30 and the due dates were generated randomly. The results show that the proposed algorithm solved all 20 instances considered by each group until $n = 13$ jobs and $m = 8$ machines. It was also able to solve all the instances of the group $n = 14$ and $m = 4$ machines. Regarding the special sets of instances, the branch-and-bound proposed was not able to solve all the 20 problems within the CPU limit time set to 3600 seconds. In order to test the results, the algorithm is compared to the branch-and-bound for the two-machine case presented in [Sen et al., 1989], since the authors did not find any other work to solve the m-machine problem optimally. In this case, 120 problems were generated with $n \in \{10, 11, 12, 13, 14, 15\}$ and 20 instances for each value of n. The proposed branch-and-bound was able to solve optimally all the problem instances generated while the branch-and-bound proposed in [Sen et al., 1989] did not solve any of the 20 instances with more than 13 jobs.

A branch-and-bound for the two-machine flowshop problem to minimize the total tardiness is proposed by [Pan and Fan, 1997]. The authors presented several dominance properties for the precedence relations between jobs in an optimal solution, among them `DC-PF1` and `DC-PF2` are given below.

`DC-PF1`: For any two jobs J_i and J_j, if $p_{1,i} \leq p_{1,j}$, $p_{2,i} = p_{2,j}$ and $d_i \leq d_j$, then there exists an optimal sequence such that J_i precedes J_j.

`DC-PF2`: Let $S = \sigma_1 J_i J_j \sigma_3$ be a sequence and $S' = \sigma_1 J_j J_i \sigma_3$. $\sum T_j(S) \leq \sum T_j(S')$ if $p_{1,i} \leq \min(p_{2,i}, p_{1,j})$, $p_{2,i} \leq p_{2,j}$ and $d_i \leq d_j$.

A lower bound on the total tardiness of a subproblem is presented by the authors, which improves Kim's lower bound. The performance of the proposed algorithm is compared with the method proposed by [Kim, 1993b]. A total of 600 problems was randomly generated such that the processing times were uniformly distributed over the range $1-10$. Due dates were computed from another uniform distribution as previously done with P the sum of the processing times of all the jobs on machine M_2 plus the minimum processing time among all the jobs on machine M_1 [Sen et al., 1989]. A total of 60 sets of problems was considered and for each one 10 replicates were generated randomly. The results showed that the branch-and-bound proposed outperformed that presented by [Kim, 1993b] in terms of average time and number of problems solved. The proposed method was able to solve all the problems with up to 16 jobs and those with 18 jobs. Finally, this paper improved the solution efficiency in comparison to Kim's algorithm, by using the dominance rules and the better quality of the lower bound developed.

In [Pan et al., 2002], the authors proposed another branch-and-bound algorithm to minimize the total tardiness in a two-machine flow shop, where the sequence is built from front to back. New dominance conditions (`DC-PCC1` to `DC-PCC4`) are derived to determine

the precedence constraints between jobs in optimal solution, and a new lower bound on the total tardiness of a subproblem is presented. The initial solution is given by the EDD rule. Four dominance conditions are given bellow:

`DC-PCC1`: For job J_j, if there is a job J_i satisfying $p_{2,j} \leq p_{2,i}$, $p_{1,i}+p_{2,i} \leq p_{1,j}+p_{2,j}$ and $p_{2,i} - d_i \geq p_{2,j} - d_j$, then an optimal sequence exists where J_j is not first in the sequence.

`DC-PCC2`: Let $S = \sigma_1 J_i J_j \sigma_3$ be a sequence and $S' = \sigma_1 J_j J_i \sigma_3$. $\sum T_j(S) \leq \sum T_j(S')$ if $C(J_i|\sigma_1 J_j) - d_i \leq 0$, $C(J_j|\sigma_1 J_i) - d_j \leq 0$ and $C(J_j|\sigma_1 J_i) \leq C(J_i|\sigma_1 J_j)$ (where $C(\alpha|\beta)$ denotes the completion time of α if it is sequenced after β).

`DC-PCC3`: Let $S = \sigma_1 J_i \sigma_2 J_j \sigma_3$ be a sequence and $S' = \sigma_1 J_j \sigma_2 J_i \sigma_3$. $\sum T_j(S) \leq \sum T_j(S')$ if $C(J_i|\sigma_1) - d_i \geq 0$, $C(J_j|\sigma_1) - d_j \geq 0$, $C(J_j|\sigma_1) \geq C(J_i|\sigma_1)$, $p_{1,j} \geq p_{1,i}$ and $p_{2,j} \leq p_{2,i}$.

`DC-PCC4`: Let $S = \sigma_1 J_i J_j \sigma_3$ be a sequence and $S' = \sigma_1 J_j J_i \sigma_3$. $\sum T_j(S) \leq \sum T_j(S')$ if $C(J_i|\sigma_1) - d_i \geq 0$, $C(J_j|\sigma_1) - d_j \geq 0$, $C(J_j|\sigma_1) \geq C(J_i|\sigma_1)$ and $\min(p_{1,j}, p_{2,i}) \geq p_{1,i}$.

The computational experiments were carried out using 600 randomly generated problems where the processing times are uniformly distributed between 1 and 10. The due dates are also uniformly distributed as in the previous studies where P is the sum of the processing times in machine M_2 plus the smallest processing time in machine M_1. Five different problem sizes with $n \in \{16, 18, 20, 22, 24\}$ were tested according to the parameters T, R and n with $T \in \{0.25, 0.5, 0.75\}$, $R \in \{0.25, 0.5, 0.75, 1\}$. Therefore, 60 combinations are generated; each one is repeated 10 times. The proposed branch-and-bound solved optimally all the replicates up to 24 jobs with $T = 0.5$ and $R = 0.5$. A problem was considered not solved if an algorithm needed more than 3600 seconds of elapsed time to obtain the optimal solution. The results of the proposed algorithm are compared to the branch-and-bound presented by Pan and Fan [Pan and Fan, 1997] which was able to solve optimally all the 10 replicates up to 20 jobs where $T = 0.25$ and $R = 0.75$. Finally, the proposed method performed better than the method proposed by Pan and Fan [Pan and Fan, 1997] in terms of average CPU time and number of problems solved.

In [Schaller, 2005], Schaller presented a branch-and-bound algorithm for the two-machine flow shop scheduling problem with the objective of minimizing the total tardiness. They improved three dominance conditions and a lower bound of Pan et al [Pan et al., 2002]. A new dominance rule `DC-S1` was also developed to help reducing the search tree in this algorithm:

`DC-S1`: Let $S = \sigma_1 J_i J_j \sigma_3$ be a sequence and $S' = \sigma_1 J_j J_i \sigma_3$. There exists an optimal schedule such that job J_i is in the position immediately following J_j if $C_j(S) - d_j \leq 0$, $C_i(S') - d_i \leq 0$ and $C_j(S) \geq C_i(S')$

Three branch-and-bound algorithms were presented by combining the different dominance conditions and lower bounds. A depth first strategy is used and the algorithms were tested on randomly generated problems where the processing time of the jobs was computed using a uniform distribution between 1 and 10. The due dates are also randomly generated in the same way in [Pan et al., 2002], [Pan and Fan, 1997]. In total, 12 sets were generated and each problem set consists of 10 problems, where $T \in \{0.5, 0.75\}$, $R \in \{0.5, 1\}$ and $n \in \{16, 18, 20\}$. The CPU time limit was set to 20 minutes, if a method was not able to solve the problem within this time limit, the algorithm was terminated and the best objective value found was returned. The computation and comparison showed

that the last two of three proposed algorithms solved 80 problem instances with $n = 16, 18$ while the first method solved 78 of the 80 instances. For 40 instances with $n = 20$, the second method solved 34 instances, the third one solved 36 instances while the first one solved only 32 instances. The results showed that using the replacement dominance conditions, the tighter lower bound and the new dominance condition improved the branch-and-bound algorithm's efficiency.

Billaut [Billaut, 2006] presented new dominance conditions for the $F2||\sum T_j$ problem and proved that some dominance conditions of the literature were dominated. The results shown that one of them is better than some existing dominance conditions.

`DC-B1`: Let $S = \sigma_1 J_i J_j \sigma_3$ be a sequence and $S' = \sigma_1 J_j J_i \sigma_3$. If $\sum T_j(S) \leq \sum T_j(S')$, then the node with J_j before J_i can be pruned.

Chung et al [Chung et al., 2006] also developed a branch-and-bound algorithm for the total tardiness m-machine flow shop problem. The authors proposed a machine-based lower bound and a dominance rule for pruning nodes. They used depth first exploration and a backtracking strategy for this algorithm. Computational experiments were carried out using problem instances randomly generated. Twelve combinations of n and m values are tested with $n \in \{10, 15, 20\}$ and $m \in \{2, 4, 6, 8\}$. The authors proposed a set of 45360 problem instances where 19440 instances are for $n = 10$, 19440 instances are for $n = 15$ and 6480 instances for $n = 20$. The results are compared to the branch-and-bound proposed by Kim [Kim, 1995] and it shows that their algorithm outperforms the method of Kim both in terms of CPU time and the number of instances solved.

In [Billaut and Zhang, 2007], the authors proposed new dominance conditions and a new lower bound which are implemented in a branch-and-bound for the $F2||\sum T_j$ problem. For the computational experiments, the authors generated randomly 20 instances with $n \in \{12, 14, 16\}$, according to the definition given by [Pan and Fan, 1997]. The results prove the interest of the new tools proposed and the new proposed lower bound remains comfortable when the number of jobs increases.

To conclude this presentation of exact methods, we can say that the exact methods in the literature are limited to problems with up to 20 jobs and 8 machines. So to solve larger problems of more realistic sizes in these conditions, it is necessary to find a solution method providing of good quality in a reasonable amount of time. That is the reason why heuristic and metaheuristic methods have been proposed in the literature. In the next section, we review the most well known heuristics and metaheuristics for this problem.

2.3 Heuristic algorithms

Since the permutation flow shop problem is an NP-hard problem with computations being prohibitively expensive, a practical approach is to use heuristic methods to obtain near-optimal solutions. A heuristic algorithm is a method which finds "good" solutions in an acceptable computation time, but without being able to ensure the optimality of the solution. Heuristic algorithms can be broadly classified into *dispatching rules*, *constructive and improvement* heuristics and *metaheuristics*. In the next subsections we present some

heuristics according to this classification.

2.3.1 Dispatching rules

Sequencing and scheduling problems can be solved using dispatching rules. A dispatching rule assigns a priority to each job, which is based on the attributes of the jobs. Then, jobs are sorted according to this rule and the job with the highest priority is selected to be processed first. These rules are applied very often for finding an initial solution in some heuristic and metaheuristic methods. Several dispatching rules are presented below:

- EDD (*Earliest Due Date*): jobs are sorted in the due date non decreasing order, i.e. $d_{[1]} \leq d_{[2]} \leq ... \leq d_{[n]}$ where $d_{[k]}$ is the due date of the job in position k.
- SPT (*Shortest Processing Time*): jobs are sorted in the processing time non decreasing order, i.e. $p_{[1]} \leq p_{[2]} \leq ... \leq p_{[n]}$ where $p_{[k]}$ is the processing time of the job in position k. In a flow shop context, one has to specify on which processing time the rule is applied, it can be for the processing time on machine M_1, on machine M_2, the sum of processing times, etc.
- LPT (*Longest Processing Time*): jobs are sorted in the processing time non increasing order, i.e. $p_{[1]} \geq p_{[2]} \geq ... \geq p_{[n]}$ where $p_{[k]}$ is the processing time of the job in position k.
- SLACK: During the construction of a schedule, it is possible to compute the completion time C_j of each candidate job J_j. The slack of job J_j is equal to $s_j = d_j - C_j$. The candidate jobs are sorted in non decreasing order of s_j. Generally, the job with the smallest slack is selected, the completion times and the slacks of the remaining jobs are updated and the process iterates until all the jobs have been scheduled.
- MDD (*Modified Due Date*): Similarly to the SLACK rule, the job are sorted in non decreasing order of $\max(d_j, C_j)$.

2.3.2 Constructive and Improvement heuristics

Constructive heuristics build a schedule from scratch by making a series of iterations through a list of unscheduled jobs. A each iteration, one or more jobs are selected and added to the schedule. Each decision is taken and remains unchanged later. Once a complete sequence has been obtained, it is returned.

Contrary to constructive heuristics, *improvement heuristics* start from an existing solution and apply some improvement procedures. Several constructive and improvement heuristics are presented now.

The NEH algorithm is proposed by Nawaz et al [Nawaz et al., 1983] for the permutation flow shop scheduling problem with m machines and the makespan minimization. This detailed algorithm is given in Section 1.2.2. In the first step of the NEH algorithm, jobs are sorted in the decreasing order of their total processing time. The NEH algorithm starts by taking the first two jobs and the best sequence of these two jobs is used as an

initial (and partial) sequence. Then, each job in the order of the list is inserted at the best possible position in the partial list. The final list is returned.

The CDS algorithm is proposed by Campbell, Dudek and Smith [Campbell et al., 1970] for solving the m-machine flow shop scheduling problem with makespan minimization. The problem is solved by using $m-1$ dummy two-machine flow shop problems. In sub-problem k, dummy processing times are defined for M_1: $p'_{1,j} = \sum_{i=1}^{k} p_{i,j}$ and for M_2: $p'_{2,j} = \sum_{i=m-k+1}^{m} p_{i,j}$. Each sub-problem leads to a Johnson's sequence, that is evaluated for the whole problem with m machines. At the end, the best sequence is returned.

In [Kim, 1993a], the author used an improvement heuristic which starts with the EDD sequence. Furthermore, a different way to sort the jobs is proposed where they are sorted in non-increasing order of due dates, that is, latest due date (LDD) rule. Theses dispatching rules are used to generate the initial sequence for the NEH heuristic and lead to NEH_{edd} and NEH_{ldd} algorithms, respectively. Moreover, the author also analyzed to modify priority/dispatching rules for the mean tardiness objective: EDD, SLACK, SRMWK and MDD. Then, an improvement heuristic which starts from the solution given by the EDD rule is proposed. The initial solution is improved by interchanging pairs of jobs and this heuristic is called ENS. All these methods are compared to 1000 randomly generated test problems where the processing times are generated from a uniform distribution with a range from 1 to 35 and the due dates are generated following the method proposed in [Potts and Van Wassenhove, 1982] as mentioned before. The *tardiness factor* (T) was tested from 0.1 to 0.5 by step 0.1 and the *due date range* (R) from 0.8 to 1.8 by step 0.2. Several problem sizes were proposed with $n \in \{15, 20, 30, 40, 50\}$ and $m \in \{5, 10, 15, 20, 25\}$. The results showed that the NEH_{edd} heuristic had a good performance, but the best results were obtained by the ENS method.

In [Raman, 1995], the authors considered and developed several rules and heuristic algorithms for the single machine and two-machine cases and evaluated these algorithms for the m-machine flow shop problem. The method presented in [Ow, 1985] is extended to consider m machines such that each machine in the flow shop is considered as a bottleneck machine and a complete schedule is developed following the method presented in [Ow, 1985]. At the end of the process, there are m possible sequences and the best one is selected. This approach is called the Modified Focused Scheduling (MFS) method. In an extensive computational study, the authors compared these methods with others and show that the improvement procedure is very effective.

In [Koulamas, 1998], the authors presented an efficient shifting bottleneck algorithm (SBFT) for the two-machine flowshop total tardiness problem by exploiting its relationship to the single machine problem. The computational results show that the method is very good. They compared the performance of the SBFT and NEH heuristic with $n \in \{25, 50, 100\}$. Four problems are generated and solved for each value of n, the relative range of due dates is $RDD \in \{0.2, 0.4, 0.6, 0.8, 1\}$ and the tardiness factor is $TF \in \{0.2, 0.4, 0.6, 0.8\}$. The results indicate that SBFT could be declared optimal in many cases and obtain in general very good solutions. The SBFT solution never exceeds the optimal solution by more than a predetermined value, and they showed that this bound is tight.

In [Chakraborty and Laha, 2007], the authors developed and modified the NEH algo-

rithm [Nawaz et al., 1983]. The proposed method is easy to implement and improves the quality of the solution while maintaining the same computational complexity as NEH. In the computational experiments, this method was run on 28 different problem sizes with a number of jobs $n \in \{12, 18, 24, 30, 40, 50, 100\}$ and a number of machines $m \in \{5, 10, 15, 20\}$. 15 instances were created for each problem size. Their results show that the proposed algorithm outperforms NEH.

2.4 Metaheuristics

2.4.1 Simulated annealing

An integrate algorithm based on simulated annealing and Tabu search is presented in [Adenso-Díaz, 1996], starting from the solution of Ow in [Ow, 1985], which provides the initial solution. The original parameters of the method are changed after some computational experiments and a simulated annealing algorithm is applied to improve it. This solution is used as the initial one by a Tabu search method where the size of the neighbourhood is restricted. The author compares the solution obtained to a Tabu search starting from the same initial solution without any limitation in the size of the neighbourhood. The comparison is carried out with 720 randomly generated problems with $n \in \{10, 15, \ldots, 50\}$ and $m \in \{5, 10, 15, 20\}$. The results show that the same final solution is obtained, but the number of iterations is smaller in the case of the hybrid algorithm using a limited size of the neighborhood.

Parthasarathy et al, in [Parthasarathy and Rajendran, 1998] presented two heuristics based on the simulated annealing method for the flow shop and flowline-based manufacturing cell scheduling problem, to minimize the mean tardiness of jobs. They compared the proposed heuristics with the heuristic proposed by [Gelders and Sambandam, 1978] and [Kim, 1993a], by generating 30 problem instances with $n \in \{10, 15, 20, 25, 30\}$ and $m \in \{5, 10, 15, 20, 25, 30\}$. The processing times of jobs are generated from the discrete uniform distribution in the range [1,99]. The proposed heuristics perform better than those proposed by [Gelders and Sambandam, 1978] and [Kim, 1993a].

In [Hasija and Rajendran, 2004], a simulated annealing method is proposed to minimize the total tardiness of jobs. The authors use a specific heuristic for the initial solution [Parthasarathy and Rajendran, 1998], which is improved by applying two perturbation schemes. The first perturbation is the same method as the one proposed in [Parthasarathy and Rajendran, 1998], the second one is based on swap of jobs. The authors compared the performance of the proposed heuristic with the TS proposed by [Armentano and Ronconi, 1999] and the SA proposed by [Parthasarathy and Rajendran, 1998]. The test problems to evaluate the performance of the methods were based on the benchmark instances of [Taillard, 1993]. The due date of job J_j is given by $d_j = T_j \times [1 + u \times 3]$, where $T_j = \sum_{i=1}^{m} p_{i,j}$ is the sum of the processing time of job J_j; u is a uniform random number over the range [0, 1]. Several combinations of number of jobs $n \in \{20, 50, 100\}$ and number of machines $m \in \{5, 10, 20\}$ were considered. The results show that the proposed SA algorithm obtains better results than the SA described in [Parthasarathy and Rajendran, 1998] and the TS described in [Armentano and Ronconi, 1999].

2.4.2 Tabu search

Several Tabu Search (TS) methods have been proposed for solving the permutation flow shop scheduling problem.

In [Taillard, 1990], Taillard has shown that the problem may be solved efficiently by TS technique, which can give better solutions than the NEH algorithm. He also shown that a random pairwise swapping is computationally more expensive compared to a random insertion method and that a random pairwise swapping does not yield to a better convergence to the optimal solution than the random insertion method.

In [Nowicki and Smutnicki, 1996], the authors provided a special method based on the Tabu Search algorithm with a reduced neighborhood search and using a modified NEH algorithm for the initial solution. The method works faster and more efficiently than other know algorithms. The authors use block properties to explore the different sequences and the insertion of jobs in the adjacent blocks is promising. By virtue of these properties, the authors were able to eliminate a considerable number of moves, thereby reducing the search. The authors also employ the back jump approach in which, if there is no change in the solution for a specific number of iterations, the algorithm restarts using the current best solution, to create neighboring solutions. This is known as the diversification scheme in TS terminology.

In [Ben-Daya and Al-Fawzan, 1998], the authors proposed a tabu search approach which suggests simple techniques for generating neighborhoods of a given sequence and a combined scheme for intensification and diversification, that has not been considered before. The authors use the NEH heuristic to generate the initial solution for their TS approach. They generated neighborhood structure by using a compromise by alternating random insertion, block insertion, and random swapping in a random fashion. This means that at each iteration of the neighborhood generation process, one of the three methods is selected randomly to generate the next neighbor of the current sequence. For selecting the best neighbor in the candidate list, the authors chose the first sequence that improved the makespan for the next iteration. A Tabu list with capacity 7 was used in this approach. The authors construct a frequency matrix that records the number of times each job is located in a particular position among all the examined sequences. A sequence constructed using this matrix is used to randomly restart the neighborhood search process. Their results improve the previous implementations of the Tabu search due to Taillard [Taillard, 1990] and simulated annealing algorithm due to [Ogbu and Smith, 1990].

In [Armentano and Ronconi, 1999], the authors propose a heuristic based on Tabu search for flow shop scheduling problem to minimize the total tardiness. They compare their method with the NEH heuristic and the optimal branch-and-bound algorithm of [Kim, 1995], and showed that all the versions have an excellent performance. The authors presented also strategies such as diversification, intensification and analyzed the neighborhood restriction. The association of the diversification and intensification can improve the results of some problems with the same computation time.

In [Grabowski and Wodecki, 2004], the authors used some new properties of a classic flow shop scheduling problem, to minimize the makespan. They propose a new and very fast local search based on a Tabu search method. The authors reduced calculations for

the solution in the search of the neighborhood by using a lower bound on the makespan instead of computing the real makespan. The authors also propose a dynamic Tabu list that contains ordered pair of jobs and with a dynamic length. The tabu list is initiated empty and the oldest element in the list is deleted if no favorable moves are obtained in a particular iteration and the iteration is continued. Moreover, the size of the list can also be changed at each iteration. The authors also applied a perturbation mechanism that allows the algorithm to escape from a local optimum. This mechanism allows to diversify the search space and to visit new regions, where "good solutions" can be found. The perturbation mechanism is activated when there is no improvement in the solution after a fixed number of iterations. This parameter in the algorithm is chosen experimentally. The authors compared their algorithm with the best speed and accuracy which have been obtained by TS approach of [Nowicki and Smutnicki, 1996], [Grabowski and Wodecki, 2001] and by GA approach of [Reeves and Yamada, 1998]. They tested their algorithm on benchmark instances proposed in [Taillard, 1990]. The authors found that their algorithm was outperforming the algorithm of [Nowicki and Smutnicki, 1996] when the problem size increases. They also observed that their algorithm was much faster than the existing algorithms while finding comparable solutions.

2.4.3 Genetic algorithm

In [Onwubolu and Mutingi, 1999], the authors propose a genetic algorithm for the m-machine flow shop scheduling problem, considering the total tardiness, the number of tardy jobs and the bi-objective problem. The initial population is generated randomly with the *popsize* equal to 20. They defined the crossover operator with a probability (*pcross*) of 0.70 and a mutation operator (*pmutate*) of 0.40. In order to evaluate each individual of the solution space, the authors used a fitness function which applies the mapping process of [Goldberg, 1989]. The authors combined also replacement strategies which has been formulated and presented in the past. In this study, the authors evaluated also the objective function and advanced the best performing individuals into the next state. This technique prevents the lost of the best solutions and allows to obtain better results than [Gupta et al., 1993]. The authors compared their results with another metaheuristic and show that it is both more computationally effective and efficient than simulated annealing [Zegordi et al., 1995]. The results show that the GA can obtain near-optimal solutions for the flow shop problem and large problem instances, with an acceptable computation time.

In [Ruiz et al., 2006], the authors presented two GA with makespan minimization for the permutation flow shop scheduling problem (this paper is cited here because we have been inspired by this method). The first method, they applied is the "DOE approach" of [Montgomery, 2000] to set the parameters and the operators of the GA. An initial population is based on NEH heuristic and on a modification of this method. The authors also proposed four new crossover operators that perform better than many other methods. Furthermore, they used a restart scheme, reinitializing a given portion of the population, therefore reintroducing diversity in the population and allowing to escape from local optimal. In the second method, a local search is applied to some individuals after selection, crossover and mutation. The two proposed methods were compared to 11 other methods such as GA, TS, SA and other recent advanced algorithms. For the evaluations, new data

sets have been generated using standard benchmark instances of Taillard [Taillard, 1993] with 120 different problem instances with 20 jobs and 5 machines to 500 jobs and 20 machines. The stopping criterion is equal to $n(m/2)t$ milliseconds, where $t = 90$. The obtained results showed that the proposed algorithms outperform all the other compared algorithms and they are easy to implement.

In [Vallada et al., 2008], the author reviewed and evaluated the heuristics and metaheuristics for the minimization of the total tardiness for the m-machine flow shop scheduling problem. A total of 40 different methods, 23 heuristics and dispatching rules and 17 metaheuristics were implemented and tested with the same benchmark of instances containing 540 test problems with up to 350 jobs and 50 machines. The configuration of tardiness factor (T), due date range (R), number of jobs (n) and number of machines (m) are $T \in \{0.2, 0.4, 0.6\}$, $R \in \{0.2, 0.6, 1\}$, $n \in \{50, 150, 250, 350\}$ and $m \in \{10, 30, 50\}$. The limit time of CPU is set for all the metaheuristic to $n(m/2)/90$ milliseconds. The authors proved that the heuristic algorithms which are based on the insertion and interchange of jobs have the best performances. They also concluded that TS methods are good metaheuristics for permutation flow shop scheduling problem with the minimization of total tardiness.

In [Vallada and Ruiz, 2010], the authors developed three genetic algorithms to minimize the total tardiness for permutation flow shop scheduling problem. The authors used advanced techniques such as path relinking, local search and procedures to diversify the population. In the local search, a job is removed from the sequence and inserted in any possible position. The job is finally placed at the better position in terms of total tardiness value. To ensure the diversity of the population, the authors compute a value for each job, based on the number of times this job appears in the different positions in the solutions of the population. In addition, they applied a speed up procedure in order to reduce the computation effort for the local search technique. In the computational experiments, the instances are generated by the procedure of [Vallada et al., 2008]. Each algorithm is tested with a set of 24 randomly generated test instances. The three proposed algorithms are compared with the best existing methods and the results show that it is the better existing algorithm for the minimization of total tardiness.

2.5 Matheuristics

Since several years, a new type of approximated algorithms, including exact resolution inside heuristic approaches, has received a great interest in the literature, because of their very best performances on some difficult problems [Maniezzo et al., 2010], [Talbi, 2013]. These methods are called *matheuristics*. In [Della Croce et al., 2011] the authors introduce an MIP resolution into a neighborhood search algorithm for the $F2||\sum C_j$ problem. The authors presented a two-step method: the first step consists in finding an initial solution by using a recovering beam search algorithm and in the second step, an intensive neighborhood search is performed. In the second step also, the structure of the neighborhood should be as much as possible "orthogonal" to the structure of the neighborhood used in the first step, in order to prevent from a local optimum. In the computational experiments, 20 instances are generated as in [Della Croce et al., 2004] with $n \in \{100, 300, 500\}$ and processing times

randomly generated in the range [1,100]. The proposed method is compared with RBS, ILS (*Integrate Local Search*) and ACO (*Ant Colony Optimization*) for $n = 100$. In other cases ($n = 300, 500$), this method is compared with RBS and ILS. The results showed that their method outperform the RBS, ILS, ACO ($n = 100$) and RBS, ILS for $n = 300, 500$. But ILS performed better than their procedure on one instance over 20 (for $n = 500$). The obtained results demonstrated that an hybrid approach implementing a post-optimization refinement procedure by means of an MILP solver is able to achieve valuable results that are better than those given by existing heuristics.

Chapter 3

Exact methods

In this chapter, we propose and describe *mixed integer linear programming* formulation and *branch-and-bound* algorithms for the problem.

3.1 Mixed Integer Linear Programming (MILP)

We propose a MILP formulation of the problem based on positional variables (firstly introduced in [Wagner, 1959]): $x_{j,k}$ is equal to 1 if job J_j is in position k, and 0 otherwise. $C_{i,k}$ is the completion times of the job in position k on machine M_i, and $T_k \geq 0$ is the tardiness of the job in position k. The MILP is the following:

$$\text{Minimize } \sum_{k=1}^{n} T_k \tag{3.1}$$

$$\text{subject to } \sum_{k=1}^{n} x_{j,k} = 1,\ \forall j \in \{1..n\} \tag{3.2}$$

$$\sum_{j=1}^{n} x_{j,k} = 1,\ \forall k \in \{1..n\} \tag{3.3}$$

$$C_{1,1} = \sum_{j=1}^{n} p_{1,j} x_{j,1} \tag{3.4}$$

$$C_{1,k} = C_{1,k-1} + \sum_{j=1}^{n} p_{1,j} x_{j,k},\ \forall k \in \{2..n\} \tag{3.5}$$

$$C_{i,1} = C_{i-1,1} + \sum_{j=1}^{n} p_{i,j} x_{j,1},\ \forall i \in \{2..m\} \tag{3.6}$$

$$C_{i,k} \geq C_{i-1,k} + \sum_{j=1}^{n} p_{i,j} x_{j,k},\ \forall i \in \{2..m\},\ \forall k \in \{2..n\} \tag{3.7}$$

$$C_{i,k} \geq C_{i,k-1} + \sum_{j=1}^{n} p_{i,j} x_{j,k},\ \forall i \in \{2..m\},\ \forall k \in \{2..n\} \tag{3.8}$$

$$T_k \geq C_{m,k} - \sum_{j=1}^{n} d_j x_{j,k},\ \forall k \in \{1..n\} \tag{3.9}$$

$$\text{variables } C_{i,k},\ \forall i \in \{1..m\},\ \forall k \in \{1..n\};\ T_k,\ \forall k \in \{1..n\} \tag{3.10}$$

$$x_{j,k} \in \{0,1\},\ \forall j \in \{1..n\}, k \in \{1..n\} \tag{3.11}$$

Constraints (3.2) and (3.3) ensure that there is exactly one job per position and one position per job. Constraints (3.4) and (3.6) compute the completion times on machine M_1. Constraints (3.5), (3.7) and (3.8) determine the completion times on machine M_i, $\forall i \geq 2$. Constraints (3.9) determine the tardiness of job in position k. This model contains n^2 binary variables, $n(m+1)$ continuous variables and $[(2n-1)(m+1)+2]$ constraints.

3.2 Branch and Bound (B&B)

3.2.1 B&B for the m-machine permutation flowshop scheduling problem

In this section, we consider a B&B algorithm for the problem. A B&B algorithm searches the complete space of solutions for a given problem for the best solution (see **Section 1.2.1**). The detail of this branch-and-bound algorithm is given in **Algo. 3**.

Algorithm 3 Branch-and-bound algorithm

```
1: Initiation
2: UB ← Calculate initial solution using heuristic method (EDD, NEH,...)
3: R_0 ← Calculate root node
4: L is the list of nodes which have to be explored
5: while (L not empty) do
6:    Extract a node N ∈ L
7:    LB(N) ← calculate LB of node N
8:    if LB(N) < UB then
9:       if N is a leaf then
10:         UB ← LB(N)
11:      else
12:         Calculate the children or child nodes
13:         Insert them into L
14:      end if
15:   end if
16: end while
17: return UB (optimal value)
```

Our algorithm uses a *depth first search* strategy for the following reasons:

- the number of active nodes is always less than or equal to n,
- the bottom of the tree is reached faster, allowing potentially good upper bounds to be found earlier,
- a stack can be used to make easy the computations.

Upper bound

We use the heuristic method called EDD (*Earliest Due Date*) for computing the total tardiness of the upper bound. The current UB will be replaced by the LB of a complete solution (a leaf of a tree search is generally a feasible solution and its LB corresponds to the objective function) if total tardiness of LB is less than current UB.

Lower bound

For each partial sequence, a lower bound can be obtained, consisting of two independent bounds, that are calculated from two mutually exclusive sets of jobs: (1) the set of jobs included in the *partial sequence* $(\sigma) = (\sigma(1), ..., \sigma(s))$, indicating that job $\sigma(j)$ occupies the jth position on each machine, for $1 \leq j \leq s$, where $1 \leq s \leq n$ and (2) the set of jobs not included $(\overline{\sigma}) = (\overline{\sigma}(1), ..., \overline{\sigma}(n-s))$ in it. The sum of these two bounds for node $(\sigma\overline{\sigma})$ $= (\sigma(1), ..., \sigma(s), \overline{\sigma}(1), ..., \overline{\sigma}(n-s))$ serves as a lower bound on the total tardiness.

- **A bound for the jobs in** (σ)

  Since the sequence of jobs in a partial sequence is defined, the sum of tardinesses of the jobs for the partial sequence (σ) can be calculated as follows:

$$\begin{aligned} C_{i,j} &= \max(C_{i-1,j}, C_{i,j-1}) + p_{i,j}, \\ T_j &= \max(0, C_{m,j} - d_j), \\ &\forall i \in \{1, ..., m\},\ \forall j \in \{1, ..., s\} \end{aligned}$$

  remember that $C_{i,j}$, and $p_{i,j}$ are the completion time and the processing time of job J_j on machine M_i and d_j is the due date of job J_j. The value of the total tardiness of σ is denoted by $LB(\sigma)$.

- **A bound for the jobs in** $(\overline{\sigma})$

  The lower bound of the jobs in $(\overline{\sigma})$ can be calculated as follows. First, we build m pseudo data sets for $(\overline{\sigma})$ where the data set number k is defined as follows:

  - for $i \in \{1, ..., k-1\}$ and for $j \in \{s+1, ..., n\}$: $p_{i,j} = \min_{j' \in \overline{\sigma}} p_{i,j'}$
  - for $i \in \{k+1, ..., m\}$ and for $j \in \{s+1, ..., n\}$: $p_{i,j} = \min_{j' \in \overline{\sigma}} p_{i,j'}$
  - processing times of the jobs in $\overline{\sigma}$ on machine M_k are sorted in SPT order: $p_{k,s+1} \leq p_{k,s+2} \leq ... \leq p_{k,n}$
  - due dates of the jobs in $\overline{\sigma}$ are sorted in EDD order: $d_{s+1} \leq d_{s+2} \leq ... \leq d_n$

The total tardiness of the jobs in $\overline{\sigma}$ is computed, assuming that these jobs are sequenced after σ in the order $(s+1, ..., n)$. The value of the total tardiness for the pseudo data set number k is denoted by $LB(\overline{\sigma})^{(k)}$. The value of the lower bound of the total tardiness of $\overline{\sigma}$ is:

$$LB(\overline{\sigma}) = \max_{k \in \{1,...,m\}} LB(\overline{\sigma})^{(k)}$$

The lower bound associated to the node $(\sigma\overline{\sigma})$ is equal to $LB(\sigma) + LB(\overline{\sigma})$.

3.2.2 Lower bounds for 2-machine permutation flow shop scheduling problem

In this section, we propose some classical lower bounds for the $F2||\sum T_j$ problem [Ta et al., 2013a]. Then, a lower bound based on a partial relaxation of the MILP is proposed.

3.2.2.1 Classical lower bounds

We consider here the computation of lower bounds at the root node of a search tree procedure. The lower bounds that we present in this paragraph are very classical in the scheduling literature (see [Pan and Fan, 1997] for example).

From a given instance I, two dummy instances $I1$ and $I2$ are generated.

Instance $I1$ is built as follows. The processing time of each job on machine M_1 is equal to the minimum processing time of jobs on machine M_1. The processing times of jobs on machine M_2 are given by the processing time of jobs on machine M_2 in SPT order. The due dates of the jobs are given by the due dates in EDD order. The jobs are sequenced from 1 to n and the value of the objective function for this instance is denoted by LB1.

Instance $I2$ is built as follows. The processing times of jobs on machine M_1 are given by the processing time of jobs on machine M_1 in SPT order. The processing time of each job on machine M_2 is equal to the minimum processing time of jobs on machine M_2. The due dates of the jobs are given by the due dates in EDD order. The jobs are sequenced from 1 to n and the value of the objective function for this instance is denoted by LB2.

3.2.2.2 Improved lower bound

The idea of these lower bounds hybridization is to use the MILP model and to relax the integrity of a subset of variables corresponding to some jobs [Ta et al., 2013a]. By doing this, some jobs are completely scheduled (those for which the variables are binary) and some jobs are scheduled preemptively (those for which the variables are continuous). In Alg. 4, we denote by $\mathcal{C}$ the set of variables $x_{j,k}$ that are continuous (in $[0, 1]$), by $\mathcal{B}$ the set of variables that remain binary and $\mathcal{J}$ is a set of jobs. We assume that $|\mathcal{B}| = H_\ell$ is fixed. We denote by $MILP(\mathcal{C}, \mathcal{B})$ the resolution of the MILP model with the variables of $\mathcal{C}$ relaxed and the variables of $\mathcal{B}$ binary.

The particular case where $\mathcal{B} = \emptyset$ ($H_\ell = 0$) corresponds to the classical linear relaxation. This lower bound is denoted by $LB3$.

Algorithm 4 Hybrid lower bound algorithm

while stopping criterion not met **do**
  Choose R randomly in $[1, n - H_\ell + 1]$
  $\mathcal{B} = \{R, R+1, ..., R+H_\ell - 1\}$
  $LB = MILP(\mathcal{J} \setminus \mathcal{B}, \mathcal{B})$
  $LBmax = \max\{LB, LBmax\}$
end while
$LB4 = LBmax$
return $LB4$

In our implementation, the stopping criterion is a time limit denotes by $TimeLimLB$.

3.2.3 Dominance conditions for $m = 2$ machines

A dominance condition helps us to prune nodes. We introduce the notations:

- $\overline{T}(S)$: total tardiness of sequence S
- $S = \sigma_1 i \sigma_2 j \sigma_3$ denotes that we consider in S a subsequence σ_1, then job i, then a subsequence σ_2, then job j and finally a subsequence σ_3
- $S' = \sigma_1 j \sigma_2 i \sigma_3$ denotes exactly the same sequence as S except for the interchange of i and j.
- $C(\alpha \mid \beta)$ denotes the completion time of α if it is sequenced after β, with α and β subsequences of jobs.

In [Pan and Fan, 1997] proposed dominance conditions in another *back to front* branch and bound algorithm. The DC-PF1 and DC-PF2 conditions are general and can be used in an B&B algorithm (see Chapter 2).

Pan et al [Pan et al., 2002] developed a B&B algorithm where the sequence is built from *front to back*, with new dominance conditions and a new lower bound (see Chapter 2).

3.3 Computational experiments

3.3.1 Data generation

We have tested the algorithms on data sets which have been randomly generated for case $m = 2$ (notice that there is no benchmark instance for the two-machine flowshop problem). The processing times $p_{i,j}$ have been generated in [1,100], the due dates d_j have been generated in $[50, 50n]$. The number of jobs n belongs to $\{10, 14, 20\}$ for upper bounds and $n \in \{20, 30, 50, 70, 100, 150, 200, 250, 300, 350, 400, 500\}$ for lower bounds. 30 instances have been generated per value of n.

The time limit of CPLEX and $TimeLimLB$ have been fixed to 600 seconds, size $H_\ell = 6$.

3.3.2 Comparison of the exact methods

Three upper bounds are compared in terms of quality: the branch-and-bound that doesn't use the dominance conditions (B&B), the branch-and-bound that uses the dominance condition (denoted by ($B\&B_{DC}$) and CPLEX (uses model in Section. 3.1).

Table 3.1: Comparison of $B\&B$ algorithms and CPLEX for $n = 10$

	$B\&B$			$B\&B_{DC}$			CPLEX	
Ins	$\sum T_j$	Cpu(s)	node(s)	$\sum T_j$	Cpu(s)	node(s)	$\sum T_j$	Cpu(s)
1	692	0,07	3701	692	0,05	2498	692	0,24
2	515	0,02	923	515	0,02	877	515	0,90
3	1008	0,08	3706	1008	0,06	3083	1008	0,50
4	907	0,09	4538	907	0,07	3704	907	1,07
5	1118	0,00	112	1118	0,00	87	1118	0,20
6	112	0,03	1675	112	0,01	825	112	0,17
7	1077	0,04	2052	1077	0,04	1869	1077	0,60
8	974	0,02	831	974	0,01	635	974	0,4
9	636	0,05	2498	636	0,02	1146	636	0,14
10	343	0,12	5966	343	0,08	4467	343	0,20
11	826	0,04	1955	826	0,03	1529	826	0,12
12	796	0,03	1469	796	0,03	1287	796	0,69
13	204	0,02	1228	204	0,01	754	204	0,17
14	371	0,12	6195	371	0,05	2596	371	0,39
15	496	0,02	1211	496	0,02	1034	496	0,16
16	121	0,03	1052	121	0,02	697	121	0,5
17	604	0,06	3072	604	0,05	2420	604	0,53
18	254	0,03	1468	254	0,02	932	254	0,07
19	1589	0,02	1164	1589	0,02	1104	1589	0,35
20	0	0,01	1	0	0,01	1	0	0,04
21	927	0,03	1363	927	0,02	1189	927	0,32
22	76	0,01	401	76	0,01	308	76	0,25
23	115	0,04	2017	115	0,03	1428	115	0,10
24	99	0,02	1037	99	0,02	748	99	0,08
25	291	0,11	5531	291	0,06	3147	291	0,09
26	1193	0,04	1914	1193	0,03	1843	1193	0,21
27	760	0,00	120	760	0,00	114	760	0,11
28	120	0,01	580	120	0,01	321	120	0,11
29	1377	0,03	1926	1377	0,02	1318	1377	0,07
30	754	0,08	4055	754	0,07	3406	754	0,82
Average		0,04			0,03			0,32

As expected, we can see in Table 3.1, 3.2, that the value of the total tardiness of B&B is equal to value of $B\&B_{DC}$ and to value returned by CPLEX. The computation time of $B\&B_{DC}$ is the quickest in three methods for $n = 10$. However, the computation time of CPLEX is faster than $B\&B_{DC}$ and B&B methods for $n = 14, 20$. The computation time

Table 3.2: Comparison of $B\&B$ algorithms and CPLEX for $n = 14$

	$B\&B$			$B\&B_{DC}$			CPLEX	
Ins	$\sum T_j$	Cpu(s)	node(s)	$\sum T_j$	Cpu(s)	node(s)	$\sum T_j$	Cpu(s)
1	850	13,51	422767	850	7,47	244112	850	1,56
2	857	23,30	827826	857	13,68	501374	857	4,66
3	2318	119,38	4285800	2318	64,01	2294787	2318	0,89
4	983	6,14	211325	983	4,32	151005	983	0,75
5	720	6,51	206642	720	3,48	118801	720	1,16
6	1108	61,85	2249608	1108	25,4	970641	1108	2,41
7	1713	0,29	7719	1713	0,23	6114	1713	0,76
8	929	156,54	5644618	929	81,41	3079850	929	19,30
9	1014	23,45	766716	1014	19,74	646350	1014	10,02
10	1512	9,60	289300	1512	5,12	170854	1512	1,00
11	887	12,18	371564	887	6,18	198995	887	2,47
12	956	8,59	267228	956	4,64	149350	956	2,16
13	1176	7,18	250426	1176	4,34	155860	1176	1,28
14	1027	9,42	301240	1027	4,24	139858	1027	7,38
15	866	12,67	415272	866	4,90	170943	866	0,24
16	1341	0,69	18327	1341	0,45	12173	1341	1,26
17	1319	69,95	2608974	1319	23,14	890680	1319	0,83
18	1571	2,72	75587	1571	1,88	54034	1571	4,00
19	2397	9,47	303541	2397	6,11	202129	2397	0,97
20	725	12,3	402165	725	8,73	295276	725	15,15
21	722	24,59	765251	722	4,18	143134	722	0,73
22	1190	30,77	1057627	1190	15,88	565951	1190	2,39
23	244	2,12	67076	244	1,32	46765	244	1,85
24	39	4,78	188980	39	1,42	60952	39	0,29
25	929	66,65	2553782	929	13,74	541870	929	6,85
26	781	3,38	123939	781	0,97	36977	781	0,13
27	1488	4,45	131941	1488	3,00	88801	1488	2,82
28	683	27,1	930349	683	13,44	491349	683	4,19
29	1169	2,81	82662	1169	1,11	33975	1169	1,59
30	1769	2,32	69933	1769	0,86	26635	1769	0,35
Average		24,49			11,51			3,32

of $B\&B_{DC}$ is quicker than B&B, because it uses dominance conditions for pruning nodes, so the number of explored nodes is smaller than for B&B.

In Table 3.3, the results show that CPLEX performs well for $n = 20$, but the its computation time increases significantly (average computation time equals 124,12 seconds, 0,32 seconds for $n = 10$ and 3,32 seconds for $n = 14$). 90% of $B\&B_{DC}$ can not give a solution in less than 600 seconds.

Table 3.3: Comparison of $B\&B_{DC}$ algorithm and CPLEX for $n = 20$

	$B\&B_{DC}$			CPLEX	
Ins	$\sum T_j$	Cpu(s)	node(s)	$\sum T_j$	Cpu(s)
1	-	>600	-	1190	600,71
2	525	456.68	14239117	525	2,70
3	-	>600	-	218	0,95
4	-	>600	-	2378	37,54
5	-	>600	-	1572	257,89
6	-	>600	-	979	1,65
7	-	>600	-	1594	26,16
8	-	>600	-	2817	6,21
9	-	>600	-	1969	16,36
10	-	>600	-	2632	4,61
11	33	0.11	2392	33	600,14
12	-	>600	-	1014	14,95
13	-	>600	-	1757	135,96
14	3536	361.23	5568560	3536	6,24
15	-	>600	-	2462	29,67
16	-	>600	-	2406	144,08
17	-	>600	-	325	34,03
18	-	>600	-	2275	109,99
19	-	>600	-	1297	3,67
20	-	>600	-	2005	600,82
21	-	>600	-	2274	15,44
22	-	>600	-	1116	14,08
23	-	>600	-	1059	1,61
24	-	>600	-	3887	94,53
25	-	>600	-	1714	351,29
26	-	>600	-	291	116,05
27	-	>600	-	3139	18,46
28	-	>600	-	729	0,79
29	-	>600	-	1506	122,85
30	-	>600	-	2054	354,21
Average					124,12

3.3.3 Comparison of lower bounds

Four lower bounds are also compared: the two classical lower bounds (LB1 and LB2), the linear relaxation LB3 and the partial linear relaxation (also called the hybrid algorithm, denoted by LB4). The results are presented in Table 3.4.

In Table 3.4, column 'Best' always indicate the number of times the lower bound is the best among the four lower bounds and column 'Cpu(s)' is the computation time in seconds.

It is clear that LB1 and LB2 do not outperformed by the other bounds. However, these

Table 3.4: Comparison of lower bounds

	LB1		LB2		LB3		LB4	
n	Best	Cpu(s)	Best	Cpu(s)	Best	Cpu(s)	Best	Cpu(s)
20	0	0	0	0	0	0,04	30	2,74
50	0	0	0	0	0	0,26	30	99,63
100	0	0	0	0	1	1,68	30	602,98
150	0	0	0	0	1	5,3	30	686,56
200	0	0	0	0	2	12,92	29	675,9
250	2	0	2	0	3	24,85	29	617,97
300	0	0	0	0	18	48,13	12	604,2
350	1	0	1	0	15	76,42	17	650,59
400	0	0	3	0	15	140,01	14	607,45
500	3	0,01	3	0	25	203,67	9	635,85

bounds can be computed very quickly, which is a very interesting advantage in a search tree procedure. For more than 300 jobs, one can see the performance of LB4 decreasing. The problem comes from the size of the problem to solve. With $n = 300$ jobs, the number of binary variables is equal to 1800 ($H_\ell \times n$, with $H_\ell = 6$) and CPLEX is not always able to return a feasible solution within the time limit. Anyway, for small size instances, this method outperforms the classical linear relaxation, and such a method should be tested in a branch-and-bound algorithm, where the size of the problem to solve is generally – at least for the $F2||\sum T_j$ problem – smaller than 50 jobs.

3.4 Conclusion of chapter 3

In this chapter, we proposed exact methods: MILP and branch-and-bound for m-machine permutation flow shop scheduling problem to minimize total tardiness. We tested the methods with small to medium instances for two machines and number of jobs $n \in \{10, 14, 20\}$ random. The results show that the computation time of CPLEX is quicker than our B&B methods for $n \geq 14$. The B&B using the dominance conditions performs well and its computation time is less than the B&B method that doesn't use the dominance conditions. A new lower bound algorithm and classical lower bound are proposed and evaluated. The new lower bound performs well for small instances.

Chapter 4

Heuristic algorithms

A heuristic algorithm is a method which finds "good" solutions within an acceptable computation time without being able to ensure the optimality of the solution. A practical approach is to use heuristic methods to obtain quickly near-optimal solutions. In this chapter, we present heuristics and metaheuristic algorithms that we have developed and the computational results that we have obtained.

4.1 Basic heuristics

In this section we propose two basic heuristic algorithms, EDD and NEH, that run in $O(n \log n)$ time.

4.1.1 EDD algorithm

EDD (*Earliest Due Date*, see also Section 2.3.1 page 46): The job with the smallest due date is selected first, $min\{d_j\}$, where d_j denotes the due date of the job j. The algorithm is described in **Algo. 5**.

Algorithm 5 EDD algorithm

1: **Input**: S = a set of jobs,
2: **Sorted**: the jobs by non decreasing order of d_j,
3: **Output**: A set of jobs sorted in non decreasing order of d_j

4.1.2 NEH algorithm

In [Nawaz et al., 1983], the authors develop NEH heuristic for m-machine flow shop scheduling problem with makespan minimization. We propose and apply the method for minimizing the total tardiness for the m-machine permutation flow shop scheduling problem. NEH algorithm is described in details for the problem below (see **Algo. 6**).

We propose another version of NEH heuristic for the problem: NEH_{EDD} algorithm is described in **Algo. 7**.

Algorithm 6 NEH algorithm

1: Input: S = jobs sorted in the decreasing order of P_j,
2: where $P_j = \sum_{i=1}^{m} p_{i,j}$, $\forall j = 1, \ldots, n$
3: Consider the partial sequence with minimum total tardiness and minimum makespan in case of ties among $\{(S_{[1]}, S_{[2]}), (S_{[2]}, S_{[1]})\}$
4: **for** $k = 3$ **to** n **do**
5: Test the insertion of $S_{[k]}$ at any possible position in S' from 1 to $k + 1$
6: Keep the best insertion, i.e. the insertion with minimum $\sum T_j$, and the insertion with minimum makespan in case of ties.
7: **end for**

Algorithm 7 NEH_{EDD} algorithm

1: Input: S = jobs sorted in EDD order,
2: Consider the partial sequence with minimum $\sum T_j$ and minimum makespan in case of ties among $\{(S_{[1]}, S_{[2]}), (S_{[2]}, S_{[1]})\}$
3: **for** $k = 3$ **to** n **do**
4: Test the insertion of $S_{[k]}$ at any possible position in S' from 1 to $k + 1$
5: Keep the best insertion, i.e. the insertion with minimum total tardiness, and the insertion with minimum makespan in case of ties.
6: **end for**

4.2 Truncated search tree methods

4.2.1 Beam search algorithm

Principles

Beam search (BS) algorithms are truncated branch-and-bound algorithms which were first introduced in the context of scheduling by Ow and Morton [Ow and Morton, 1988]. The exploration of the tree is a breadth first search with aggressive pruning of the branches at each level, according to an evaluation function. In a BS algorithm, only the most promising w nodes (instead of all nodes) at each level of the search tree are retained for further branching. The parameter w is called the *beam width*. The beam width controls the extensiveness of the search. The greater the beam width, the fewer nodes are pruned, which increases the cost of computational effort. When the beam width is an infinite number (or a sufficiently high number), no nodes are pruned and the beam search method becomes the branch-and-bound method with breadth first exploration. If $w = 1$, the beam search extends only one solution path and it becomes a greedy method. There is no backtrack in the beam search and no possible recover from wrong decisions, therefore there is no guarantee that the method finds an optimal solution, because the best solution may have been potentially pruned.

Algorithm

The main steps of the *beam search* algorithm are described in **Algo. 8** [Ta et al., 2013b].

Algorithm 8 Beam search algorithm

Initialization:
$C = \emptyset$
$B = \{\text{root}\}$
Repeat:
for each node S in B **do**
  Generate the corresponding children of S: $\tau_1, \tau_2, \ldots$
  Calculate $LB(\tau_j)$ and $UB(\tau_j)$.
  $V(\tau_j) = \alpha UB(\tau_j) + (1 - \alpha)LB(\tau_j)$
  Select the w best child nodes and add them to C.
end for
$B = \emptyset$
Selection:
Select w best nodes in C and add them to B.
$C = \emptyset$.
Until ((the nodes in B are leaves (they correspond to a complete sequence) or (CPU $>$ $TimeLimitBS$))
Select the best node.

A counter-intuitive example of the Beam Search algorithm with $w = 1$ and $w = 2$

BS algorithm is illustrated in **Fig. 4.1** and **Fig. 4.2** for a scheduling problem with six jobs and two machines. The objective of this section is to illustrate the fact that the Beam Search algorithm with a beam width of 2 can be worse than the same method with a beam width of 1, whatever the computation time is. In other words, enlarging the size of the beam is not a guaranty of quality improvement. V denotes the value of the evaluation function of each node. At each level, only w nodes are kept. The data are given in Table 4.1.

Table 4.1: Data of six jobs and two machines

j	0	1	2	3	4	5
$p_{1,j}$	39	69	2	80	19	25
$p_{2,j}$	73	68	10	16	3	85
d_j	239	27	64	15	235	233

In **Fig. 4.1** ($w = 1$), the final solution is $\{2, 5, 3, 0, 4, 1\}$ and $\sum T_j = 388$. In **Fig. 4.2** ($w = 2$), the final solution is $\{2, 5, 1, 3, 4, 0\}$ and $\sum T_j = 402$. We can see that if $w = 2$ the children nodes of node $\sigma = (2, 5, 3)$ are not kept because two children nodes of node $\sigma = (2, 5, 1)$ are more promising. But at the end, these nodes lead to a solution with total tardiness of 402, whereas node $(2, 5, 3)$ leads to a better solution.

So, increasing the size of the beam width increases the computation time and does not guaranty that the solution that is returned will be better.

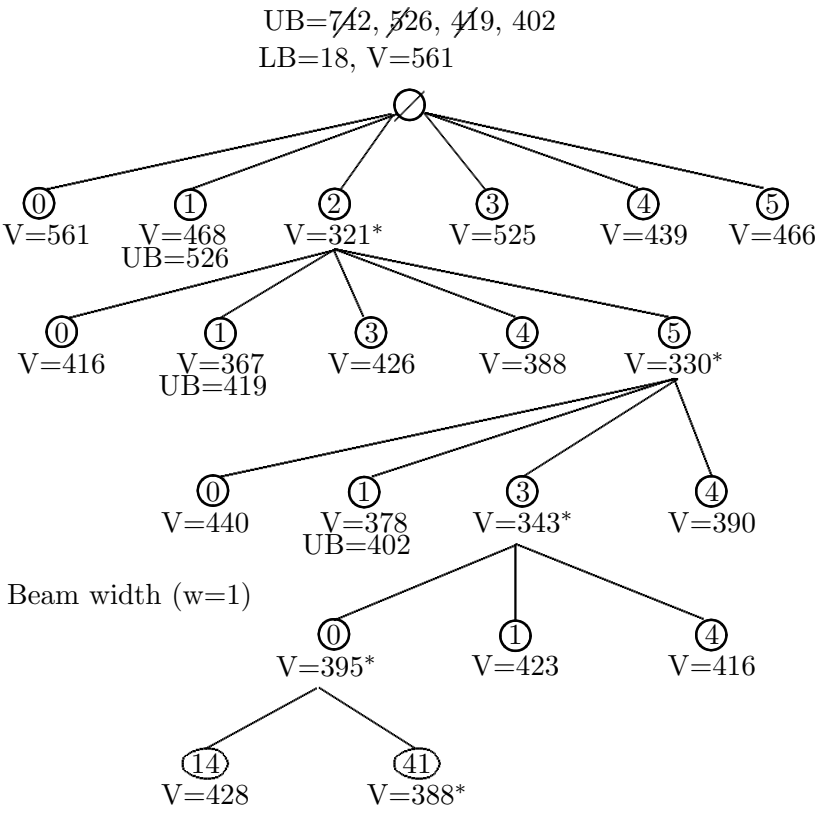


Figure 4.1: Illustration of BS with beam width ($w = 1$)

4.2.2 Recovering beam search algorithm

Principles

The *Recovering Beam Search* (RBS) algorithm is an hybrid heuristic method for combinatorial [Della Croce and T'kindt, 2002], [Della Croce et al., 2004], [Ta et al., 2013c] optimization, which is an improvement of the beam search algorithm. As discussed before, the BS cannot recover from decisions: if a branch leading to the optimal solution in the search tree is pruned in the nodes evaluation process, there is no way to reach afterwards that solution. An improvement of RBS is the *recovering step*, which aims at recovering from previous wrong decisions. This step is invoked to each of the w best child nodes generated. For a given node, the recovering phase, by means of interchange operators applied to the current partial schedule S, checks whether the current solution is dominated by another partial solution S', sharing the same search tree level. If a better partial solution S' is found, it becomes the new current partial solution.

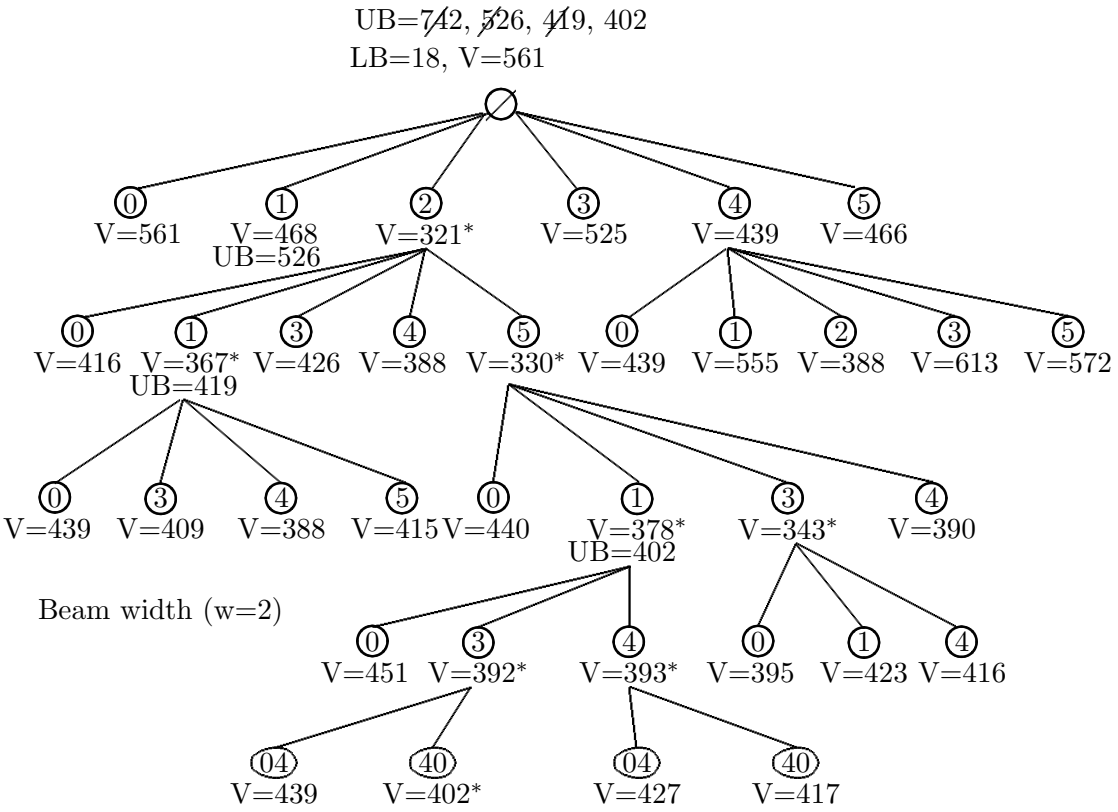


Figure 4.2: Illustration of BS with beam width ($w = 2$)

Neighborhood operators

Two perturbations mechanisms have been implemented [Della Croce et al., 2004], [Wu et al., 2007]. We denote by S the current sequence. The neighborhood operators applied to $S = S_1/S_{[i]}/S_2/S_{[j]}/S_3$ with S_1, S_2 and S_3 three subsequences of S and $S_{[i]}$ and $S_{[j]}$ the jobs in positions i and j in S ($i \neq j$) are the following:

- SWAP: A neighbor of S is created by interchanging the jobs in position i and j, leading to sequence $S' = S_1/S_{[j]}/S_2/S_{[i]}/S_3$.

- EBSR (*Extraction and Backward Shifted Re-insertion*): A neighbor of S is created by extracting $S_{[j]}$ and re-inserting $S_{[j]}$ backward just before $S_{[i]}$, leading to sequence $S' = S_1/S_{[j]}/S_{[i]}/S_2/S_3$.

- EFSR (*Extraction and Forward Shifted Re-insertion*): A neighbor of S is created by extracting $S_{[i]}$ and re-inserting it forward immediately after $S_{[j]}$, leading to a sequence $S' = S_1/S_2/S_{[j]}/S_{[i]}/S_3$.

The three neighborhood operators are illustrated in **Fig. 4.3**.

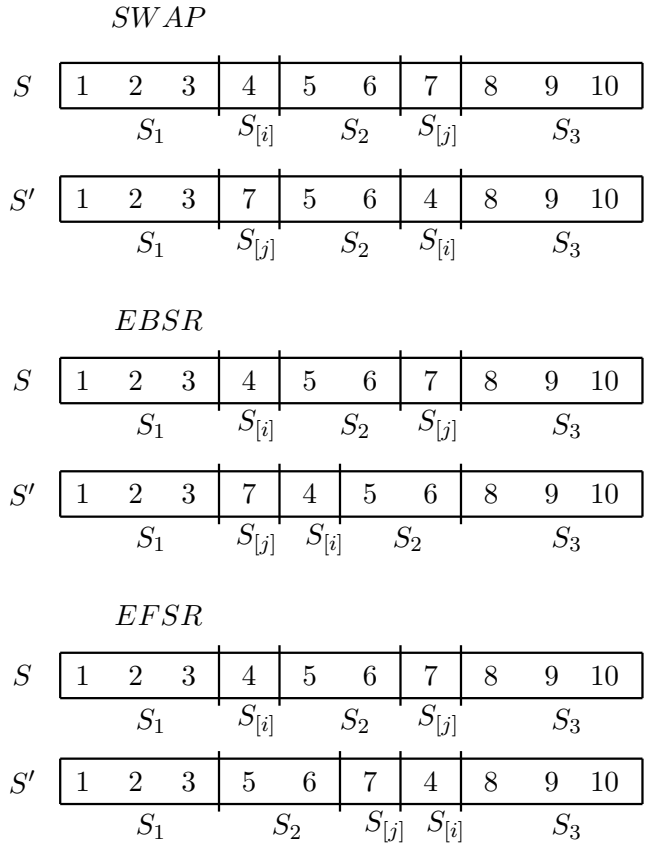


Figure 4.3: Illustration of three neighborhood operators

Algorithm

The main steps of the RBS algorithm are described in **Algo. 9**.

4.2.3 Evaluation of nodes for BS and RBS algorithms

Each node of the search tree is evaluated by an evaluation function V, based on a weighted sum of a lower bound (LB) and an upper bound (UB). The function V is defined by

$$V = \alpha UB + (1 - \alpha)LB$$

where $0 \leq \alpha \leq 1$ is an experimentally defined parameter.

The UB at each node is calculated by sequencing the unscheduled jobs in EDD order after the scheduled jobs. The LB used in the evaluation function is the same method as the one described in **Section 3.2**.

Algorithm 9 Recovering beam search algorithm

Initialization:
$C = \emptyset$
$B = \{\text{root}\}$
Repeat:
for each node S in B **do**
  Generate the corresponding children of S: $\tau_1, \tau_2, \ldots$
  Calculate $LB(\tau_j)$ and $UB(\tau_j)$.
  $V(\tau_j) = \alpha UB(\tau_j) + (1 - \alpha)LB(\tau_j)$
  Select the w best child nodes and add them to C.
end for
$B = \emptyset$
Selection:
Select the w best nodes in C and add them to B.
Set $C = \emptyset$.
Recovering:
for each node S in B **do**
  Apply the neighborhood operators on S and search for a partial solution S' that dominates S.
  if S' is found **then**
    Replace S by S' in B
  end if
end for
Until ((the nodes in B are leaves (they correspond to a complete sequence) or (CPU $>$ *TimeLimitRBS*))
Select the best node.

4.3 Metaheuristic algorithms

In this section we present two metaheuristic developed for solving our problem. The first is a genetic algorithm, the second is a Tabu search. The general principle of these methods has been described in **Section 1.2.2**.

4.3.1 Genetic algorithm

We describe in this section our implementation [Ta et al., 2013d], [Ta et al., 2014b] of the crossover and of the mutation operators.

Genetic operators

- **Coding**: The crucial step in designing a Genetic algorithm is to define an encoding, i.e. a way to represent a solution. In the case of the m-machine permutation flow shop scheduling problem with n jobs indexed from 1 to n, an individual is represented by a *permutation*.

- **Initial population**: The initial population P_0 contains $PopSize$ individuals. The individual is obtained by sequencing the jobs according to a given rule. The other individuals are randomly generated.

  In GA_{EDD}, one individual is given by EDD algorithm, and the others are randomly generated. In GA_{NEH}, one individual is given by NEH algorithm described in Algo. 6, and the others are randomly generated. In GA_{EN}, one individual is given by the best sequence among EDD and NEH and the others are randomly generated. And $GA_{E\&N}$, two individuals are given by EDD and NEH, the others are randomly generated.

- **Fitness**: The *fitness* of an individual S is the value of the objective function $\sum T_j(S)$ of the corresponding sequence.

- **Crossover** Several *crossover operators* are used, the *one-point crossover* (X1), the *linear order crossover* (LOX), the *Similar Job Order Crossover* or (SJOX), the *Similar Block Order Crossover* or (SBOX) and the *Similar Block 2-Point Order Crossover* or (SB2OX). The operators are described the follow:

  – X1 [Della Croce et al., 2004]: One crossover point is randomly generated in $\{1..n\}$. Let $A = A1//A2$ and $B = B1//B2$ be the two parents. Two offsprings are calculated. Offspring 1 denoted by $O1$ contains the jobs of $A1$ in the order of A and the jobs of $A2$ in the order of B. Offspring 2 denoted by $O2$ contains the jobs of $B1$ in the order of B and the jobs of $B2$ in the order of A. The X1 operator is illustrated in **Fig. 4.4**.

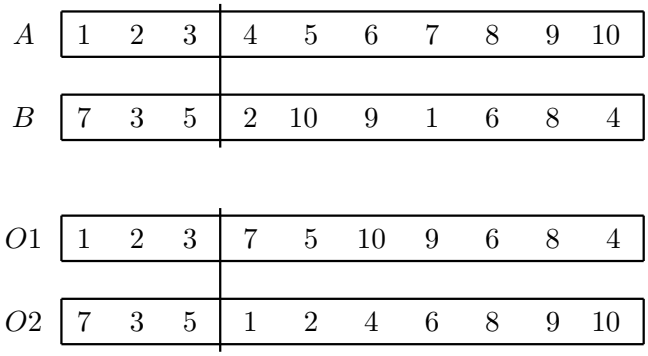


Figure 4.4: Illustration of $X1$ crossover operator

  – LOX [Della Croce et al., 2004]: Two different crossover points are randomly generated $\{1..n\}$. Let $A = A1//A2//A3$ and $B = B1//B2//B3$ be the two parents. Two offsprings are calculated. Offspring 1 denoted by $O1$ contains in the middle the jobs of $A2$ in the order of A. The jobs of $A1 \cup A3$ in the order of B fill the first and the last part of A. Offspring 2 denoted by $O2$ contains in the middle the jobs of $B2$ in the order of B. The jobs of $B1 \cup B3$ in the order of A fill the first and the last part of B. The LOX operator is illustrated in Fig. 4.5.

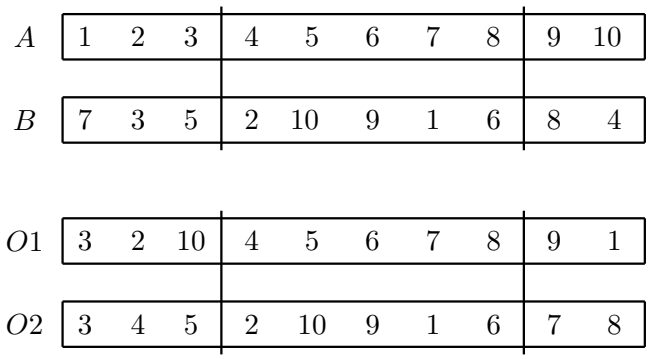
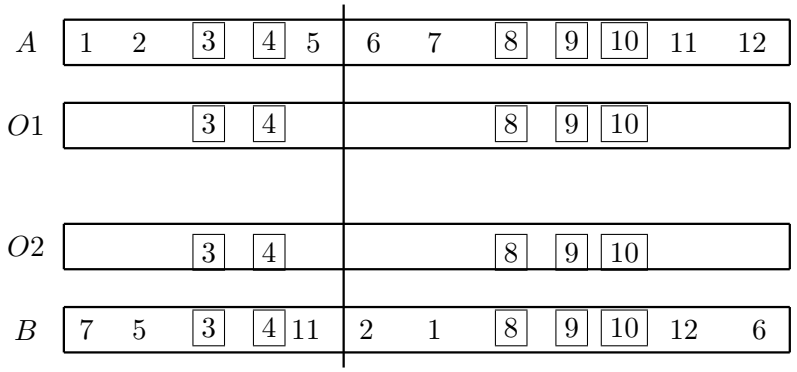


Figure 4.5: Illustration of LOX crossover operator

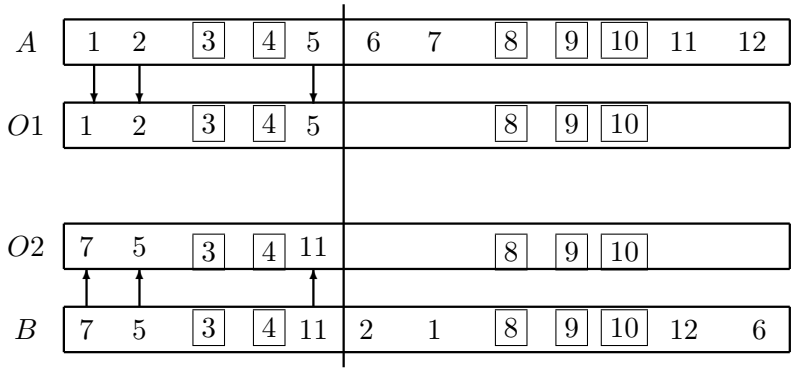
- SJOX [Ruiz et al., 2006]: Let A and B be two parents which are chosen randomly in the initial population. Two offsprings $O1$ and $O2$ are created as follows: First, identical jobs at the same positions on both parents are copied in both offspring (**Fig. 4.6(a)**). Next, one point crossover is randomly chosen and the missing jobs in the offspring $O1$ and $O2$ are copied into the parents A and B respectively (**Fig. 4.6(b)**). Then the jobs at the right side of the cut point are filled according to the job sequence of parents B and A to form the offspring $O1$ and $O2$ respectively (**Fig. 4.6(c)**).
- SBOX [Ruiz et al., 2006]: Let A and B be two parents which are chosen random in initial population. Two offsprings $O1$ and $O2$ are created as follows: First, blocks of at least two consecutive identical jobs at the same positions on both parents are copied in the two offspring (**Fig. 4.7(a)**). Second, one point crossover is randomly chosen and the missing jobs in the offspring $O1$ and $O2$ are copied from the parents A and B respectively (**Fig. 4.7(b)**). Then the jobs by the right side of the cut point are filled according to the job sequence of parents B and A to form the offspring $O1$ and $O2$ respectively (**Fig. 4.7(c)**).
- SB2OX [Ruiz et al., 2006]: The main difference with the SBOX crossover is that in the second step, two-point crossover is randomly chosen and the missing jobs in the offspring $O1$ and $O2$ between these two points are copied from the parents A and B respectively. Then the offsprings are filled by copying the jobs of parents B and A to form the offspring $O1$ and $O2$ respectively.

In our implementation of the Genetic Algorithm, the crossover operator is chosen randomly, with equal probability.

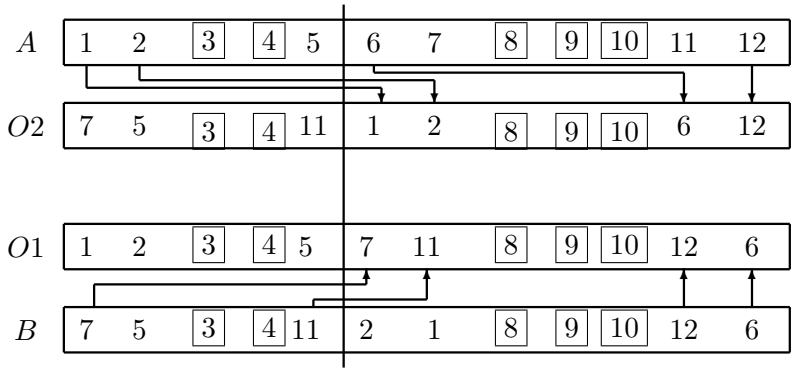
- **Mutation**: The *mutation operators* have been used:
  - SWAP (see **Section 4.2.2**)
  - EBSR (see **Section 4.2.2**)
  - EFSR (see **Section 4.2.2**) *These three mutation operators (SWAP, EBSR, EFSR) are illustrated in Fig. 4.3.*
  - Inversion: given a sequence $S = S_1/S_{[i]}/S_2/S_{[j]}/S_3$, with S_1, S_2 and S_3 three subsequences and $S_{[i]}$ and $S_{[j]}$ the jobs in positions i and j are randomly chosen



(a) The identical jobs in both parents are copied over to the offspring



(b) The missing jobs in the offsprings are copied to the parents respectively

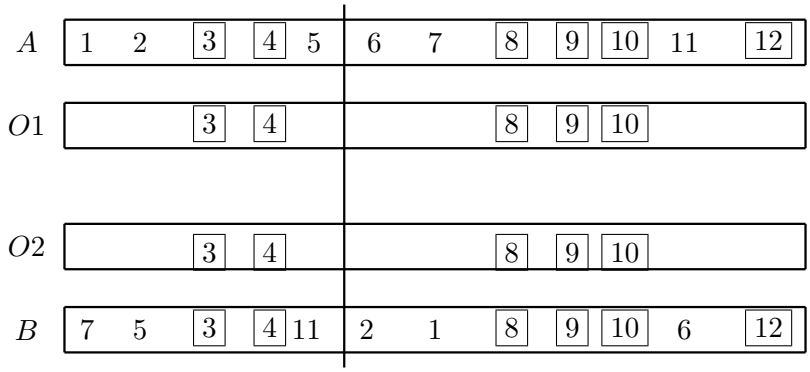


(c) The missing jobs of offsprings are filled in the relative order of the other parent

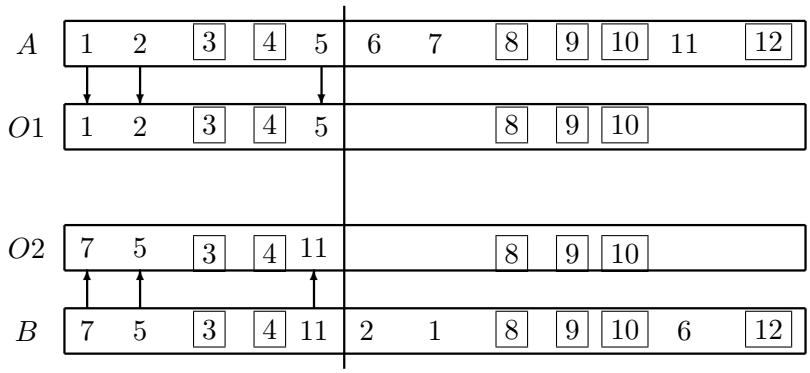
Figure 4.6: Illustration of SJOX crossover operator

$(i < j)$ in S. A neighbor of S is created by inserting $S_{[j]}/\overline{S_2}/S_{[i]}$ between S_1 and S_3, where $\overline{S_2}$ is the inverse of sequence S_2 (see **Fig. 4.8**).

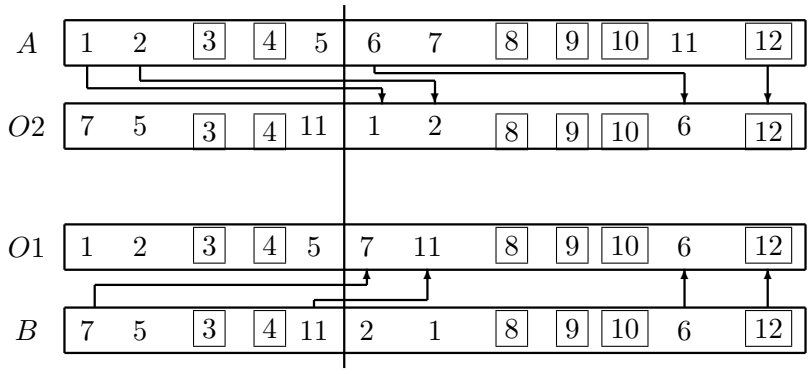
- **Selection and generational scheme**: At iteration k, two parents are randomly selected in population P_{k-1}. The two crossover operators are applied on the two



(a) The identical jobs in both parents are copied over to the offspring (only blocks of at least two consecutive jobs)



(b) The missing jobs in the offsprings are copied to the parents respectively



(c) The missing jobs of offsprings are filled in the relative order of the other parent

Figure 4.7: Illustration of SBOX crossover operator

parents, generating four offsprings, inserted into population set C_k. The process is repeated until $CrossSize$ offsprings have been generated.

The mutation operator is applied on randomly selected individuals of population

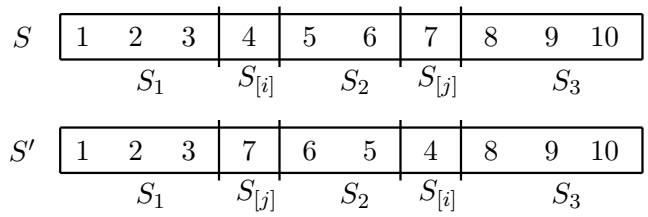


Figure 4.8: Illustration of inversion mutation operator

P_{k-1}. The new individuals constitute a population M_k of size $MutSize$.

The $PopSize$ best individuals of $P_{k-1} \cup C_k \cup M_k$ constitute population P_k.

- **Stopping criterion**: The process iterates until a given time limit has been reached. This time limit is denoted by $TimeLimGA = n(m/2)t$ ms [Vallada et al., 2008], where $t = 90$.

A lot of parameters and operators have been tested for the genetic algorithms, it concerns:

- the generation on the initial population,
- the crossover operators,
- the mutation operators.

The tests do not lead to the best results that have been obtained and therefor, they are not presented here, but given in **Appendix B**.

4.3.2 Tabu search

We describe in this section our implementation of the TS algorithm [Ta et al., 2013a].

Initial solution

The proposed Tabu search algorithm starts from an initial solution. This initial solution is classically generated by using simple heuristic methods, such as EDD, SPT, NEH, etc. In the TS that we propose, different initial solutions are considered. If the initial solution is given by EDD rule, the method is denoted by TS_{EDD}. If the initial sequence is given by applying an adaptation of NEH algorithm (see **Section 4.1.2 Algo. 6**), the method is denoted by TS_{NEH}. Finally, if the initial sequence is given by the best solution beetween EDD and NEH, the method is denoted by TS_{EN}.

Neighborhood definition

We denote by S the current sequence. We denote by $N(S)$ the set of all neighbors of S which can be created by SWAP, EBSR, EFSR, Inversion operators (see **Section 4.3.1**).

Moves and selecting the best neighbor

The objective function is to minimize the total tardiness. The best neighbor in the candidate list is the non-tabu sequence with the smallest total tardiness. The move strategy which is applied to the list is the first-in-first-out (FIFO) strategy. Old attributes are deleted as new attributes are inserted.

Tabu list

The size of the tabu list is a very important parameter, which can be either fixed or variable. In [Glover, 1989, Glover, 1990], the author provided some general methods of tabu list implementations. In [Nowicki and Smutnicki, 1996] the authors generate a tabu list by storing attributes of the visited permutations, defined by certain pair of adjacent jobs. Our tabu list contains pairs of positions (i, j), corresponding to the neighborhood definition and the size of the tabu list is fixed.

Stopping condition

The algorithm is stopped when the time limit has been reached. This time limit is denoted by $TimeLimitTS = n(m/2)t$ ms, with $t = 90$.

Detailed algorithm

The detailed TS algorithm is given in **Algo. 10**. FlagSwap, FlagEBSR, FlagEFSR and FlagInversion allow to make a selection of the neighbors. LimitSwap, LimitEBSR, LimitEFSR and LimitInversion allow to limit the size of the neighborhood. Del(T) deletes the upper element of the Tabu list and Add($T, (k, j)$) adds element (k, j) to the Tabu list.

The other parameters that have been implemented and tested for the Tabu Search are reported in **Appendix C**.

4.4 Computational experiments

The algorithms have been tested on a PC Intel $core^{TM}i5$ CPU 2.4GHz. 108 benchmark instances proposed in [Vallada et al., 2008] have been used for the evaluation. Nine instances of these benchmark instances are used for each combination of n and m, with $n \in \{50, 150, 250, 350\}$ and $m \in \{10, 30, 50\}$. In these instances, the processing times are uniformly distributed between 1 and 99. The due dates are generated with an uniform distribution between $P(1-\tau-\rho/2)$ and $P(1-\tau+\rho/2)$ following the method of Potts and Van Wassenhove [Potts and Van Wassenhove, 1982] with P a lower bound of the makespan and τ and ρ two parameters called *tardiness factor* and *due date range*, which take the following values: $\tau \in \{0.2, 0.4, 0.6\}$, $\rho \in \{0.2, 0.6, 1\}$. The first instance (among five) of [Vallada et al., 2008] for each tuple (n, m, τ, ρ) has been used for the tests, which gives the 108 instances.

Algorithm 10 Tabu search algorithm

Initialization
S_0 = initial solution, S = current solution
$S' = S_0$ // best solution of $N(S)$
$S^* = S_0$ // best solution of $N(S)$ and *non-tabu*
$f^* = f(S_0)$ // f^* value of S^* and $f(S_0)$ value of S_0
$T = \emptyset$ // T is the tabu list
while (CPU $\leq$ *TimeLimitTS*) **do**
  $f(S') = \infty$,
  for $k = 0$ **to** $n - 1$ **do**
    for $j = k + 1$ **to** n **do**
      if (FlagSwap = 1) and ($j - k \leq$LimitSwap) **then**
        $S = S'$, $f(S) = f(S')$, SWAP$(S, (k, j))$,
        if $((k, j) \notin T)$ **then**
          Calculate$(f(S))$,
          if $(f(S) < f(S'))$ **then** $S' = S$, $f(S') = f(S)$, move $= (k, j)$, **end if**
        end if
      end if
      if $(j \neq k + 1)$ and (FlagEBSR = 1) and ($j - k \leq$LimitEBSR) **then**
        $S = S'$, $f(S) = f(S')$, EBSR$(S, (k, j))$,
        if $((k, j) \notin T)$ **then**
          Calculate$(f(S))$,
          if $(f(S) < f(S'))$ **then** $S' = S$, $f(S') = f(S)$, move $= (k, j)$, **end if**
        end if
      end if
      if $(j \neq k + 1)$ and (FlagEFSR = 1) and ($j - k \leq$LimitEFSR) **then**
        $S = S'$, $f(S) = f(S')$, EFSR$(S, (k, j))$,
        if $((k, j) \notin T)$ **then**
          Calculate$(f(S))$,
          if $(f(S) < f(S'))$ **then** $S' = S$, $f(S') = f(S)$, move $= (k, j)$, **end if**
        end if
      end if
      if $(j \neq k + 1)$ and (FlagInversion = 1) and ($j - k \leq$LimitInversion) **then**
        $S = S'$, $f(S) = f(S')$ Inversion$(S, (k, j))$,
        if $((k, j) \notin T)$ **then**
          Calculate$(f(S))$,
          if $(f(S) < f(S'))$ **then** $S' = S$, move $= (k, j)$, **end if**
        end if
      end if
    end for
  end for
  if $(f(S') < f^*)$ **then** $S^* = S'$, $f^* = f(S)$, **end if**
  if (SizeTabu $\geq$ TabuMax) **then** Del(T) **end if**
  Add$(T, (k, j))$
end while

In all tables, each line summarizes the results for 9 instances and of course, the methods may return solutions with the same quality, so the total per line of 'Best' may exceed 9.

4.4.1 Comparison of EDD and NEH sequences

In Table 4.2, column 'Best' for 'EDD' indicates the number of times the method 'EDD' outperforms the method 'NEH', column Cpu(s) indicates the average computation time of 'EDD' per nine instances, column 'Δ_{EDD}' indicates the average deviation between 'EDD' and 'NEH'. Column 'Best' for 'NEH' indicates the number of times the method 'NEH' outperforms the method 'EDD', column Cpu(s) indicates the average computation time of 'NEH' per nine instances, column 'Δ_{NEH}' indicates the average deviation between 'NEH' and the 'EDD'.

The results show that NEH outperforms EDD in most of the cases.

$$\Delta_{EDD} = \frac{EDD - \min(EDD, NEH)}{EDD}$$

$$\Delta_{NEH} = \frac{NEH - \min(NEH, EDD)}{NEH}$$

Table 4.2: Comparison of EDD and NEH algorithms

	EDD			NEH		
$n \times m$	Best	Cpu(s)	Δ_{EDD}	Best	Cpu(s)	Δ_{NEH}
50 × 10	2	0,01	32,17%	7	0,01	10,66%
50 × 30	0	0,00	41,44%	9	0,00	0,00%
50 × 50	0	0,01	31,52%	9	0,01	0,00%
150 × 10	3	0,01	22,06%	6	0,01	23,70%
150 × 30	2	0,01	34,45%	7	0,01	11,97%
150 × 50	1	0,01	30,00%	8	0,01	3,77%
250 × 10	4	0,01	12,90%	5	0,01	35,46%
250 × 30	2	0,01	23,37%	7	0,01	14,52%
250 × 50	2	0,01	26,72%	7	0,01	5,55%
350 × 10	4	0,01	6,74%	5	0,01	35,13%
350 × 30	2	0,01	23,43%	7	0,01	16,72%
350 × 50	2	0,01	21,12%	7	0,01	10,09%
	24		25,49%	84		13,96%

4.4.2 Comparison of the beam search algorithms

The time limit of BS algorithms is fixed to $TimeLimitBS = n(m/2)t/1000$ seconds.

Three *beam widths* ($w = 1, 3, 5$) of BS method have been tested and compared in terms of quality. In Table 4.3, column 'Best' for 'BS_w' indicates the number of times the method BS_w outperforms the other methods, column Cpu(s) indicates the average computation

time of BS_w per nine instances, column 'Δ_{BS_w}' ($BS_w \in \{BS_1, BS_3, BS_5\}$) indicates the average deviation between BS_w and the best method between BS_1, BS_3 and BS_5.

$$\Delta_{BS_w} = \frac{BS_w - \min(BS_1, BS_3, BS_5)}{BS_w}$$

Table 4.3: Comparison of beam search algorithms

	BS_1			BS_3			BS_5		
$n \times m$	Best	Cpu(s)	Δ_{BS_1}	Best	Cpu(s)	Δ_{BS_3}	Best	Cpu(s)	Δ_{BS_5}
50 × 10	2	0,30	19,56%	2	0,64	3,49%	8	1,65	0,04%
50 × 30	0	2,38	9,48%	3	5,16	3,35%	6	11,73	0,31%
50 × 50	1	6,03	8,72%	0	14,49	2,27%	8	31,58	0,05%
150 × 10	2	13,30	10,01%	9	36,59	0,00%	2	53,38	9,66%
150 × 30	2	103,57	1,39%	5	195,20	1,56%	2	202,82	15,05%
150 × 50	5	295,34	0,60%	4	338,27	2,76%	1	339,26	6,99%
250 × 10	7	74,68	1,90%	4	88,03	6,39%	2	88,86	20,82%
250 × 30	5	481,53	1,48%	5	301,52	3,84%	1	301,93	12,81%
250 × 50	7	567,28	0,30%	2	565,56	3,45%	0	569,91	12,16%
350 × 10	7	123,37	1,42%	3	123,11	3,39%	3	123,58	8,33%
350 × 30	6	426,43	0,47%	4	425,38	7,67%	1	427,84	14,15%
350 × 50	6	800,10	0,04%	3	798,13	10,45%	1	810,59	16,34%
	50	241,19	5,75%	44	241,06	3,96%	35	246,06	9,34%

We can see from Table 4.3 that the value of the beam width that leads to the best results is $w = 1$. The example 4.2.1 shows that beam width with $w = 1$ is better than $w = 2$. However, the average deviation between the solutions returned by this method and the best solutions is 5,97%. This value is around 3,96% for BS_3 (it is better than BS_1) and 9,34% for BS_5.

4.4.3 Comparison of the recovering beam search algorithms

The time limit of RBS algorithms is fixed to $TimeLimitRBS = n(m/2)t/1000$ seconds.

Three *recovering beam widths* ($w = 1, 3, 5$) of the RBS method have been tested and compared in terms of quality. In Table 4.4, column 'Best' for 'RBS_w' indicates the number of times the method RBS_w outperforms the other methods, column Cpu(s) indicates the average computation time of RBS_w per nine instances, column 'Δ_{RBS_w}' ($RBS_w \in \{RBS_1, RBS_3, RBS_5\}$) indicates the average deviation between RBS_w and the best method between RBS_1, RBS_3 and RBS_5.

$$\Delta_{RBS_w} = \frac{RBS_w - \min(RBS_1, RBS_3, RBS_5)}{RBS_w}$$

Each line summarizes the results for 9 instances and of course, the methods may return solutions with the same quality, so the total per line of 'Best' may exceed 9.

Table 4.4: Comparison of recovering beam search algorithms

	RBS_1			RBS_3			RBS_5		
$n \times m$	Best	Cpu(s)	Δ_{RBS_1}	Best	Cpu(s)	Δ_{RBS_3}	Best	Cpu(s)	Δ_{RBS_5}
50 × 10	2	0,46	4,43%	3	1,53	4,8%	8	2,73	0,14%
50 × 30	0	3,15	7,97%	1	11,33	4,42%	8	16,61	0,06%
50 × 50	0	7,30	3,34%	2	25,70	0,64%	7	43,14	0,15%
150 × 10	8	26,53	0,85%	3	52,93	5,53%	2	54,08	16,52%
150 × 30	9	126,34	0,00%	2	189,22	18,86%	2	193,25	19,70%
150 × 50	9	305,44	0,00%	1	338,98	15,72%	1	339,01	16,92%
250 × 10	8	80,24	0,11%	3	88,83	19,17%	2	89,38	20,48%
250 × 30	9	301,28	0,00%	4	304,04	15,75%	4	303,07	17,36%
250 × 50	8	567,14	11,11%	6	570,98	26,10%	7	503,95	17,19%
350 × 10	9	123,50	0,00%	3	124,11	15,51%	3	123,78	17,61%
350 × 30	9	427,46	0,00%	6	427,25	12,09%	6	427,34	12,33%
350 × 50	9	797,47	0,00%	5	816,58	16,47%	5	803,50	16,01%
	81	230,53	1,38%	40	245,96	11,96%	54	243,95	12,72%

We can see in Table 4.4 the value of the beam width that leads to the best results ($w = 1$). For more than 150 jobs, it is clear that RBS_1 has very good performances in comparison with RBS_w (w = 3, 5) and the computation time of RBS_1 (230,53s) is faster in comparison with RBS_3 (245,96s) and RBS_5 (243,95s). The average deviation between the solutions returned by this method ($w = 1$) and the best solutions is 1,38%. This value is around 11,96% for RBS_3 and 12,72% for RBS_5.

4.4.4 Comparison of the genetic algorithms

The time limit of the GA is fixed to $TimeLimGA = (n(m/2) \times 90)/1000$ seconds (as defined in [Vallada et al., 2008]). For the genetic algorithms, a lot of preliminary experiments have been conducted for the parameters settings. At the end, two parameters sets seem to lead to the best results, denoted case 1 and case 2.

	$case1$	$case2$
$PopSize = \|P_k\|$	150	150
$CrossSize = \|C_k\|$	200	600
$MutSize = \|M_k\|$	100	360

For the same instance, the genetic algorithm has been executed ten times and it returns quite always solutions with the same quality. The average relative deviation between ten runs is less than 3%

The several GA methods are compared in terms of quality. In Tables 4.5, column 'Best' for '$GA_1(G)$' ($G \in \{EDD, NEH, EN, E\&N\}$) indicates the number of times the method $GA_1(G)$ outperforms the other methods, column Cpu(s) indicates the average computation time of $GA_1(G)$ per nine instances, column 'Δ_G' indicates the average deviation between $GA_1(G)$ and the best method between $GA_1(EDD)$, $GA_1(NEH)$, $GA_1(EN)$ and $GA_1(E\&N)$.

Column 'Best' for '$GA_2(G)$' ($G \in \{EDD, NEH, EN, E\&N\}$) indicates the number of times the method $GA_2(G)$ outperforms the other methods, column Cpu(s) indicates the average computation time of $GA_2(G)$ per nine instances, column 'Δ_G' indicates the average deviation between $GA_2(G)$ and the best method between $GA_2(EDD)$, $GA_2(NEH)$, $GA_2(EN)$ and $GA_2(E\&N)$.

$$\Delta_G = \frac{GA_c(G) - \min(GA_c(EDD), GA_c(NEH), GA_c(EN), GA_c(E\&N))}{GA_c(G)}$$

with ($c \in \{1, 2\}$)

As we can see in Table 4.5, the genetic algorithm with the initial population given by EDD rule leads to the best results. In Table 4.5, on average for case 1, the deviation between the solutions returned by this method and the best solutions is 2,75%. This value is around 7,98% for $GA_1(NEH)$, 5,84% for $GA_1(EN)$ and 3,01% for $GA_1(E\&N)$. For case 2, the average deviation between the solutions returned by this method and the best solutions is 2,62%. This value is around 9,19% for $GA_2(NEH)$, 8,24% for $GA_2(EN)$ and 2,82% for $GA_2(E\&N)$.

Table 4.5: Comparison of genetic algorithms of case 1 and case 2

	$GA_1(EDD)$			$GA_1(NEH)$			$GA_1(EN)$			$GA_1(E\&N)$		
$n \times m$	Best	Cpu(s)	Δ_{EDD}	Best	Cpu(s)	Δ_{NEH}	Best	Cpu(s)	Δ_{EN}	Best	Cpu(s)	$\Delta_{E\&N}$
50 × 10	5	22,00	0,93%	4	22,00	2,92%	2	22,00	3,83%	4	22,00	3,60%
50 × 30	3	67,01	2,76%	3	67,00	2,87%	1	67,00	4,10%	2	67,01	4,48%
50 × 50	5	112,01	1,48%	0	112,01	2,62%	2	112,01	1,88%	2	178,11	1,65%
150 × 10	6	67,02	3,51%	4	67,02	7,45%	2	67,02	9,61%	3	67,02	4,25%
150 × 30	6	202,04	6,98%	1	202,02	9,64%	2	202,05	10,14%	3	202,05	4,32%
150 × 50	3	337,05	9,89%	1	337,03	11,97%	2	337,05	10,43%	3	337,04	2,41%
250 × 10	6	112,03	0,95%	3	112,04	5,40%	3	112,07	6,58%	6	112,06	1,63%
250 × 30	6	337,05	0,32%	2	337,05	7,17%	3	597,56	5,48%	4	337,05	3,49%
250 × 50	5	562,08	1,22%	3	562,06	5,36%	2	562,06	2,29%	2	562,11	2,71%
350 × 10	6	157,09	3,45%	2	157,03	17,25%	3	157,07	7,34%	4	157,08	2,30%
350 × 30	5	472,13	1,18%	1	472,06	16,85%	3	472,10	5,26%	5	472,12	2,76%
350 × 50	6	787,09	0,32%	1	787,11	6,31%	3	787,14	3,09%	2	787,15	2,58%
	62	269,55	2,75%	25	269,54	7,98%	28	291,26	5,84%	40	275,07	3,01%
	$GA_2(EDD)$			$GA_2(NEH)$			$GA_2(EN)$			$GA_2(E\&N)$		
50 × 10	5	22,03	0,34%	4	22,02	2,52%	4	22,03	2,52%	4	22,02	1,47%
50 × 30	4	67,11	1,75%	4	67,04	2,04%	4	67,05	2,04%	1	67,04	3,13%
50 × 50	6	112,04	0,70%	2	112,04	2,28%	2	112,05	2,28%	1	112,05	2,03%
150 × 10	8	67,12	1,22%	2	67,09	12,69%	2	67,16	7,90%	3	67,09	1,45%
150 × 30	6	202,09	5,42%	2	202,13	9,77%	2	202,14	9,04%	3	202,11	3,66%
150 × 50	5	337,15	0,58%	2	337,17	5,92%	2	337,14	2,68%	2	337,17	4,78%
250 × 10	7	112,19	1,58%	2	112,38	22,43%	3	112,32	8,80%	5	112,19	2,91%
250 × 30	7	337,18	9,95%	2	337,35	6,30%	2	337,21	16,18%	1	337,19	5,35%
250 × 50	4	562,26	3,20%	1	638,75	6,89%	1	562,36	5,71%	6	562,41	1,26%
350 × 10	7	157,21	5,30%	2	157,62	24,97%	2	157,35	17,46%	4	157,30	1,23%
350 × 30	7	472,24	0,69%	2	472,27	6,94%	1	472,45	18,23%	4	472,33	4,11%
350 × 50	7	787,39	0,67%	1	787,43	7,51%	1	787,76	6,05%	3	787,33	2,41%
	73	269,67	2,62%	26	276,11	9,19%	26	269,75	8,24%	37	269,69	2,82%

Comparison of case 1 and case 2 for the best genetic algorithm

In Table 4.6, column 'Best' for '$GA_\chi(EDD)$' indicates the number of times the method $GA_\chi(EDD)$ outperforms the other methods, column Cpu(s) indicates the average computation time of $GA_\chi(EDD)$ per nine instances, column 'Δ_χ' ($\chi \in \{1, 2\}$) indicates the average deviation between $GA_\chi(EDD)$ and the best method between $GA_1(EDD)$ and $GA_2(EDD)$.

$$\Delta_\chi(EDD) = \frac{GA_\chi(EDD) - \min(GA_1(EDD), GA_2(EDD)}{GA_\chi(EDD)}$$

We can see from Table 4.6, that the genetic algorithm ($GA_1(EDD)$) with the initial population given by EDD rule and the parameters of case 1 ($PopSize = |P_k| = 150$ individuals, $CrossSize = |C_k| = 200$ individuals, $MutSize = |M_k| = 100$ individuals) leads to the best results. The average deviation between the solutions returned by this method and the best solutions is 1,39%. This value is around 2,96% for $GA_2(EDD)$.

Algorithm $GA_1(EDD)$ has been used in the following for the comparisons with the Tabu search algorithms.

Table 4.6: Comparison of best genetic algorithms of case 1 and case 2

	$GA_1(EDD)$			$GA_2(EDD)$		
$n \times m$	Best	Cpu(s)	$\Delta_1(EDD)$	Best	Cpu(s)	$\Delta_2(EDD)$
50 × 10	6	22,00	1,12%	5	22,03	2,79%
50 × 30	3	67,01	2,52%	6	67,11	1,48%
50 × 50	2	112,01	1,30%	7	112,04	0,12%
150 × 10	6	67,02	1,92%	5	67,12	1,29%
150 × 30	4	202,04	1,14%	6	202,09	0,49%
150 × 50	2	337,05	5,24%	7	337,15	0,07%
250 × 10	6	112,03	0,38%	6	112,19	1,33%
250 × 30	6	337,05	0,74%	4	337,18	11,86%
250 × 50	6	562,08	0,58%	4	562,26	2,57%
350 × 10	9	157,09	0,00%	2	157,21	9,90%
350 × 30	6	472,13	1,30%	5	472,24	1,61%
350 × 50	6	787,09	0,42%	4	787,39	2,04%
	62	269,67	1,39%	61	269,55	2,96%

4.4.5 Comparison of the Tabu search algorithms

For the Tabu list algorithms, a lot of preliminary experiments have conducted to the following parameters settings. The time limit of TS is $TimeLimTS = (n(m/2) \times 90)/1000$ seconds.

The three best TS methods are compared in terms of quality. In Table 4.7, column 'Best' for '$TS_\ell(T)$' (with ℓ is a number element of Tabu list ($\ell = 40$), T is an initial solution($T \in \{EDD, EN, NEH\}$)) indicates the number of times the method $TS_\ell(T)$ outperforms the other methods, column Cpu(s) indicates the average computation time of

Table 4.7: Comparison of Tabu search algorithms

	$TS_{40}(EDD)$			$TS_{40}(EN)$			$TS_{40}(NEH)$		
$n \times m$	Best	Cpu(s)	Δ_{EDD}	Best	Cpu(s)	Δ_{EN}	Best	Cpu(s)	Δ_{NEH}
50 × 10	6	22,01	0,23%	4	22,00	0,72%	5	22,01	0,63%
50 × 30	7	67,02	0,15%	2	67,03	1,93%	0	67,06	1,81%
50 × 50	7	112,04	0,28%	1	112,06	1,14%	2	112,05	1,12%
150 × 10	5	67,18	1,53%	6	67,02	0,88%	2	67,22	6,26%
150 × 30	8	202,28	2,60%	1	202,62	5,74%	2	202,74	2,11%
150 × 50	8	337,78	0,01%	1	338,18	8,30%	0	337,72	10,69%
250 × 10	7	112,67	0,75%	4	112,74	1,83%	4	113,10	1,51%
250 × 30	7	338,47	0,05%	3	339,38	3,93%	4	339,10	3,97%
250 × 50	7	565,79	0,22%	3	566,30	2,15%	2	565,65	4,73%
350 × 10	9	159,00	0,00%	4	158,28	2,20%	2	159,08	13,88%
350 × 30	6	476,29	11,30%	3	478,27	15,30%	2	475,08	4,06%
350 × 50	8	793,20	0,08%	2	797,05	4,94%	2	799,20	6,33%
	85	271,14	1,43%	34	271,74	4,09%	27	271,33	4,76%

$TS_\ell(T)$ per nine instances, column 'Δ_T' indicates the average deviation between $TS_\ell(T)$ and the best method between $TS_{40}(EDD)$, $TS_{40}(EN)$ and $TS_{40}(NEH)$.

$$\Delta_T = \frac{TS_\ell(T) - \min(TS_{40}(EDD), TS_{40}(EN), TS_{40}(NEH))}{TS_\ell(T)}$$

In Table 4.7, we can also see that the Tabu search algorithm where the initial solution is given by EDD rule leads to the best results. On average, the deviation between the solutions returned by this method and the best solutions is 1,43%. These values are around 4,09% for $TS_{40}(EN)$ and 4,76% for $TS_{40}(NEH)$.

4.4.6 Comparison of the best algorithm among BS, RBS, GA and TS

Now, the four algorithms (BS, RBS, GA, TS) are compared. The results are presented in Table 4.8. Column 'Best' for '$Algo_\chi$' ($Algo_\chi \in \{BS_1, RBS_1, GA_1(EDD), TS_{40}(EDD)\}$) indicates the number of times the method $Algo_\chi$ outperforms the other methods, column Cpu(s) indicates the average computation time of $Algo_\chi$ per nine instances, column 'Δ_{Algo_χ}' indicates the average deviation between $Algo_\chi$ and the best method between BS_1, RBS_1, $GA_1(EDD)$ and $TS_{40}(EDD)$.

$$\Delta_{Algo_\chi} = \frac{Algo_\chi - \min(BS_1, RBS_1, GA_1(EDD), TS_{40}(EDD))}{Algo_\chi}$$

The best results are given by the Tabu search initialized by EDD rule. The average deviation between $TS_{40}(EDD)$ and the best solution is 1,16%, the average computation time of $TS_{40}(EDD)$ per 108 instances is 265,31 seconds. These values are 38,20% and 241,06 seconds for BS_1, 27,81% and 245,96 seconds for RBS_1, 8,43% and 269,55 seconds for $GA_1(EDD)$.

Table 4.8: Comparison of the best BS, RBS, GA and TS algorithms

	BS_1			RBS_1			$GA_1(EDD)$			$TS_{40}(EDD)$		
$n \times m$	Best	Cpu(s)	Δ_{BS_1}	Best	Cpu(s)	Δ_{RBS_1}	Best	Cpu(s)	$\Delta_{GA_1(EDD)}$	Best	Cpu(s)	$\Delta_{TS_{40}(EDD)}$
50 × 10	1	0,03	34,27%	2	1,53	10,38%	2	22,00	7,54%	9	22,01	0,00%
50 × 30	0	2,38	26,47%	0	11,33	10,00%	0	67,01	8,94%	9	67,02	0%
50 × 50	0	6,03	21,77%	0	25,70	4,53%	0	112,01	4,60%	9	112,04	0,00%
150 × 10	2	13,30	30,75%	2	52,93	18,50%	2	67,02	10,93%	9	67,18	0,00%
150 × 30	0	103,57	43,41%	1	189,22	22,92%	2	202,04	6,49%	8	202,28	0,01%
150 × 50	0	295,34	33,96%	0	338,98	23,91%	0	337,05	13,00%	9	337,78	0,00%
250 × 10	2	74,68	30,18%	2	88,83	34,87%	3	112,03	5,05%	9	112,67	0,00%
250 × 30	1	481,53	38,80%	1	304,04	32,85%	3	337,05	1,89%	8	338,47	0,13%
250 × 50	0	567,28	40,19%	0	570,98	29,07%	3	562,08	1,57%	7	567,79	0,67%
350 × 10	2	123,37	34,85%	2	124,11	37,63%	2	157,09	17,76%	9	159,35	0,00%
350 × 30	1	426,43	39,37%	1	427,25	32,98%	2	427,13	4,18%	8	476,19	11,11%
350 × 50	0	800,00	43,41%	0	816,58	35,41%	2	787,09	3,19%	8	793,20	0,38%
	9	241,06	38,20%	13	245,96	27,81%	21	269,55	8,43%	101	265,31	1,16%

4.5 Conclusion of chapter 4

In this chapter, we have proposed two greedy algorithms (NEH and EDD), a *Beam search*, a *Recovering beam search*, a *Genetic algorithm* and a *Tabu search* algorithm for solving the problem. We also proposed the neighborhood operators. The algorithms are tested and evaluated from 108 benchmark instances of [Vallada et al., 2008]. Many parameters for each method have been tested. The results obtained by the proposed algorithms show that TS is the method that performs the best. The algorithms initiated by EDD heuristic are always better than the algorithms initiated by EN or NEH. We also showed that the genetic algorithm is a good metaheuristic for the problem.

Chapter 5

Matheuristic algorithms

In this chapter, we propose several matheuristic methods, based on a resolution of a partial problem by using an MILP solver (see section 3.1).
Notice that these methods can be used with any initial solution. Several versions of these algorithms have been derived, depending on how the initial sequence is obtained. In these methods, the solver for the MILP model is called iteratively.

5.1 General framework "AXB"

We consider a sequence $S = AXB$, where A, X and B are three partial subsequences. An index denoted by R and a size window denoted by H make the separation between A, X and B: the sequence of jobs from position 1 to position $R-1$ constitute sequence A, the sequence of jobs from position $R+H$ to the end constitute sequence B. These two subsequences are supposed to be unchanged. The sequence of jobs between position R and position $R+H-1$ constitute sequence X and this sequence is re-optimized, leading to subsequence X' [Ta et al., 2014b].

The sequence $S' = AX'B$ is a new sequence, hopefully better than S, and an iteration process is performed for other values of R, up to $n-H$. When all the values for R have been tested, a neighborhood (based on swaps of jobs for example) is applied to sequence S, in order to modify and possibly improve this solution, and the procedure iterates from $R=1$, until the stopping criterion is reached. The stopping criterion is a time limit called $TimeLimMH$. This process is illustrated in **Fig. 5.1** and the general algorithm of the method is given in **Algo. 11**.

In **Algo. 11** and **Algo. 12**, $S_{[k]}$ denotes the job of S in position k. If an improvement is found at a given iteration for the jobs in positions $[R, R+H-1]$, then the positions considered for the next iteration are in the interval $[R+H, R+2H-1]$, if $R+2H-1 \leq n$ (i.e. $(R-1)+H \leq n-H$). Otherwise, the positions considered for the next iteration are in the interval $[R+1, R+H]$.

In **Algo. 12**, pairs of jobs are swapped if the difference between the positions of the two jobs does not exceed $n/2$. If a swap improves the solution, the current sequence is updated and swaps continue.

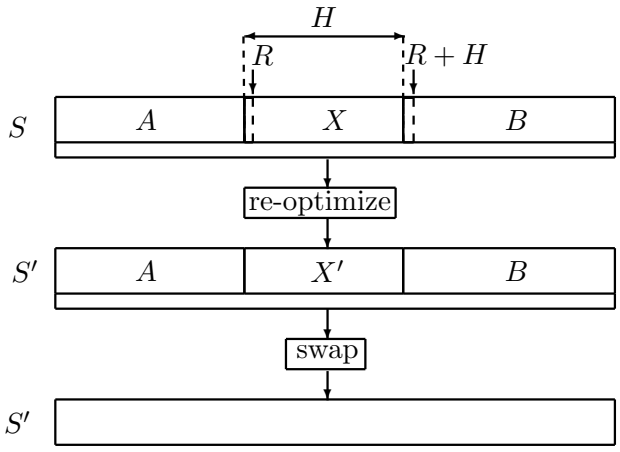


Figure 5.1: Illustration of the general framework of the matheuristic algorithm

Algorithm 11 The general Matheuristic algorithm

Input: S = initial solution
improved = true
while (CPU $\leq TimeLimMH$) and (improved = true) **do**
  improved = false; $R = 0$
  while $R \leq n - H$ **do**
    $A = (S_{[1]}, S_{[2]}, ..., S_{[R-1]})$; $X = (S_{[R]}, S_{[R+1]}, ..., S_{[R+H-1]})$
    $B = (S_{[R+H]}, S_{[R+H+1]}, ..., S_{[n]})$
    X' = re-optimization of X
    $S' = AX'B$
    if $(\sum T_j(S') < \sum T_j(S))$ **then**
      improved = true; $S = S'$
      if $(R + H \leq n - H)$ **then**
        $R = R + H - 1$
      end if
    end if
    $R = R + 1$
  end while
  S' = Swap(S)
  if $(\sum T_j(S') < \sum T_j(S))$ **then**
    improved = true; $S = S'$
  end if
end while
return(S)

Notice that this method can be used with any initial solution. Several versions of this algorithm are derived, depending on how sequence X' is obtained. The solver for the MILP model is called iteratively.

In [Della Croce et al., 2011], a similar method is proposed for the $F2||\sum C_j$ problem,

Algorithm 12 Swap (S)

Input: S = initial solution
for $i = 1$ **to** $n - 1$ **do**
  $j = i + 1$
  while $(j \leq n)$ and $(j - i \leq n/2)$ **do**
    $S' = (S_{[1]}, ..., S_{[i-1]}, S_{[j]}, S_{[i+1]}, ..., S_{[j-1]}, S_{[i]}, S_{[j+1]}, ...S_{[n]})$
    if $(\sum T_j(S') < \sum T_j(S))$ **then**
      $S = S'$
    end if
    $j = j + 1$
  end while
end for
return(S)

where the initial solution is given by a Recovering Beam Search algorithm, a different method is used for fixing the parameters of the MILP, but without any Swap procedure.

5.2 Algorithms based on the general "AXB" framework

In this section, several versions of this general framework are implemented, leading to five different versions of the matheuristic.

5.2.1 Matheuristic algorithm $MH_{XB}(S)$

The simplest way to re-optimize S in $MH(S)$ is to introduce the following constraints into the MILP model:

$$x_{S_{[k]},k} = 1, \ \forall k \in \{1, .., R-1\} \cup \{R+H, .., n\} \tag{5.1}$$

These $n - H$ constraints ensure that sequences A and B will not be changed in S' [Ta et al., 2014b]. However, because sequence A is known, there is no need to give to the solver a complete MILP model with n^2 binary variables. Therefore, in order to reduce the size of the MILP, we compute the completion times of the last job of sequence A on each machine and we only re-optimize XB, within an MILP model with only H^2 binary variables. This reduction is interesting for large values of R. Similarly as $MH_{AXB}(S)$, when all the values for R have been tested, a neighborhood is applied to sequence S, in order to modify and possibly improve this solution, and the procedure iterates from $R = 1$, until the stopping criterion is reached. We denote by CA_i the completion time of the last job of sequence A on machine M_i and by $\sum T_j(A)$ the total tardiness of the jobs in A. New constraints are added to the model related to the CA_i. We only indicate here these new constraints. The rest of the model is unchanged, except for the definition of indices: the problem which is solved is smaller than before and only the $n - R + 1$ jobs which are not in A are sequenced from position 1 (i.e. R) to position $n - R + 1$ (i.e. n). Notice

that the indices in the expression of the constraints (5.2) and (5.3) refer to the complete sequence S.

$$C_{1,R} = CA_1 + \sum_{j=1}^{n} p_{1,j} x_{j,R} \tag{5.2}$$

$$C_{i,R} \geq CA_i + \sum_{j=1}^{n} p_{i,j} x_{j,R}, \ \forall i \in \{2, ..., m\} \tag{5.3}$$

Remember that $C_{i,R}$ is the completion time on machine M_i of the job in position R in S, i.e. of the first job of XB. If we denote by $\sum T_j(XB^*)$ the value of the optimal solution of this model, the value of S' is given by:

$$\sum T_j(S') = \sum T_j(A) + \sum T_j(XB^*)$$

This process of the $MH_{XB}(S)$ is illustrated in **Fig. 5.2**

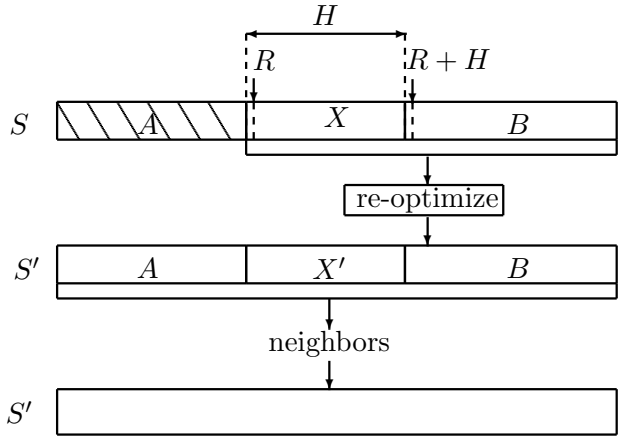


Figure 5.2: Illustration of the $MH_{XB}(S)$ algorithm

5.2.2 Matheuristic algorithm $MH_{XB_1}(S)$

This is a method similar to $MH_{XB}(S)$, but a neighborhood operator (among SWAP, EFSR, EBSR and Inversion) is applied to the sequence of jobs from position 1 to position $R-1$ (i.e. sequence A), leading to a new sequence A'. The objective function of the neighborhood operator is such that sequence A' will not penalize too much sequence X, i.e. it is a linear combination of the total tardiness of the jobs of A' and the makespan of A'. The expression of this objective function to minimize is:

$$\text{Z}{=}\alpha \left(\sum_{i \in A'} T_i \right) + (1 - \alpha) C_{m,R-1} \tag{5.4}$$

where $C_{m,R-1}$ denotes the completion time on machine M_m of the job in position $R-1$, i.e. the last job of sequence A'.

This process of the $MH_{XB_1}(S)$ is illustrated in **Fig. 5.3**.

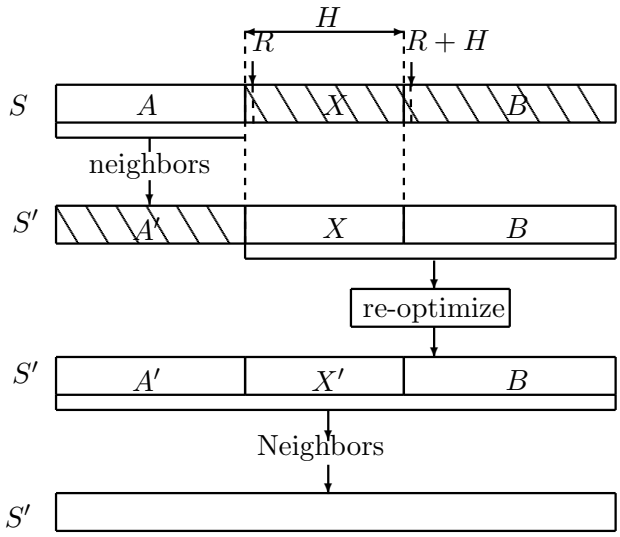


Figure 5.3: Illustration of the MH_{XB_1} algorithm

The total tardiness of A' and the completion time of the last job of A' on each machine are computed and denoted by CA_i, the constraints with CA_i are introduced in (5.2) and (5.3) as for algorithm $MH_{XB}(S)$. The value of S' is given by:

$$\sum T_j(S') = \sum T_j(A') + \sum T_j(XB^*)$$

where $\sum T_j(XB^*)$ is computed as in **Section 5.2.1**.

5.2.3 Matheuristic algorithm $MH_X(S)$

In order to continue reducing the size of the problem to be optimized by the solver, we limit now the problem exactly to the optimization of sequence X [Ta et al., 2014b]. Of course, minimizing the total tardiness of the jobs of X (sequenced after the jobs of A) and after that sequencing the jobs of B leads to a worse solution than minimizing the total tardiness of the jobs of XB, even if B is fixed. Therefore, we introduce another criterion in the objective function for the optimization of X. More precisely, in order to finish the jobs of X not too late, which could be penalizing for B, we change the objective function for a linear combination of the total tardiness of the jobs of X and the makespan of X (exactly as before for A', but here in the MILP formulation). Therefore, the objective function is equal to:

$$\text{Minimize } \alpha \left(\sum_{k=R}^{R+H-1} T_k \right) + (1-\alpha) C_{m,R+H-1} \tag{5.5}$$

where $C_{m,R+H-1}$ denotes the completion time on machine M_m of the job in position $R+H-1$, i.e. the last job of sequence X.

This process of the the $MH_X(S)$ is illustrated in **Fig. 5.4**.

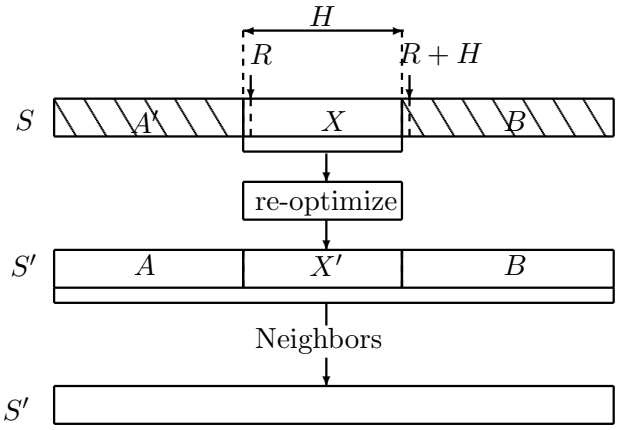


Figure 5.4: Illustration of the $MH_X(S)$ algorithm

First, the total tardiness of A and the completion time of the last job of A on each machine are computed and the constraints with CA_i are introduced in the model (same way as in $MH_{XB}(S)$). Then, the optimization of X is done by the solver, and a sequence $X' = X^*$ is obtained. Similarly as $MH_{AXB}(S)$, the sequence $S' = AX'B$ is a new sequence, hopefully better than S, and an iteration process is performed for other values of R, up to $n-H$. When all the values for R have been tested, a neighborhood is applied to sequence S, in order to modify and possibly improve this solution, and the procedure iterates from $R=1$, until the stopping criterion is reached. The completion times of the last job of X' on each machine are denoted by CX_i. Then, sequence B is scheduled after X, taking the CX_i values into account. Finally, the value of S' is given by:

$$\sum T_j(S') = \sum T_j(A) + \sum T_j(X^*) + \sum T_j(B)$$

5.2.4 Matheuristic algorithm $MH_{X_1}(S)$

We propose another matheuristic method of type MH_X called $MH_{X_1}(S)$, based also on a partial resolution of the MILP (see Section 3.1). Similarly as in Section 5.1, given is a sequence $S = AXB$, an index R and a size window H. Firstly, neighborhood operators (among SWAP, EFSR, EBSR and Inversion) are applied to the sequence A and the obtained sequence is called A'. Secondly, the sequence X is re-optimized, giving sequence X'. Finally, neighborhood operators are also applied to sequence B, giving sequence B'. The sequence $S' = A'X'B'$ is a new sequence, hopefully better than S, and the process iterates for other values of R, up to $n-H$. This process is performed until the stopping criterion is reached, i.e. a time limit called $TimeLimMH$. This matheuristic algorithm is illustrated in **Fig. 5.5** and the algorithm is given in **Algo. 13**.

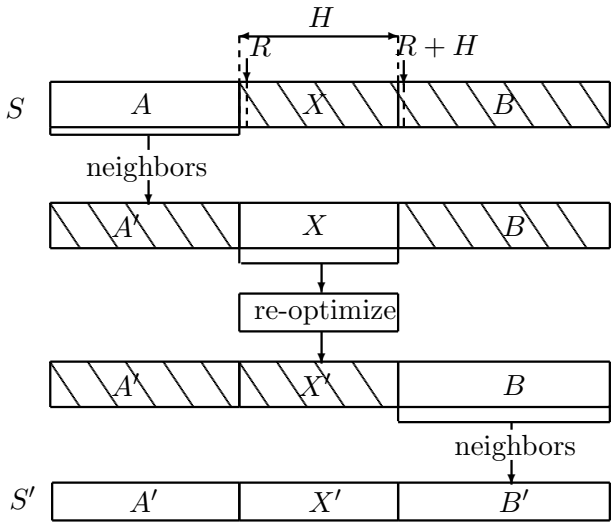


Figure 5.5: Illustration of the $MH_{X_1}(S)$ algorithm

Algorithm 13 The $MH_{X_1}(S)$ algorithm

Input: S = initial solution
improved = true
while (CPU $\leq TimeLimMH$) and (improved = true) **do**
  improved = false ; $R = 0$
  while $R \leq n - H$ **do**
    $A = (S_{[1]}, S_{[2]}, ..., S_{[R-1]})$
    A' = Neighborhood operator(A)
    $X = (S_{[R]}, S_{[R+1]}, ..., S_{[R+H-1]})$
    X' = re-optimization of X
    $B = (S_{[R+H]}, S_{[R+H+1]}, ..., S_{[n]})$
    B' = Neighborhood operator(B)
    $S' = A'X'B'$
    if $(\sum T_j(S') < \sum T_j(S))$ **then**
      improved = true ; $S = S'$
      if $(R + H \leq n - H)$ **then**
        $R = R + H - 1$
      end if
    end if
    $R = R + 1$
  end while
end while
return(S)

The objective function for finding the best possible subsequence A' is a linear combination of the total tardiness of the jobs of A' and of the makespan of A'. This objective function is equal to:

$$Z_A = \alpha \left(\sum_{i=1}^{R-1} T_i \right) + (1-\alpha) C_{m,R-1} \tag{5.6}$$

where $C_{m,R-1}$ denotes the completion time on machine M_m of the job in position $R-1$, i.e. the last job of sequence A'.

For the same reasons, the objective function for finding the best possible subsequence X' is also a linear combination of the total tardiness of the jobs of X' and of the makespan of X'. This objective function is equal to:

$$Z_X = \alpha \left(\sum_{k=R}^{R+H-1} T_k \right) + (1-\alpha) C_{m,R+H-1} \tag{5.7}$$

where $C_{m,R+H-1}$ denotes the completion time on machine M_m of the job in position $R+H-1$, i.e. the last job of sequence X'.

First, the total tardiness of A' and the completion time of the last job of A' on each machine are computed and denoted by CA_i, the constraints with CA_i are introduced in the model (same way as in $MH_{XB}(S)$). Then, the optimization of X is done by the solver, and a sequence $X' = X^*$ is obtained. The completion times of the last job of X' on each machine are denoted by CX_i. Then, sequence B' is scheduled after X, taking the CX_i values into account with the following objective function:

$$Z_B = \sum_{k=R+H}^{n} T_k$$

Finally, the value of S' is given by:

$$\sum T_j(S') = \sum T_j(A') + \sum T_j(X^*) + \sum T_j(B')$$

5.2.5 Matheuristic algorithm $MH_{X_2}(S)$

Another matheuristic method of type $MH_X(S)$ is called $MH_{X_2}(S)$. Similarly as in **Section 5.1**, the sequence A and the sequence B are unchanged. The sequence X is re-optimized, giving sequence X'. The sequence $S' = AX'B$ is a new sequence. Then, the neighborhood operators (among SWAP, EBSR, EFSR, Inversion) are applied to sequence S', in order to modify it and possibly improve this solution. The procedure iterates from $R = 1$, until the stopping criterion is reached, i.e. a time limit called $TimeLimMH$. The algorithm denoted **Algo. 14** is given below.

Similarly as in **Section 5.2.3**, the value of S' is given by:

$$\sum T_j(S')^* = \sum T_j(A) + \sum T_j(X^*) + \sum T_j(B)$$

Algorithm 14 The MH_{X_2} algorithm

Input: S = initial solution
improved = true
while (CPU $\leq TimeLimMH$) and (improved = true) **do**
  improved = false ; $R = 0$
  while $R \leq n - H$ **do**
    $A = (S_{[1]}, S_{[2]}, ..., S_{[R-1]})$
    $X = (S_{[R]}, S_{[R+1]}, ..., S_{[R+H-1]})$
    X' = re-optimization of X
    $B = (S_{[R+H]}, S_{[R+H+1]}, ..., S_{[n]})$
    $S' = AX'B$
    if $(\sum T_j(S')^* < \sum T_j(S))$ **then**
      improved = true ; $S = S'$
    end if
    S'= Neighborhood operator(S)
    Compute $(\sum T_j(S'))$
    if $(\sum T_j(S') < \sum T_j(S))$ **then**
      improved = true ; $S = S'$
      if $(R + H \leq n - H)$ **then**
        $R = R + H - 1$
      end if
    end if
    $R = R + 1$
  end while
end while
return(S)

5.3 Matheuristic algorithm with limited positions

With this method denoted by $MH_{POS}(S)$ [Ta et al., 2014b], the hypothesis is that the positions of jobs in S are not that bad, and only few changes in the subsequence X are sufficient to improve the solution.

We assume that a job in a position $k \in \{R, ..., R + H - 1\}$ may only be scheduled at a position between $k - \delta$ and $k + \delta$.

The following constraints are added to the MILP:

$$\sum_{\ell=R}^{k+\delta} x_{S_{[k]},\ell} = 1, \ \forall k \in \{R, R + \delta - 1\} \tag{5.8}$$

$$\sum_{\ell=k-\delta}^{k+\delta} x_{S_{[k]},\ell} = 1, \ \forall k \in \{R + \delta, R + H - 1 - \delta\} \tag{5.9}$$

$$\sum_{\ell=k-\delta}^{R+H-1} x_{S_{[k]},\ell} = 1, \ \forall k \in \{R + H - 2 - \delta, R + H - 1\} \tag{5.10}$$

The process is the same as in $MH_{XB}(S)$, i.e. sequence XB is re-optimized, with this limitation for the possible positions of each job. The idea is that with this limitation in the position changes, it will be possible to increase the size of H. Finally, the value of S' is given by:

$$\sum T_j(S') = \sum T_j(A) + \sum T_j(XB^*)$$

The process is illustrated in **Fig. 5.6**.

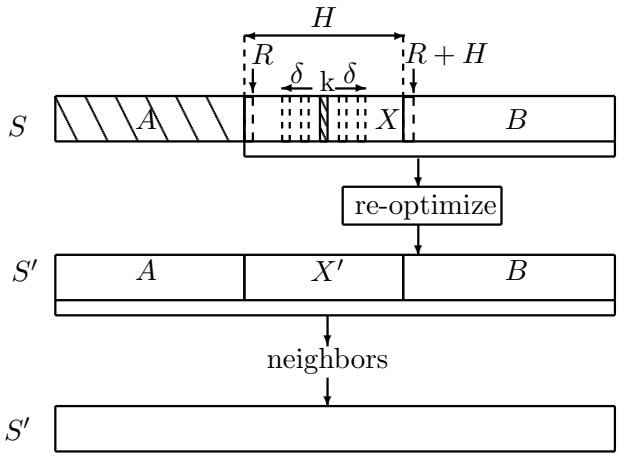


Figure 5.6: Illustration of the $MH_{POS}(S)$ algorithm

5.4 Computational experiments

These algorithms have been tested by using 108 benchmark instances proposed in [Vallada et al., 2008]. Nine instances of these benchmark instances are used for each combination of n and m, with $n \in \{50, 150, 250, 350\}$ and $m \in \{10, 30, 50\}$.

The matheuristic algorithms have been tested with several initial solutions and are compared in terms of quality.

The time limit of the matheuristic algorithms has been fixed to $TimeLimMH = (200 + n + m)$ seconds. After some preliminary experiments, the size window has been fixed to $H = 6$, the coefficient α in the linear combination for $MH_{XB}(S)$, $MH_{XB_1}(S)$, $MH_X(S)$, $MH_{X_1}(S)$ and $MH_{X_2}(S)$ has been fixed to $\alpha = 0.5$, and the coefficient δ in $MH_{POS}(S)$ has been fixed to $\delta = 3$. The solver that has been used for solving the MILP model is CPLEX v12.2.

5.4.1 Comparison of the matheuristic algorithms

Comparison of the matheuristic algorithms with EDD as an initial solution (denoted by MH(EDD))

In Tables 5.1, 5.2 and 5.3, column 'Best' for '$MH_{X_1}(EDD)$' indicates the number of times the method $MH_{X_1}(EDD)$ outperforms the other methods, column Cpu(s) indicates

the average computation time of $MH_{X_1}(EDD)$ per nine instances.

Column '$\Delta_p(EDD)$' ($p(EDD) \in \{MH_{X_1}(EDD), MH_{X_2}(EDD), MH_{XB}(EDD), MH_{XB_1}(EDD), MH_{POS}(EDD), MH_X(EDD)\}$) indicates the average deviation between $p(EDD)$ and the best matheuristic between $MH_{X_1}(EDD)$, $MH_{X_2}(EDD)$, $MH_{XB}(EDD)$, $MH_{XB_1}(EDD)$, $MH_{POS}(EDD)$ and $MH_X(EDD)$.

$$\Delta_p(EDD) = \frac{p(EDD) - \min(MH_{all}(EDD))}{p(EDD)}$$

with $MH_{all}(EDD)$ is all methods ($MH_{XB}(EDD)$, $MH_{XB_1}(EDD)$, $MH_{POS}(EDD)$, $MH_X(EDD)$, $MH_{X_1}(EDD)$, $MH_{X_2}(EDD)$)

In Tables 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, column '$Best_{all}$' for '$MH_{all}(EDD)$' or '$MH_{all}(EN)$' indicates the number of times the method '$MH_{all}(EDD)$' or '$MH_{all}(EN)$' outperforms the other matheuristic methods, where $all \in \{XB, XB_1, POS, X, X_1, X_2\}$. We can see that $MH_{X_1}(EDD)$ is the best method in terms of number of best solutions returned.

Table 5.1: Comparison of matheuristic algorithms by EDD initial solution (1)

	$MH_{X_1}(EDD)$				$MH_{X_2}(EDD)$			
$n \times m$	$Best_{all}$	Best	Cpu(s)	$\Delta_{X_1}(EDD)$	$Best_{all}$	Best	Cpu(s)	$\Delta_{X_2}(EDD)$
50 × 10	3	4	260,11	13,34%	2	2	260,09	7,03%
50 × 30	1	1	280,17	10,06%	1	1	280,15	10,89%
50 × 50	4	4	300,40	0,48%	1	2	300,39	2,87%
150 × 10	2	5	360,21	11,82%	2	3	360,04	5,96%
150 × 30	5	7	380,61	3,96%	1	1	380,03	6,46%
150 × 50	4	7	402,81	3,99%	1	1	400,21	5,07%
250 × 10	5	9	460,73	0,00%	3	3	460,02	2,87%
250 × 30	6	8	483,64	11,11%	1	1	480,05	17,25%
250 × 50	8	8	503,42	0,08%	0	0	500,11	17,00%
350 × 10	3	6	562,58	11,37%	4	4	560,01	1,28%
350 × 30	6	6	591,63	12,20%	1	1	580,08	16,11%
350 × 50	0	0	606,78	27,18%	3	6	600,04	3,46%
	47	65	428,32	8,80%	20	25	425,82	8,02%

In Tables 5.1, 5.2 and 5.3, we can also see that the $MH_{X_1}(EDD)$ matheuristic algorithm (initialized by EDD sequence) leads to the best results in terms of number of best solutions returned. On average, the deviation between the solutions returned by this method and the best solutions is 8,80%. This value is around 5,84% for $MH_X(EDD)$, 8,02% for $MH_{X_2}(EDD)$, 23,46% for $MH_{XB}(EDD)$, 36,35% for $MH_{X_1}(EDD)$ and 32,63% for $MH_{POS}(EDD)$.

Comparison of the matheuristic algorithms initialized by EN

We compared to the matheuristic algorithms are initialized by EN that is the best solution among EDD and NEH (denoted by $MH(EN)$)

Table 5.2: Comparison of matheuristic algorithms by EDD initial solution (2)

	$MH_{XB}(EDD)$				$MH_{XB_1}(EDD)$			
$n \times m$	$Best_{all}$	Best	Cpu(s)	$\Delta_{XB}(EDD)$	$Best_{all}$	Best	Cpu(s)	$\Delta_{XB_1}(EDD)$
50 × 10	2	3	260,23	13,95%	1	1	260,25	21,08%
50 × 30	2	2	281,08	8,94%	0	0	281,24	17,62%
50 × 50	1	1	304,26	4,36%	0	0	301,80	11,62%
150 × 10	2	3	360,41	4,79%	1	1	360,61	42,48%
150 × 30	1	1	384,19	13,95%	0	0	383,76	41,18%
150 × 50	0	0	418,15	16,81%	0	0	415,52	28,73%
250 × 10	3	3	462,14	10,86%	1	1	462,26	50,74%
250 × 30	1	1	487,57	35,18%	1	1	485,83	42,18%
250 × 50	0	0	524,79	44,13%	0	0	523,70	44,49%
350 × 10	2	2	562,87	37,89%	2	2	565,43	44,82%
350 × 30	1	1	594,79	45,02%	1	1	598,22	45,32%
350 × 50	0	0	636,69	45,96%	0	0	668,86	45,99%
	15	17	439,76	23,46%	7	8	437,99	36,35%

Table 5.3: Comparison of matheuristic algorithms by EDD initial solution (3)

	$MH_{POS}(EDD)$				$MH_X(EDD)$			
$n \times m$	$Best_{all}$	Best	Cpu(s)	$\Delta_{POS}(EDD)$	$Best_{all}$	Best	Cpu(s)	$\Delta_X(EDD)$
50 × 10	2	2	231,34	18,02%	2	3	266,11	2,32%
50 × 30	1	1	249,76	5,02%	3	4	280,23	8,69%
50 × 50	0	0	302,45	6,32%	2	2	300,56	1,78%
150 × 10	2	2	361,25	11,21%	3	3	260,30	5,11%
150 × 30	0	0	385,56	38,44%	3	3	380,18	2,87%
150 × 50	0	0	406,76	35,96%	0	1	400,56	5,52%
250 × 10	1	1	462,95	48,76%	3	3	461,88	3,03%
250 × 30	1	1	489,21	44,69%	2	2	486,62	5,75%
250 × 50	0	0	520,97	45,42%	1	1	504,20	15,88%
350 × 10	2	2	564,95	45,59%	3	3	588,03	4,26%
350 × 30	1	1	594,84	45,72%	4	4	608,87	3,63%
350 × 50	0	0	665,74	46,43%	4	4	636,10	1,39%
	8	9	441,35	32,63%	30	33	434,20	5,84%

In Tables 5.4, 5.5 and 5.6, column 'Best' for '$MH_{XB}(EN)$' indicates the number of times the method $MH_{XB}(EN)$ outperforms the other methods, column Cpu(s) indicates the average computation time of $MH_{XB}(EN)$ per nine instances.

Column '$\Delta_p(EN)$' ($p(EN) \in \{MH_{XB}(EN), MH_{XB_1}(EN), MH_{POS}(EN), MH_X(EN), MH_{X_1}(EN), MH_{X_2}(EN)\}$) indicates the average deviation between $p(EN)$ and the best matheuristic between $MH_{XB}(EN)$, $MH_{XB_1}(EN)$, $MH_X(EN)$, $MH_{X_1}(EN)$, $MH_{X_2}(EN)$ and $MH_{POS}(EN)$.

Table 5.4: Comparison of matheuristic algorithms by EN initial solution (1)

	$MH_{X_1}(EN)$				$MH_{X_2}(EN)$			
$n \times m$	$Best_{all}$	Best	Cpu(s)	$\Delta_{EN_{X_1}}$	$Best_{all}$	Best	Cpu(s)	$\Delta_{X_2}(EN)$
50 × 10	2	4	260,08	14,97%	2	2	260,12	6,68%
50 × 30	1	6	280,30	0,23%	0	0	280,31	4,17%
50 × 50	1	5	300,39	0,62%	0	2	300,39	2,04%
150 × 10	6	8	360,43	11,11%	2	2	360,60	7,82%
150 × 30	3	8	381,59	2,78%	1	2	380,06	6,70%
150 × 50	4	8	436,19	5,29%	1	1	400,07	6,21%
250 × 10	7	8	461,35	0,14%	3	4	460,20	4,13%
250 × 30	3	7	482,98	3,81%	1	2	480,01	5,45%
250 × 50	1	4	513,17	3,03%	0	3	500,02	12,32%
350 × 10	5	8	562,96	11,11%	3	3	560,01	3,92%
350 × 30	1	1	587,08	23,35%	1	4	580,02	1,30%
350 × 50	0	0	609,21	26,90%	3	3	600,02	3,88%
	34	67	432,09	8,61%	17	26	425,81	5,38%

Table 5.5: Comparison of matheuristic algorithms by EN initial solution (2)

	$MH_{XB}(EN)$				$MH_{XB_1}(EN)$			
$n \times m$	$Best_{all}$	Best	Cpu(s)	$\Delta_{XB}(EN)$	$Best_{all}$	Best	Cpu(s)	$\Delta_{XB_1}(EN)$
50 × 10	4	4	260,11	1,92%	1	1	260,16	22,51%
50 × 30	1	2	280,56	5,60%	0	0	281,19	6,72%
50 × 50	0	1	300,95	2,34%	0	1	301,72	2,98%
150 × 10	2	2	360,56	6,37%	1	1	360,79	42,38%
150 × 30	1	1	382,50	11,42%	0	0	385,05	32,61%
150 × 50	0	0	405,06	16,38%	0	0	410,04	20,40%
250 × 10	3	3	462,49	12,05%	1	1	461,80	49,99%
250 × 30	1	1	491,34	33,66%	1	1	489,47	33,73%
250 × 50	0	0	533,71	26,80%	0	0	529,10	26,72%
350 × 10	2	2	564,19	37,24%	2	2	565,27	42,79%
350 × 30	1	1	602,85	31,97%	1	1	622,82	31,99%
350 × 50	0	0	655,23	33,00%	0	0	658,30	33,01%
	15	17	441,85	18,26%	7	8	439,37	28,82%

$$\Delta_p(EN) = \frac{p(EN) - \min(MH_{all}(EN)}{p(EN)}$$

with $MH_{all}(EN)$ is all methods ($MH_{XB}(EN)$, $MH_{XB_1}(EN)$, $MH_{POS}(EN)$, $MH_X(EN)$, $MH_{X_1}(EN)$, $MH_{X_2}(EN)$)).

In Tables 5.6, 5.5 and 5.6, we can also see that the $MH_{X_1}(EN)$ matheuristic algorithm leads to the best results with a number of best solutions equal to 67. On average, the deviation between the solutions returned by this method and the best solutions is 8,61%. This value is around 5,82% for $MH_X(EN)$, 5,38% for $MH_{X_2}(EN)$, 18,26% for $MH_{XB}(EN)$,

Table 5.6: Comparison of matheuristic algorithms by EN initial solution(3)

	$MH_{POS}(EN)$				$MH_X(EN)$			
$n \times m$	$Best_{all}$	Best	Cpu(s)	$\Delta_{POS}(EN)$	$Best_{all}$	Best	Cpu(s)	$\Delta_X(EN)$
50 × 10	2	2	261,84	16,37%	2	3	260,01	6,23%
50 × 30	0	0	286,99	5,80%	0	1	280,02	3,94%
50 × 50	0	1	314,80	4,16%	0	1	300,03	2,97%
150 × 10	2	2	360,99	12,48%	2	2	360,53	6,68%
150 × 30	0	0	388,90	30,12%	1	1	380,59	9,28%
150 × 50	0	0	502,67	22,69%	0	0	401,20	9,16%
250 × 10	1	1	464,29	45,43%	3	3	463,30	3,04%
250 × 30	1	1	509,80	33,71%	1	2	483,60	5,34%
250 × 50	0	0	617,49	26,45%	1	3	506,20	1,98%
350 × 10	2	2	567,73	42,88%	2	2	574,53	13,29%
350 × 30	1	1	622,75	32,04%	1	6	595,83	7,78%
350 × 50	0‘	0	840,77	32,84%	2	7	618,95	0,09%
	9	10	475,61	31,20%	15	31	430,79	5,82%

28,82% for $MH_{X_1}(EN)$ and 31,20% for $MH_{POS}(EN)$.
Conclution: The results clearly show the domination of MH_{X_1} in most of the cases. We can also see that the MH_X, MH_{X_1}, MH_{X_2} are better than the MH_{XB}, MH_{XB_1}, MH_{POS} algorithms. The main reason is due to the computation time required by CPLEX for solving each MILP. For MH_{XB} and MH_{XB_1}, CPLEX takes time for loading the problem and solving it, as soon as the problem size increases, which limits the number of neighbors explored. The fact that some variables are already decided clearly helps, but is not sufficient for making this method competitive. The advantage of MH_X, MH_{X_1}, MH_{X_2} are that the problem given to CPLEX is very small and very quickly solved. Therefore, the algorithm can perform several times the loop (starting several times with $R = 1$), before reaching quickly a better solution. The method MH_{POS} is not able to find good solutions before the end of the algorithm for the same reasons. Clearly, the quality of the matheuristic is strongly related to its ability of solving quickly to optimality a big number of partial sequences.

Comparison of the $MH_{X_1}(EDD)$ and $MH_{X_1}(EN)$ algorithm

In Table 5.7, column ‘Best’ for $MH_{X_1}(EDD)$ indicates the number of times this method strictly outperforms method $MH_{X_1}(EN)$, column Cpu(s) indicates the average computation time of $MH_{X_1}(EDD)$ per nine instances. Column ‘$\Delta_{E_{X_1}}$’ indicates the average deviation between $MH_{X_1}(EDD)$ and $MH_{X_1}(EN)$.

$$\Delta_{X_1}(EDD) = \frac{MH_{X_1}(EDD) - MH_{X_1}(EN)}{MH_{X_1}(EDD)}$$

Column ‘Best’ for ‘$MH_{X_1}(EN)$’ indicates the number of times the method $MH_{X_1}(EN)$ strictly outperforms method $MH_{X_1}(EN)$, column Cpu(s) indicates the average computation time of $MH_{X_1}(EN)$ per nine instances. Column ‘$\Delta_{X_1}(EN)$’ indicates the average

deviation between $MH_{X_1}(EN)$ and $MH_{X_1}(E)$.

$$\Delta_{X_1}(EN) = \frac{MH_{X_1}(EN) - MH_{X_1}(E)}{MH_{X_1}(EN)}$$

Column 'Best' for 'EDD' indicates the number of times the EDD rule outperforms the best solution among EDD and NEH rule (denote by EN). Column 'Best' for 'EN' indicates the number of times the EN outperforms the EDD.

Table 5.7: Comparison of the $MH_{X_1}(EDD)$ and $MH_{X_1}(EN)$ algorithm

	EDD	EN	$MH_{X_1}(EDD)$			$MH_{X_1}(EN)$		
$n \times m$	Best	Best	Best	Cpu(s)	$\Delta_{X_1}(EDD)$	Best	Cpu(s)	$\Delta_{X_1}(EN)$
50 × 10	2	9	7	260,11	-1,48%	4	260,08	1,37%
50 × 30	0	9	3	280,17	1,23%	6	280,30	-1,35%
50 × 50	0	9	8	300,40	-1,97%	1	300,39	1,89%
150 × 10	3	9	3	360,21	3,00%	8	360,43	-3,46%
150 × 30	2	9	7	380,61	-2,78%	3	381,59	2,51%
50 × 50	1	9	5	402,81	-2,98%	4	436,49	2,23%
250 × 10	4	9	5	460,73	0,58%	7	461,35	-0,64%
250 × 30	2	9	7	483,64	-32,45%	3	482,98	11,36%
250 × 50	2	9	9	503,42	-11,68%	1	513,17	9,06%
350 × 10	4	9	4	562,58	10,83%	7	562,96	-45,02%
350 × 30	2	9	9	591,63	-88,65%	1	587,08	27,82%
350 × 50	2	9	5	606,78	-18,45%	4	609,21	9,21%
	24	108	72	432,76	-12,07%	49	436,34	1,25%

We can see that it is clear that $MH_{X_1}(EDD)$ outperforms $MH_{X_1}(EN)$, although EN sequence is obviously better than EDD sequence. The average deviation returned by this method and $MH_{X_1}(EDD)$ method is -12,07% and 1,25% for $MH_{X_1}(EN)$.

5.4.2 Comparison of the $MH_{X_1}(EDD)$ and $GA_1(EDD)$ algorithms

The best proposed matheuristic algorithms ($MH_{X_1}(EDD)$) is now compared to the best proposed genetic algorithm ($GA_1(EDD)$), both algorithms are initialized by EDD rule and the limited computation time is fixed to $(200 + n + m)$ seconds. In Table 5.8, column 'Best' for '$MH_{X_1}(EDD)$' indicates the number of times the method $MH_{X_1}(EDD)$ outperforms method $GA_1(EDD)$. Column '$\Delta_{X_1}(EDD)$' indicates the average deviation between $MH_{X_1}(EDD)$ and $GA_1(EDD)$.

$$\Delta_{X_1}(EDD) = \frac{MH_{X_1}(EDD) - GA_1(EDD)}{MH_{X_1}(EDD)}$$

Column 'Best' for '$GA_1(EDD)$' indicates the number of times the method $GA_1(EDD)$ outperforms method $MH_{X_1}(EDD)$, column '$\Delta_{GA_1(EDD)}$' indicates the average deviation between $GA_1(EDD)$ and $MH_{X_1}(EDD)$.

$$\Delta_{GA_1(EDD)} = \frac{GA_{EDD_1} - MH_{X_1}(EDD)}{GA_1(EDD)}$$

Table 5.8: Comparison of the best $MH_{X_1}(EDD)$ and $GA_1(EDD)$ algorithm

	$MH_{X_1}(EDD)$			$GA_1(EDD)$		
$n \times m$	Best	Cpu(s)	$\Delta_{X_1}(EDD)$	Best	Cpu(s)	$\Delta_{GA_1(EDD)}$
50 × 10	5	260,11	11,91%	5	260,01	-1,30%
50 × 30	5	280,17	0,06%	4	280,01	-0,20%
50 × 50	6	300,40	-0,62%	3	300,02	0,59%
150 × 10	7	360,21	6,30%	3	360,02	4,06%
150 × 30	7	380,61	-1,21%	3	380,02	1,14%
150 × 50	6	402,81	1,35%	3	400,05	-3,67%
250 × 10	9	460,73	-3,28%	3	460,03	3,08%
250 × 30	8	483,64	8,07%	2	480,05	2,90%
250 × 50	9	503,42	-3,59%	1	500,07	3,43%
350 × 10	7	562,58	7,39%	4	560,08	3,46%
350 × 30	6	591,63	7,42%	4	580,10	3,31%
350 × 50	0	606,78	25,36%	9	600,10	-22,68%
Sum/Avg	75	432,76	4,93%	44	430,05	-0,49%

The results clearly show that the matheuristic outperforms the genetic algorithm in most of the cases. However, the performance of the genetic algorithm is better than the performance of the matheuristic for $(n \times m) = (350 \times 50)$, We also notice that for $n = 50$ jobs, the two methods are equivalent.

5.4.3 Comparison of the TS_{EDD}, $MH_{X_1}(EDD)$ and matheuristic algorithm is initialized by TS_{EDD} ($MH_{TS}(EDD)$)

Matheuristic algorithm is initialized by TS_{EDD} denoted $MH_{TS}(EDD)$ that has limit time $TimeLimMH_{TS}(EDD) = TimeLimTS + 300$ seconds. The time limit of Tabu search has been fixed to $TimeLimTS = (n(m/2) \times 90)/1000$ seconds.

In Table 5.9, column '$Best_{X_1}$' for 'TS_{EDD}' indicates the number of times the method TS_{EDD} outperforms method $MH_{X_1}(EDD)$, column Cpu(s) indicates the average computation time of TS_{EDD} per nine instances. Column '$Best_{TS}$' for '$MH_{X_1}(EDD)$' indicates the number of times the method $MH_{X_1}(EDD)$ outperforms method TS_{EDD}, column Cpu(s) indicates the average computation time of $MH_{X_1}(EDD)$ per nine instances. Column 'Δ_{EDD}' indicates the average deviation between EDD and $MH_{X_1}(EDD)$.

$$\Delta_{EDD} = \frac{EDD - MH_{X_1}(EDD)}{EDD}$$

Column 'Best' for '$MH_{TS}(EDD)$' indicates the number of times the method $MH_{TS}(EDD)$ is strictly better than TS_{EDD} and $MH_{X_1}(EDD)$, column Cpu(s) indicates the average computation time of $MH_{TS}(EDD)$ per nine instances. Column '$\Delta_{TS}(EDD)$' indicates the average deviation between TS_{EDD} and $MH_{TS}(EDD)$.

$$\Delta_{TS}(EDD) = \frac{TS_{EDD} - MH_{TS}(EDD)}{TS_{EDD}}$$

Table 5.9: Comparison of TS and matheuristic algorithms MH_{X_1}

	TS_{EDD}		$MH_{X_1}(EDD)$			$MH_{TS}(EDD)$		
$n \times m$	$Best_{X_1}$	Cpu(s)	$Best_{TS}$	Cpu(s)	Δ_{EDD}	$Best$	Cpu(s)	$\Delta_{TS_{EDD}}$
50 × 10	8	22,02	0	260,11	57,59%	3	282,08	0,04%
50 × 30	9	67,03	0	280,17	36,32%	2	347,27	0,16%
50 × 50	9	112,04	0	300,40	29,61%	5	412,46	0,05%
150 × 10	8	67,21	0	360,21	57,69%	3	427,25	0,28%
150 × 30	4	202,53	4	380,61	54,41%	3	583,38	0,20%
150 × 50	5	338,25	4	402,81	41,85%	3	738,57	1,09%
250 × 10	4	112,39	2	460,73	56,14%	4	573,17	0,45%
250 × 30	2	338,99	6	483,68	47,74%	1	820,48	0,87%
250 × 50	2	566,82	6	503,42	47,35%	3	1066,03	1,62%
350 × 10	5	159,19	2	562,58	46,27%	6	719,25	0,66%
350 × 30	4	477,22	4	591,63	46,21%	1	1060,78	0,30%
350 × 50	9	797,10	0	606,78	37,40%	0	1394,75	0,00%
Total	69	265,74	28	428,32	46,55%	34	692,05	

As we can see, $MH_{X_1}(EDD)$ improves significantly the initial solution given by EDD, with 46,55% of improvement average. Furthermore, this method has good performances for number of jobs and machines between 150 jobs × 30 machines and 350 jobs × 30 machines, even if the computation time is relatively important. However, for other case of jobs and machines, the improvement of the initial solution is not sufficient in comparison with the Tabu search and it is clear that TS algorithm has very good performances in comparison with $MH_{X_1}(EDD)$. Of course, the method which performs the best is $MH_{TS}(EDD)$. This method quite always find a better solution than TS or $MH_{X_1}(EDD)$. However, the average deviation between TS and $MH_{TS}(EDD)$ is not that important and the computation time is greater. It seems that the TS algorithm performs well.

5.4.4 Comparison of the matheuristic algorithm is initialized by TS_{EDD} (denoted by $MH_{TS}(EDD)$) and results of [Vallada et al., 2008]

In this section, the algorithm $MH_{TS}(EDD)$ is compared to the results obtained by [Vallada et al., 2008] on 504 instances. 42 instances of these benchmark instances are used for each combination of n and m, with $n \in \{50, 150, 250, 350\}$ and $m \in \{10, 30, 50\}$.

In Table 5.10, column 'Best $MH_{TS}(EDD) \leq$ result of (Val)' indicates the number of times the method $MH_{TS}(EDD)$ is better or equal to the result of [Vallada et al., 2008]. Column 'Best $MH_{TS}(EDD) <$ result of (Val)' indicates the number of times the method $MH_{TS}(EDD)$ is strictly better than the result of [Vallada et al., 2008].

The results shows the good performances of $MH_{TS}(EDD)$ for a number machines equal to 10 and a number of jobs comprised between 50 jobs and 350 jobs. However, for the other cases, the preformances of $MH_{TS}(EDD)$ are not sufficient in comparison

with the result of [Vallada et al., 2008]. The reason is that for $m > 10$ machines, the computation time of CPLEX is too important and the performance of the matheuristic decreases.

Table 5.10: Comparison of $MH_{TS}(EDD)$ and results of [Vallada et al., 2008] (Val)

$n \times m$	Best $MH_{TS}(EDD) \leq$ Result of (Val)	Best $MH_{TS}(EDD) <$ Result of (Val)
50×10	11	0
50×30	0	0
50×50	0	0
150×10	13	2
150×30	5	0
150×50	0	0
250×10	24	10
250×30	7	0
250×50	5	0
350×10	29	20
350×30	5	1
350×50	0	0
Sum	99 / 504	33 /504

5.5 Conclusions of chapter 5

The chapter focused on the matheuristic algorithms for solving m-machine permutation flow shop scheduling problem, with the objective to minimize the total tardiness. We proposed six new matheuristic algorithms that are based on the insertion of exact solution into a neighborhood search algorithm. The solution of the Tabu search, the EDD sequence or any other sequence can be used as an input for the matheuristic algorithms. The results obtained show *firstly* the good performances of the Tabu search and then that the matheuristic always improves significantly EDD or EN (the best solution among EDD and NEH) sequence and improves the Tabu search result with a small relative deviation for large instances. *Secondly*, $MH_{X_1}(EDD)$ outperforms the genetic algorithm initialized by EDD sequence plus random instances within the same computation time, in terms of number of best solutions returned. *Thirdly*, $MH_{X_1}(EDD)$ is the best algorithm of the matheuristic proposed algorithms. *Finally*, the proposed algorithms initialized by EDD rule outperform the algorithms initialed by EN, although the EN sequence is clearly better than the EDD sequence.

Chapter 6

Integrated flow shop scheduling and vehicle routing problem

In this chapter, we consider a m-machine permutation flow shop scheduling problem and vehicle routing problem (VRP) integrated [Ta et al., 2014a]. Then, we present the resolution method based on Tabu search and the results that have been obtained. Finally, a conclusion and some future research directions are proposed.

6.1 Problem definition

We consider that the jobs have to be delivered to the customers after their production by using a single vehicle. The processing time of each job on each machine and the due date of delivery for each job are known, and a matrix of travel times is given. The jobs have to be scheduled in a m-machine flow shop environment, then batches have to be defined (one batch corresponds to one trip of the vehicle) and a route has to be determined for each batch, so that the total tardiness of delivery is minimized.

The vehicle routing problem consists in defining a route starting from the production site, visiting the customers associated to the jobs in the batch, and finishing at the production site. Each customer requiring goods is visited by the vehicle (see Figure **6.1**).

The aim of this chapter is to propose an algorithm for scheduling the jobs on the m-machines flow shop, for constituting batches of jobs and for determining the vehicle routing for each batch, so that the total tardiness of delivery is minimized. This problem is clearly an NP-hard problem [Lenstra and Rinnooy Kan, 1981].

Notations

The specific notations that are used in this chapter are the following:

- to each job J_j is associated a customer location number j, the location of the production facility has an index 0,
- the delivery time between each pair of locations i and j is denoted by $l_{i,j}$,

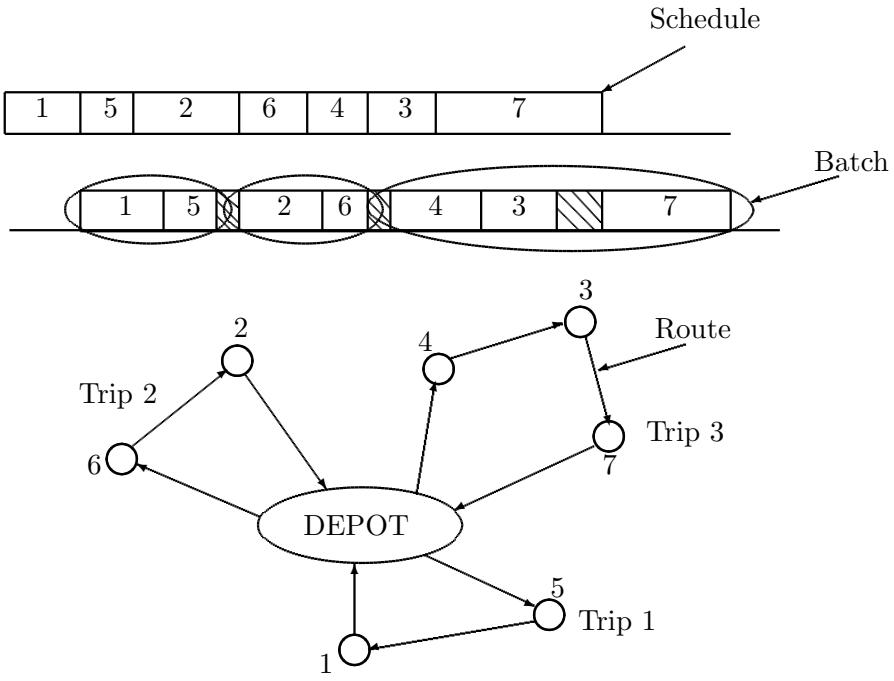


Figure 6.1: Illustration of the integrated production scheduling and vehicle routing problems

- the number of vehicles is equal to 1.

6.2 Tabu search algorithm

We only give in this section the elements of the method that are related to the problem that we consider. The basic notions about Tabu search have already been presented in Chapter 1.

6.2.1 Coding of a solution

We use an array of $3n$ elements to represent a complete solution. The first n elements represent the sequence of jobs (i.e. the schedule), the next $2n$ elements give for each trip the number of jobs in the trip and the list of jobs (the routing of these jobs is implicitely the order of the jobs in this list). In the following, the first part of the coding of a solution S (first n elements) is denoted by S^o and the second part ($2n$ elements) is denoted by S^T. The second part is decomposed into several trips. Each trip λ of S^T is denoted by S^T_λ and is composed by the number of visits k_λ and the list of visits $(S^T_{\lambda[1]}, ..., S^T_{\lambda[k_\lambda]})$. The coding is illustrated in **Fig. 6.2**.

Example: In the example presented in **Fig. 6.3**, the schedule is $\{J_1, J_5, J_2, J_6, J_4, J_3, J_7\}$, the number jobs in each trip is $(2, 2, 3, 0, 0, 0, 0)$, i.e. two jobs in the first two trips and three jobs in the third trip, the remaining trips are empty.

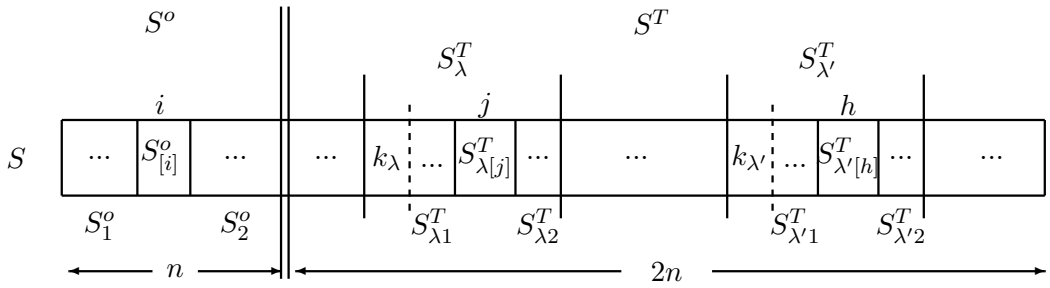


Figure 6.2: Illustration of the coding of a solution and notations

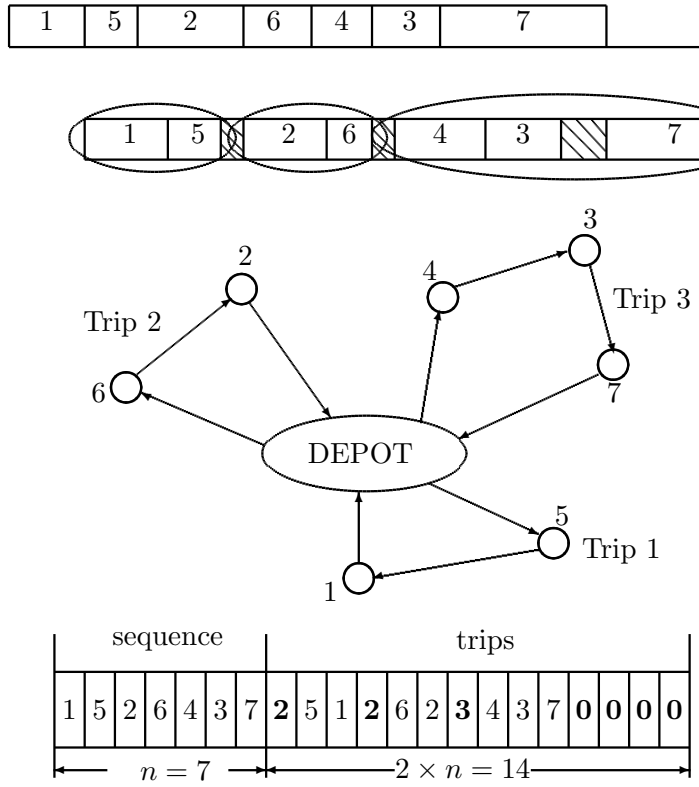


Figure 6.3: Example of the coding of a solution

6.2.2 Initial solution

EDD algorithm is used for giving an initial solution for the scheduling problem. For the trips, each trip contains only one job. The customers are visited in the same order as in the sequence.

Example: An example of an initial solution is given in **Fig. 6.4**:

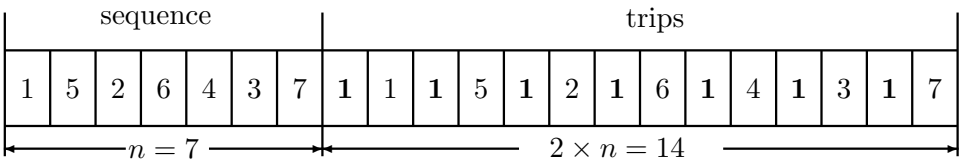


Figure 6.4: Example of initial solution

6.2.3 Neighborhood definitions

We assume that $S = S^o//S^T$ is the current complete solution with sequence S^o and trips S^T, where:

- $S^o = S^o_1/S^o_{[i]}/S^o_2/S^o_{[j]}/S^o_3$ with $S^o_{[i]}$ and $S^o_{[j]}$ the jobs in positions i and j $(i < j)$ in S^o and S^o_1, S^o_2 and S^o_3 three partial sequences.
- $S^T = k_1, S^T_1/\ldots/k_{\lambda-1}, S^T_{\lambda-1}/k_\lambda, S^T_\lambda/k_{\lambda+1}, S^T_{\lambda+1}/\ldots/k_n, S^T_n$ with k_λ the number of jobs in trip λ, S^T_λ is the sequence of jobs in trip λ, $\forall\lambda \in \{1, 2, ..., n\}$.

If $k_\lambda = 0$ then S^T_λ is empty. Notice also that $\sum_{\lambda=1}^{n} k_\lambda = n$.

Neighborhood based on the sequence of jobs

We use $SWAP^o$ operator for creating the neighbors of sequence S^o. This operator is the same as the one described in chapter 4 (see **Sections 4.2.2** and **4.3.1**).

- $SWAP^o$: A neighbor of S is created by interchanging the jobs in position i and j in sequence S^o. The corresponding jobs are also swapped in S^T. See Fig. 6.5 for an illustration of this operator.

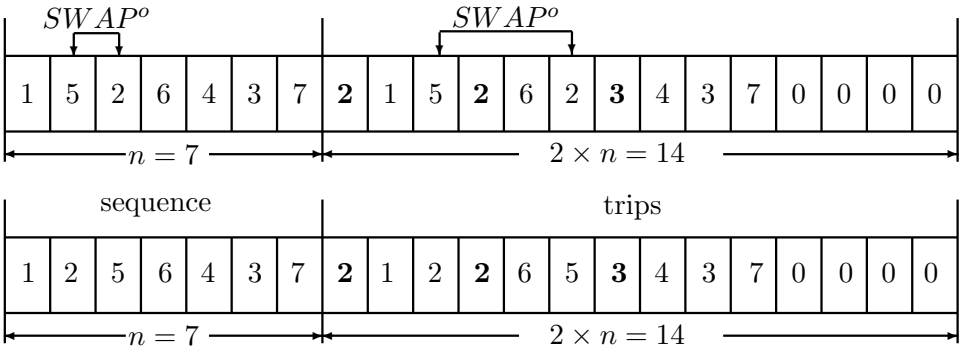


Figure 6.5: Illustration of $SWAP^o$ operator

Neighborhood based on a single trip

We use $SWAP^T$, $EBSR^T$, $ESFR^T$ and $Inversion^T$ operators for creating the neighbors of a trip S^T_λ. These operators are the same as those described in Chapter 4 (see **Sections 4.2.2** and **4.3.1**).

Given a sequence of visits S_λ^T of trip λ, two random positions i and j in S_λ^T ($i < j$ and $j \leq k_\lambda$): $S_\lambda^T = S_{\lambda 1}^T, S_{\lambda[i]}^T, S_{\lambda 2}^T, S_{\lambda[j]}^T, S_{\lambda 3}^T$.

- $SWAP^T$: A neighbor of S_λ^T is created by interchanging the jobs in position i and j, leading to sequence $S_\lambda^{T\prime} = S_{\lambda 1}^T, S_{\lambda[j]}^T, S_{\lambda 2}^T, S_{\lambda[i]}^T, S_{\lambda 3}^T$. See **Fig. 6.6** for example.

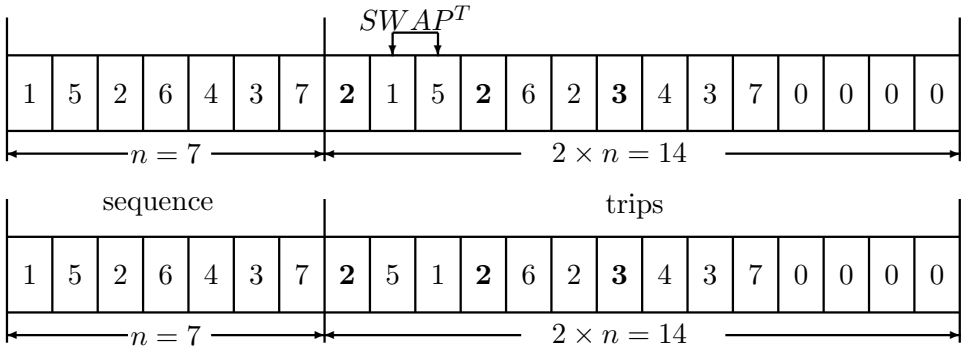


Figure 6.6: Illustration of $SWAP^T$ operator for a trip

- $EBSR^T$: A neighbor of S_λ^T is created by extracting the visit in position j and re-inserting this visit backward just before the visit in position i, leading to sequence $S_\lambda^{T\prime} = S_{\lambda 1}^T, S_{\lambda[j]}^T, S_{\lambda[i]}^T, S_{\lambda 2}^T, S_{\lambda 3}^T$. See **Fig. 6.7** for example.

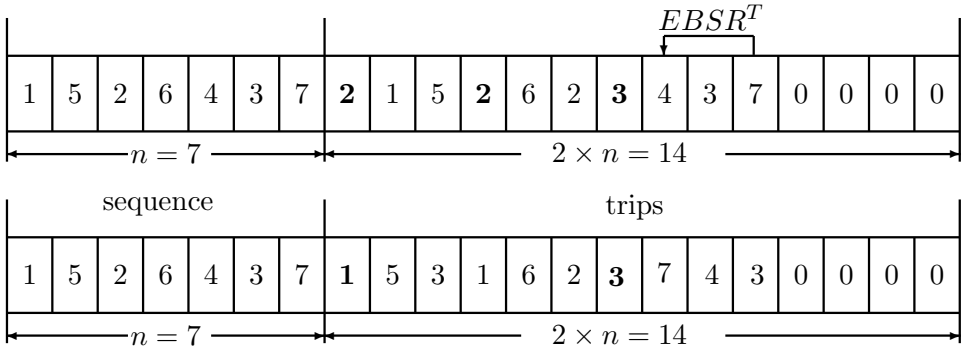


Figure 6.7: Illustration of $EBSR^T$ operator for a trip

- $EFSR^T$: A neighbor of S_λ^T is created by extracting the visit in position i and re-inserting it forward immediately after the visit in position j, leading to a sequence $S_\lambda^{T\prime} = S_{\lambda 1}^T, S_{\lambda 2}^T, S_{\lambda[j]}^T, S_{\lambda[i]}^T, S_{\lambda 3}^T$. See **Fig. 6.8** for example.
- $Inversion^T$: A neighbor of S_λ^T is created by inserting $S_{\lambda[i]}^T, S_{\lambda 2}^T, S_{\lambda[j]}^T$ in the inverse order between $S_{\lambda 1}^T$ and $S_{\lambda 3}^T$. So for example (see **Fig. 6.9**).

Neighborhood based on two trips

Given random trip in S^T, denoted by λ and μ ($\mu = \lambda + 1$, $i \leq k_\lambda$, $k_\lambda \neq 0$, $j \leq k_\mu$, $k_\mu \neq 0$) trip in S^T (corresponding to S_λ^T and S_μ^T):
$/k_\lambda, S_\lambda^T/.../k_\mu, S_\mu^T/ = /k_\lambda, S_{\lambda 1}^T/S_{\lambda[i]}^T/S_{\lambda 2}^T/.../k_\mu, S_{\mu 1}^T/S_{\mu[j]}^T/S_{\mu 2}^T/$ with:

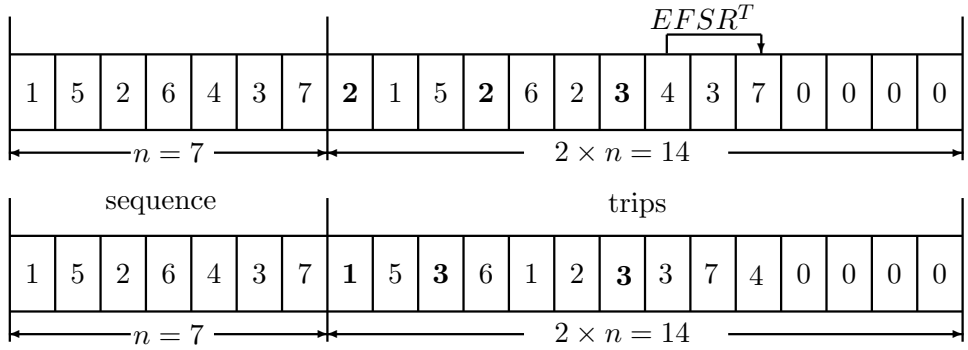


Figure 6.8: Illustration of $EFSR^T$ operator for a trip

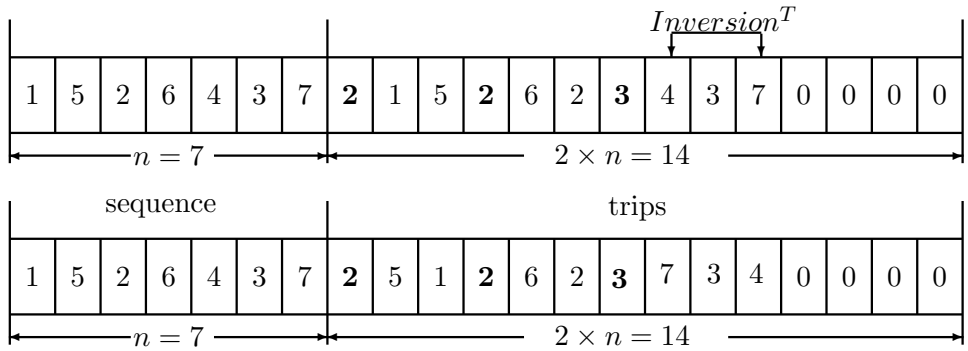


Figure 6.9: Illustration of $Inversion^T$ operator for a trip

- $S^T_{\lambda 1}$, $S^T_{\lambda 2}$ two subsequences of visits and $S^T_{\lambda[i]}$ the visit in position i in S^T_λ sequence $(i \leq k_\lambda)$

- $S^T_{\mu 1}$, $S^T_{\mu 2}$ two subsequences of visits and $S^T_{\mu[j]}$ the visit in position j in S^T_μ sequence $(j \leq k_\mu)$.

We define the following neighborhood operators:

- $SWAP^T_2$: A neighbor of $k_\lambda, S^T_\lambda // k_\mu, S^T_\mu$ is created by interchanging the jobs in position i and j, leading to sequence

  $/k_\lambda, S^{T\prime}_\lambda / ... / k_\mu, S^{T\prime}_\mu / = /k_\lambda, S^T_{\lambda 1} / S^T_{\mu[j]} / S^T_{\lambda 2} / ... / k_\mu, S^T_{\mu 1} / S^T_{\lambda[i]} / S^T_{\mu 2} /$. See for example **Fig. 6.10**.

- $EBSR^T_2$: this operator is built with the same idea. The number of visits in trips λ and μ have to be updated. A particular treatment has to be realized if a trip becomes empty. See for example **Fig. 6.11**.

- $EFSR^T_2$: this operator is built with the same idea. The number of visits in trips λ and μ have to be updated. A particular treatment has to be realized if a trip becomes empty. See for example **Fig. 6.12**.

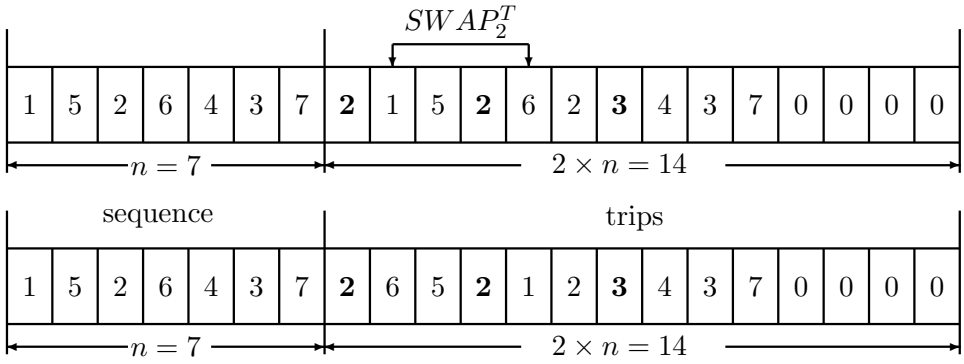


Figure 6.10: Illustration of $SWAP_2^T$ in the two-trip swap operator

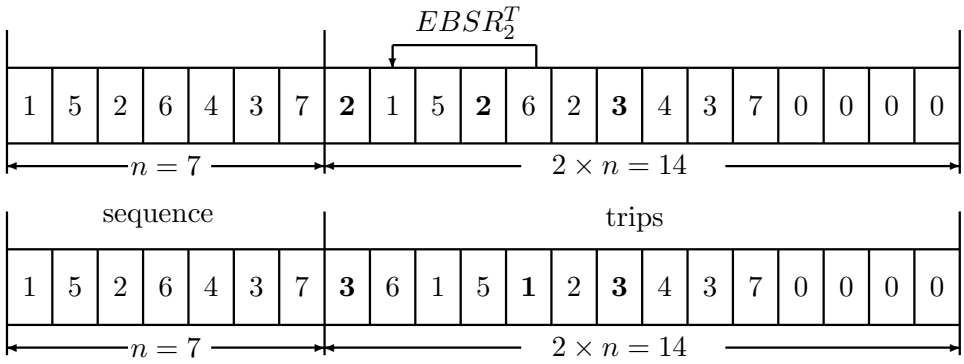


Figure 6.11: Illustration of $EBSR_2^T$ in the two-trip EBSR operator

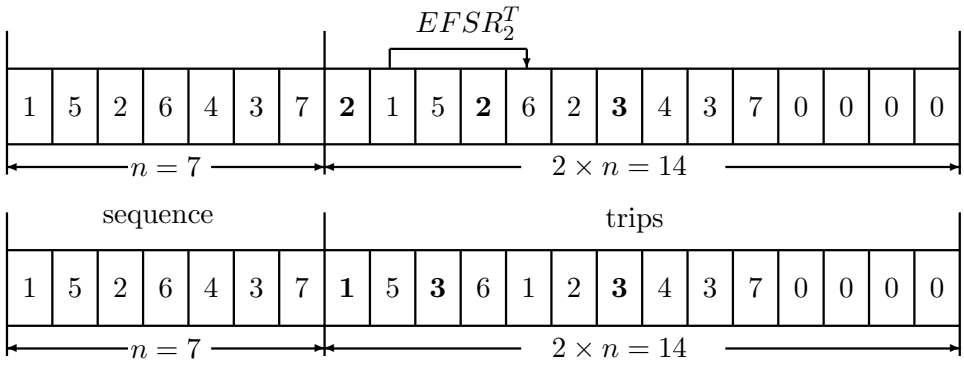


Figure 6.12: Illustration of $EFSR_2^T$ in two-trip EFSR operator

Moves and selection of the best neighbor

The aim of the problem is to minimize the total tardiness. The best neighbor in the candidate item that is non-tabu and with the smallest total tardiness. For managing the tabu list, we use the first-in-first-out (FIFO) strategy. Old attributes are deleted as new attributes are inserted.

Tabu list

We define three tabu lists and the size of the tabu lists is fixed. An element of the tabu list is defined by (λ, μ, J_i, J_j), λ, μ are the indexes of two trips, J_i and J_j are two jobs:

- For a swap of two jobs J_i and J_j in the sequence, we insert in the Tabu list the element $(0, 0, J_i, J_j)$. For example, see **Fig. 6.5**, the element inserted in the Tabu list is $(0, 0, J_5, J_2)$. In this case $\lambda = \mu = 0$.
- For a move in two trips λ and μ, we insert in the Tabu list the element (λ, μ, J_i, J_j) where i and j are the positions that are concerned in trips λ and μ respectively. For a move in a single trip, we have $\lambda = \mu$. For example, see **Fig. 6.6**, where the tabu list is $(1, 1, J_1, J_5)$ and **Fig. 6.12**, where the tabu list is $(1, 2, J_1, J_6)$.

Stopping condition

The algorithm is stopped when the time limit has been reached. This time limit is denoted by $TimeLimitTS = n(m/2)t$ ms [Vallada et al., 2008], where $t = 90$.

Detailed algorithm

The detailed TS algorithm is given in **Algo. 15**.

- $FlagSwap^o$ allow to make a selection of the neighbor operator of sequence.
- $LimitSwap^o$ allow to limit the size of the neighborhood in sequence.
- $FlagOpera^T$ allow to make a selection of the neighbor operators in sequence of a trip ($FlagOpera^T \in \{FlagSwap^T, FlagEBSR^T, FlagEFSR^T, FlagInversion^T\}$).
- $LimitOpera^T$ allow to limit the size of the neighborhood in a trip ($LimitOpera_T \in \{LimitSwap^T, LimitEBSR^T, LimitEFSR^T, LimitInversion^T\}$).
- $FlagOpera_2^T$ allow to make a selection of the neighbor operators in sequence of two different trips ($FlagOpera_2^T \in \{FlagSwap_2^T, FlagEBSR_2^T, FlagEFSR_2^T\}$).
- $Del(T)$ deletes the upper element of TS
- $Add(T, (\lambda, \mu, J_i, J_j))$ adds element $(T, (\lambda, \mu, J_i, J_j))$ to T (tabu list), $\forall \lambda, \mu \in \{0, 1, 2, ...\}$.

In **Algo. 15**, the Test($SWAP^o$), Test($Opera^T$), Test($Opera_2^T$) are described in **Algo. 16**, **17**, **18**:

6.3 Computational experiments

We present in this section the generation of data, and we discuss the results.

Algorithm 15 Tabu search algorithm for scheduling and vehicle routing

Initialization
S_0 = initial solution, S = current solution
$S' = S_0$ // best solution of $N(S)$
$S^* = S_0$ // best solution of $N(S)$ and *non-tabu*
$f^* = f(S_0)$ // f^* value of S^* and $f(S_0)$ value of S_0
$T = \emptyset$ // T is the tabu list
while (CPU $\leq$ *TimeLimitTS*) **do**
  $f(S') = \infty$,
  //Selecting neighbor in sequence
  for $i = 0$ **to** $n - 1$ **do**
    for $j = i + 1$ **to** n **do**
      Test($SWAP^o$)
    end for
  end for
  //Selecting neighbor of a trip
  for $\lambda = 0$ **to** $n - 1$ **do**
    for $i = 0$ **to** $nbjoboftrip[\lambda] - 2$ **do**
      //$nbjoboftrip[\lambda]$ is the number jobs of trip λ
      for $j = i + 1$ **to** $nbjoboftrip[\lambda] - 1$ **do**
        Test($Opera^T$)
      end for
    end for
  end for
  //Selecting neighbor in two trips
  for $\lambda = 0$ **to** $n - 2$ **do**
    for $\mu = 0$ **to** $n - 1$ **do**
      for $i = 0$ **to** $nbjoboftrip[\lambda] - 1$ **do**
        //$nbjoboftrip[\lambda]$, $nbjoboftrip[\mu]$ are the number jobs of trip λ, μ
        for $j = 0$ **to** $nbjoboftrip[\mu] - 1$ **do**
          Test($Opera_2^T$)
        end for
      end for
    end for
  end for
  if ($f(S') < f^*$) **then** $S^* = S'$, $f^* = f(S)$, **end if**
  if (SizeTabu $\geq$ TabuMax) **then** Del(T) **end if**
  Add($T, (\lambda, \mu, i, j)$)
end while

Generation data

We have tested the algorithms on a PC Intel $core^{TM}i5$ CPU 2.4GHz. Data sets have been randomly generated (notice that there is no benchmark instance for the m-machine flowshop and vehicle routing problem integrated). The processing times $p_{i,j}$ have been

Algorithm 16 Test($SWAP^o$)

if ($FlagSwap^o = 1$) and ($j - i \leq LimitSwap^o$) **then**
  $S = S'$, $f(S) = f(S')$, $SWAP^o(S, (0, 0, i, j))$,
  if $((0, 0, i, j) \notin T)$ **then**
    Calculate($f(S)$),
    if ($f(S) < f(S')$) **then** $S' = S$, $f(S') = f(S)$, move $= (0, 0, i, j)$, **end if**
  end if
end if

Algorithm 17 Test($Opare^T$)

if ($FlagOpera^T = 1$) and ($j - i \leq LimitOpera^T$) **then**
  $S = S'$, $f(S) = f(S')$, $Opera^T(S, (\lambda, \lambda, i, j))$,
  if $((\lambda, \lambda, i, j) \notin T)$ **then**
    Calculate($f(S)$),
    if ($f(S) < f(S')$) **then** $S' = S$, $f(S') = f(S)$, move $= (\lambda, \lambda, i, j)$, **end if**
  end if
end if

Algorithm 18 Test($Opare_2^T$)

if ($FlagOpera_2^T = 1$) **then**
  $S = S'$, $f(S) = f(S')$, $Opera_2^T(S, (\lambda, \mu, i, j))$,
  if $((\lambda, \mu, i, j) \notin T)$ **then**
    Calculate($f(S)$),
    if ($f(S) < f(S')$) **then** $S' = S$, $f(S') = f(S)$, move $= (\lambda, \mu, i, j)$), **end if**
  end if
end if

generated in [1,100], the due dates d_j have been generated in $[50, 50n]$, the position of "custom" j is given by its coordinates (x_j, y_j) generated in $[1, 70]$ (see **Fig. 6.13**). The delivery time $l_{i,j}$ is the classical euclidian distance:

$$l_{i,j} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$$

Ten instances are used for each combination of n and m, with $n \in \{20, 30, 50, 70, 100, 150, 200\}$ and $m \in \{2, 4\}$.

For the TS algorithm, some preliminary experiments have conducted to the following parameters settings: $TimeLimitTS = 10$ seconds, $Tabu\_list = \{10, 20, 40, 80\}$ elements.

Results

In Table 6.1, column 'Best' for TS_X ($X \in \{10, 20, 40, 80\}$) indicates the number of times this method strictly outperforms other method. Column 'Δ_{TS_X}' indicates the average deviation between TS_X and the best method between TS_{10}, TS_{20}, TS_{40} and TS_{80}.

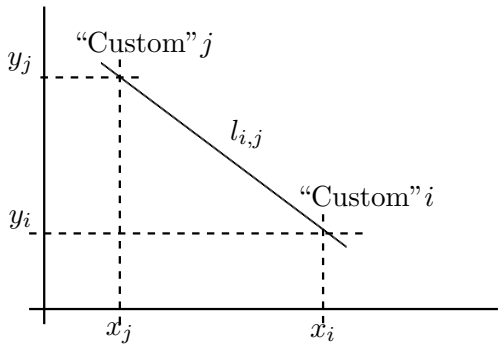


Figure 6.13: Illustration of calculation of $l_{i,j}$

$$\Delta_{TS_X} = \frac{TS_X - min(TS_{10}, TS_{20}, TS_{40}, TS_{80})}{TS_X}$$

Table 6.1: Comparison of the Tabu search algorithms

	TS_{10}		TS_{20}		TS_{40}		TS_{80}	
$n \times m$	Best	$\Delta_{TS_{10}}$	Best	$\Delta_{TS_{20}}$	Best	$\Delta_{TS_{40}}$	Best	$\Delta_{TS_{80}}$
20 × 2	3	17,58%	7	2,16%	3	17,58%	0	24,81%
30 × 2	1	11,62%	7	13,34%	3	10,97%	0	30,19%
50 × 2	0	17,40%	1	21,81%	2	12,49%	7	3,23%
70 × 2	0	16,12%	0	21,88%	2	10,99%	8	2,50%
100 × 2	1	19,24%	1	25,61%	0	14,93%	8	0,65%
150 × 2	0	23,42%	0	15,27%	4	5,29%	6	5,95%
200 × 2	0	26,35%	5	3,60%	2	6,76%	3	7,19%
20 × 4	0	9,51%	10	0,00%	0	9,51%	0	12,04%
30 × 4	0	9,28%	10	0,00%	0	9,28%	0	17,63%
50 × 4	0	8,67%	5	3,19%	2	3,47%	1	4,60%
70 × 4	1	7,34%	2	9,54%	3	5,38%	5	4,47%
100 × 4	0	18,60%	3	9,84%	2	9,23%	5	4,75%
150 × 4	0	20,08%	2	2,84%	4	3,08%	4	4,10%
200 × 4	0	18,52%	3	2,60%	5	4,16%	2	10,73%
	6	15,98%	56	9,41%	32	8,79%	49	9,49%

We can see that the TS_{20} (with tabu list is 20) leads to the best results with a number of best solutions equal to 56. On average, the deviation between the solutions returned by this method and the best solutions is 9,41%. This value is around and 15,98% for TS_{10}, 8,79% for TS_{40} and 9,49% for TS_{80}.

In Table 6.2, column 'Best TS < EDD' indicates the number of times the method TS outperforms method EDD, column Cpu(s) indicates the average computation time of TS per 10 instances. Column 'Δ' indicates the average deviation between EDD and TS.

$$\Delta = \frac{EDD - TS}{EDD}$$

Table 6.2: Comparison of the TS_{20} and EDD algorithm

$n \times m$	Best TS<EDD	Cpu(s)	Δ
20 × 2	10	10,00	72,0%
30 × 2	10	10,00	79,7%
50 × 2	10	10,00	88,3%
70 × 2	10	10,01	85,2%
100 × 2	10	10,01	91,4%
150 × 2	10	10,03	90,0%
200 × 2	10	10,06	81,3%
20 × 4	10	10,00	55,9%
30 × 4	10	10,00	69,2%
50 × 4	10	10,00	75,9%
70 × 4	10	10,01	71,4%
100 × 4	10	10,01	83,7%
150 × 4	10	10,03	77,7%
200 × 4	10	10,03	70,4%

As we can see in Table 6.2, TS improves significantly the initial solution given by EDD, with 78,0% of improvement in average.

6.4 Conclusions of chapter 6

We approach a problem where a m-machine permutation flow shop scheduling problem and a vehicle routing problem are integrated to minimize the total tardiness. To our knowledge, this is the first time that this problem is approached in the literature. We present a direct coding for a complete solution and a neighborhood method for finding a sequence and trips. We propose a tabu search algorithm for this problem, the first results show that the TS greatly improves the initial solution given by EDD and where each trip serves only one job at a time.

In the future, the first research directions are about the evaluation of the method. We will develop genetic algorithms and other methods in order to see the real quality of the tabu search. Then, we will combine mathematical programming and local search (*matheuristic*), in order to see if matheuristic are performing methods for this problem.

The work described in this chapter has been supported by the ANR project called "ATHENA".

Conclusion

In this thesis, we consider a permutation flow shop scheduling problem of m machines where the objective function is to minimize the total tardiness. We propose exact methods, classical heuristic algorithms and matheuristic algorithms for this problem. The matheuristic methods are a new type of approximated algorithms that have been proposed for solving efficiently some combinatorial optimization problems. These methods embed exact resolution into (meta)heuristic approaches. This type of resolution method has received a great interest because of their very good performances for solving some difficult problems.

Beginning with Chapter 1, we introduce the basic concepts and components of a scheduling problem and the aspects related to these components. We give a brief introduction to the theory of scheduling and present an overview of the resolution methods. We provide the description of the state-of-the-art methods for the problem in Chapter 2. The exact methods and (meta)heuristic approaches are presented in the literature review concerning the flow shop scheduling problem with the minimization of the total tardiness. The matheuristic method of Della Croce [Della Croce et al., 2011] is provided for the $F2||\sum C_j$ problem. In Chapter 3, we propose and describe mixed integer linear programming formulations and branch-and-bound algorithms for the problem. Dominance conditions are used to prune nodes. We have evaluated the methods with a random data set with small to medium instances for the two-machine case. We have also developed a new hybrid lower bound algorithm for improving the lower bounds. The new lower bound is based on a partial relaxation of the integrity of variables of the MILP model. This lower bound has good performances for small instances, but is not usable for large instances, due to the size of the MILP and to the number of binary variables introduced. Many approximate algorithms (*NEH, EDD, a Beam Search, a Recovering Beam Search, a Genetic algorithm and a Tabu Search algorithm*) have been proposed for solving the problem in Chapter 4. Many neighborhood operators have also applied for the methods. We have coded and evaluated the algorithms in order to test their performance with benchmark instances. From the heuristics and metaheuristics algorithms we can conclude that:

- The algorithms that are initiated by EDD heuristic are always better than the algorithms initiated by EN, NEH or $E\&N$.
- We can also say that algorithms (BS, RBS, GA and TS) perform very well with EDD as an initial heuristic.
- From the methods tested, the Tabu Search algorithm that is proposed outperforms

all the other methods evaluated. We also show that the genetic algorithm is a good metaheuristic for the problem.

We have proposed new matheuristic algorithms for solving the problem in Chapter 5. The matheuristics are based on the insertion of exact partial solutions into a neighborhood search algorithm. Several versions of these algorithms have been derived, depending on how the initial sequence is obtained and which subproblem is optimized. In these methods, the solver for the MILP model is called iteratively. These methods allow to improve the initial solution. With the same initial solution, the best matheuristic method performs better than the best genetic algorithm, but the performance is less evident in comparison with the best Tabu Search algorithm. In Chapter 6, we have considered a problem where a m-machine permutation flow shop scheduling problem and a vehicle routing problem are integrated to minimize the total tardiness. We have presented a direct coding for a complete solution and a neighborhood method for finding a sequence and trips. We have proposed a tabu search algorithm for this problem, the results have showed that the TS greatly improves the initial solution given by EDD heuristic and where each trip serves only one job at a time.

Several directions can be considered for a future research:

- to embed the resolution of the MILP into the genetic algorithm or into another metaheuristic, as a new neighborhood structure.
- to find other crossover and mutation operators, for improving the genetic algorithm.
- a branch-and-bound algorithm could be used in the mutation operator.
- to embed the resolution of the MILP into the Tabu search or into another metaheuristic as a new neighborhood operator.
- to propose a simulated annealing algorithm to be compared to the matheuristic algorithms for m-machine permutation flow shop scheduling problem.
- to test the matheuristics with initial solution by another method than the same GA_{EDD_1}
- For the integrated problem described in Chap. 6, we will develop genetic algorithms and other methods, in order to see the real quality of the tabu search. Then, we will combine mathematical programming and local search (*matheuristic*), in order to see if matheuristic are performing methods.
- Finally, we will develop all the matheuristic algorithms proposed in Chapter 5 for the integrated flow shop scheduling and vehicle routing problems.

Appendixes

Appendix A

The detailed example of all the phases of DP algorithm

Let consider the same instance as before. We have the first phase (see Table A.1) with one job. The tables to be done for $e_1 = \{J_1\}$, $e_1 = \{J_2\}$, $e_1 = \{J_3\}$, $e_1 = \{J_4\}$.

Table A.1: The first phase of an example with one job

e_1	$\{J_1\}$			$\{J_2\}$			$\{J_3\}$			$\{J_4\}$		
J_j	J_1			J_2			J_3			J_4		
$ip_{1,j}$	4			2			1			3		
t	$C_j(t)$	t'	F_1	$C_j(t)$	t'	F_1	$C_j(t)$	t'	F_1	$C_j(t)$	t'	F_1
0	6	2	6	7	5	7	6	5	6	6	3	6
1	6	2	6	7	5	7	6	5	6	6	3	6
2	6	2	6	7	5	7	7	6	7	6	3	6
3	6	2	6	8	6	8	8	7	8	6	3	6
4	6	2	6	9	7	9	9	8	9	7	4	7
5	7	3	7	10	8	10	10	9	10	8	5	8
6	8	4	8	11	9	11	11	10	11	9	6	9
7	9	5	9	12	10	12	12	11	12	10	7	10
8	10	6	10	13	11	13	13	12	13	11	8	11
9	11	7	11	14	12	14	14	13	14	12	9	12
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Then the second phase where two jobs are scheduled. The value of $f_2(e_2, t)$ is indicated by a star (see Table .A.2, A.3). Similar tables have to be done for $e_2 = \{J_1, J_3\}$, $e_2 = \{J_1, J_4\}$, $e_2 = \{J_2, J_3\}$, $e_2 = \{J_2, J_4\}$ and $e_2 = \{J_3, J_4\}$. The tables have to be completed with $J_j = J_1$, $J_j = J_2$, $J_j = J_3$, $J_j = J_4$.

Table A.2: The second phase of an example with two jobs (1)

e_2	$\{J_1, J_2\}$								$\{J_1, J_3\}$							
J_j	J_1				J_2				J_1				J_3			
$ip_{1,j}$	8				4				8				2			
t	$C_j(t)$	t'	F_1	F_2	$C_j(t)$	t'	F_1	F_2	$C_j(t)$	t'	F_1	F_2	$C_j(t)$	t'	F_1	F_2
0	6	2	7	17	7	5	7	16*	6	2	7	17	6	5	7	14*
1	6	2	7	17	7	5	7	16*	6	2	7	17	6	5	7	14*
2	6	2	7	17	7	5	7	16*	6	2	7	17	7	6	8	16*
3	6	2	7	17*	8	6	8	18	6	2	7	17*	8	7	9	18
4	6	2	7	17*	9	7	9	20	2	2	7	17*	9	8	10	20
5	7	3	8	19*	10	8	10	22	7	3	8	19*	10	9	11	22
6	8	4	9	21*	11	9	11	24	8	4	9	21*	11	10	12	24
7	9	5	10	23*	12	10	12	26	9	5	10	23*	12	11	13	26
8	10	6	11	25*	13	11	13	28	10	6	11	25*	13	12	14	28
9	11	7	12	27*	14	12	14	30	11	7	12	27	14	13	15	30
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

e_2	$\{J_1, J_4\}$								$\{J_2, J_3\}$							
J_j	J_1				J_4				J_2				J_3			
$ip_{1,j}$	8				6				4				2			
t	$C_j(t)$	t'	F_1	F_2	$C_j(t)$	t'	F_1	F_2	$C_j(t)$	t'	F_1	F_2	$C_j(t)$	t'	F_1	F_2
0	6	2	6	16	6	3	6	15*	7	5	10	19	6	5	10	17*
1	6	2	6	16	6	3	6	15*	7	5	10	19	6	5	10	17*
2	6	2	6	16	6	3	6	15*	7	5	10	19*	7	6	11	19*
3	6	2	6	16	6	3	6	15*	8	6	11	21*	8	7	12	21*
4	6	2	6	16*	7	4	6	16*	9	7	12	23*	9	8	13	23*
5	7	3	6	17*	8	5	7	18	10	8	13	25*	10	9	14	25*
6	8	4	7	19*	9	6	8	20*	11	9	14	27*	11	10	15	27*
7	9	5	8	21*	10	7	9	22	12	10	15	29*	12	11	16	29*
8	10	6	9	23*	11	8	10	24	13	11	16	31*	13	12	17	31*
9	11	7	10	25*	12	9	11	26	14	12	17	33*	14	13	18	33*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table A.3: The second phase of an example with two jobs (2)

e_2	$\{J_2, J_4\}$								$\{J_3, J_4\}$							
J_j	J_2				J_4				J_3				J_4			
$ip_{1,j}$																
t	$C_j(t)$	t'	F_1	F_2	$C_j(t)$	t'	F_1	F_2	$C_j(t)$	t'	F_1	F_2	$C_j(t)$	t'	F_1	F_2
0	7	5	8	17*	6	3	8	17*	6	5	8	15*	6	3	8	17
1	7	5	8	17*	6	3	8	17*	6	5	8	15*	6	3	8	17
2	7	5	8	17*	6	3	8	17*	7	6	9	17*	7	3	8	17*
3	8	6	9	19	6	3	8	17*	8	7	10	19	6	3	8	17*
4	9	7	10	21	7	4	9	19*	9	8	11	21	7	4	9	19*
5	10	8	11	23	8	5	10	21*	10	9	12	23	8	5	10	21*
6	11	9	12	25	9	6	11	23*	11	10	13	25	9	6	11	23*
7	12	10	13	27	10	7	12	25*	12	11	14	27	10	7	12	25*
8	13	11	14	29	11	8	13	27*	13	12	15	29	11	8	13	27*
9	14	12	15	31	12	9	14	29*	14	13	16	31	12	9	14	29*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

The third phase with $e_3 = \{J_1, J_2, J_3\}$ (see Table .A.4), this table has to be completed with $J_j = J_1$, $J_j = J_2$ and $J_j = J_3$ is the following.

Table A.4: The third phase of an example with third jobs (1)

e_3	$\{J_1, J_2, J_3\}$											
J_j	J_1				J_2				J_3			
$ip_{1,j}$	12				6				3			
t	$C_j(t)$	t'	F_2	F_3	$C_j(t)$	t'	F_2	F_3	$C_j(t)$	t'	F_2	F_3
0	6	2	19	33	7	5	19	30	6	5	19	27*
1	6	2	19	33	7	5	19	30	6	5	19	27*
2	6	2	19	33	7	5	19	30*	7	6	21	30*
3	6	2	19	33*	8	6	21	33*	8	7	23	33*
4	6	2	19	33*	9	7	23	36	9	8	25	36
5	7	3	21	36*	10	8	25	39	10	9	27	39
6	8	4	23	39*	11	9	27	42	11	10	29	42
7	9	5	25	42*	12	10	29	44	12	11	31	45
8	10	6	27	45*	13	11	31	46	13	12	33	48
9	11	7	29	48*	14	12	33	48	14	13	35	51
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

The similar tables have to be done for $e_3 = \{J_1, J_2, J_4\}$, $e_3 = \{J_1, J_3, J_4\}$ and $e_3 = \{J_2, J_3, J_4\}$ (see Table .A.5, A.6, A.7).

Table A.5: The third phase of an example with third jobs (2)

e_3	$\{J_1, J_2, J_4\}$											
J_j	J_1				J_2				J_4			
$ip_{1,j}$	12				6				9			
t	$C_j(t)$	t'	F_2	F_3	$C_j(t)$	t'	F_2	F_3	$C_j(t)$	t'	F_2	F_3
0	6	2	17	31	7	5	17	28*	6	3	17	29
1	6	2	17	31	7	5	17	28*	6	3	17	29
2	6	2	17	31	7	5	17	28	6	3	17	29
3	6	2	17	31	8	6	19	31	6	3	17	29*
4	6	2	17	31	9	7	21	34	7	4	17	30*
5	7	3	17	34*	10	8	23	37	8	5	19	33
6	8	4	19	35*	11	9	25	40	9	6	21	36
7	9	5	21	38*	12	10	27	43	10	7	23	39
8	10	6	23	41*	13	11	29	46	11	8	25	42
9	11	7	25	44*	14	12	31	49	12	9	27	45
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table A.6: The third phase of an example with third jobs (3)

e_3	$\{J_1, J_3, J_4\}$											
J_j	J_1				J_3				J_4			
$ip_{1,j}$	12				3				9			
t	$C_j(t)$	t'	F_2	F_3	$C_j(t)$	t'	F_2	F_3	$C_j(t)$	t'	F_2	F_3
0	6	2	17	31	6	5	17	25*	6	3	17	29
1	6	2	17	31	6	5	17	25*	6	3	17	29
2	6	2	17	31	7	6	19	28*	6	3	17	29
3	6	2	17	31	8	7	21	31	6	3	17	29*
4	6	2	17	31	9	8	23	34	7	4	17	30*
5	7	3	17	32*	10	9	25	37	8	5	19	33
6	8	4	19	34*	11	10	27	40	9	6	21	36
7	9	5	21	37*	12	11	29	43	10	7	23	39
8	10	6	23	40*	13	12	31	46	11	8	25	42
9	11	7	25	43*	14	13	33	49	12	9	27	45
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table A.7: The third phase of an example with third jobs (4)

e_3	$\{J_2, J_3, J_4\}$											
J_j	J_2				J_3				J_4			
$ip_{1,j}$	12				6				3			
t	$C_j(t)$	t'	F_2	F_3	$C_j(t)$	t'	F_2	F_3	$C_j(t)$	t'	F_2	F_3
0	7	2	17	25*	6	5	21	29	6	3	21	33
1	7	2	17	25*	6	5	21	29	6	3	21	33
2	7	2	17	25*	7	6	23	32	6	3	21	33
3	8	3	17	26*	8	7	25	35	6	3	21	33
4	9	4	19	29*	9	8	27	37	7	4	23	36
5	10	5	21	32*	10	9	29	40	8	5	25	39
6	11	6	23	35*	11	10	31	43	9	6	27	42
7	12	7	25	38*	12	11	33	46	10	7	29	45
8	13	8	27	41*	13	12	35	49	11	8	31	48
9	14	9	29	44*	14	13	37	52	12	9	33	51
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

At the end, we have the fourth phase with $e_4 = \{J_1, J_2, J_3, J_4\}$ (see Table. A.8) and only one possible value for $t = 0$.

We deduce from this table that the value of the optimal solution is equal to 41. By applying a backtrack algorithm, we found the the optimal solution is $\{J_3, J_1, J_4, J_2\}$.

Table A.8: The final phase of an example with four jobs

e_4	$\{J_1, J_2, J_3, J_4\}$															
J_j	J_1				J_2				J_3				J_4			
$ip_{1,j}$	16				8				4				12			
t	$C_j(t)$	t'	F_3	F_4	$C_j(t)$	t'	F_3	F_4	$C_j(t)$	t'	F_3	F_4	$C_j(t)$	t'	F_3	F_4
0	6	2	25	43	7	5	32	45	6	5	32	41^*	6	33	33	48

Appendix B

The results of GA

The time limit of the GA is fixed to $TimeLimGA = (n(m/2) \times 90)/1000$ seconds.

Initial population

The initial population P_0 individuals is generated by the ways: In GA_3 (case 3), two individual is given by EDD and NEH, two individual is given by $EBSR_{EDD}$ (individual is created by EBSR method that uses EDD rule as input) and $EBSR_{NEH}$, and others are randomly generated. In GA_4 (case 4), two individual is given by EDD and NEH, $5\% * PopSize$ individual is given by $EBSR_{EDD}$ and $5\% * PopSize$ individual is given by $EBSR_{NEH}$, and others are randomly generated. In GA_5 (case 5), two individual is given by EDD and NEH, $10\% * PopSize$ individual is given by $EBSR_{EDD}$ and $10\% * PopSize$ individual is given by $EBSR_{NEH}$, and others are randomly generated. These GA_3, GA_4, GA_5 are compared to $GA_1(EDD)$.

	$case3$	$case4$	$case5$
$PopSize = \|P_k\|$	150	250	150
$CrossSize = \|C_k\|$	200	300	600
$MutSize = \|M_k\|$	100	200	360

The several GA methods are compared in terms of quality. In Tables B.1, column 'Best' for 'GA_ν' ($\nu \in \{1, 3, 4, 5\}$) indicates the number of times the method GA_ν outperforms the other methods, column Cpu(s) indicates the average computation time of GA_ν per nine instances, column 'Δ_ν' indicates the average deviation between GA_ν and the best method between $GA_1(EDD)$, GA_3, GA_4 and GA_5.

$$\Delta_\nu = \frac{GA_\nu - \min(GA_1(EDD), GA_3, GA_4, GA_5)}{GA_\nu}$$

We can see in Table B.1, the genetic algorithm $GA_1(EDD)$ leads to the best results. The deviation between the solutions returned by this method and the best solutions is 2,63%. This value is around 5,14% for GA_3, 5,74% for GA_4 and 3,93% for GA_5.

Table B.1: Comparison of genetic algorithms of difference case

	$GA_1(EDD)$			GA_3			GA_4			GA_5		
$n \times m$	Best	Cpu(s)	Δ_1	Best	Cpu(s)	Δ_3	Best	Cpu(s)	Δ_4	Best	Cpu(s)	Δ_5
50 × 10	7	22,00	0,77%	3	22,01	3,44%	2	22,00	2,14%	3	22,00	1,14%
50 × 30	4	67,01	2,32%	1	67,01	3,51%	3	67,01	1,81%	1	67,00	1,56%
50 × 50	5	112,01	1,09%	1	112,01	2,13%	2	112,02	1,57%	1	112,01	2,85%
150 × 10	5	67,02	3,95%	5	67,02	4,56%	3	67,02	6,28%	2	67,02	4,93%
150 × 30	5	202,04	4,69%	2	202,01	8,41%	3	202,02	3,46%	2	202,02	1,25%
150 × 50	1	337,05	9,36%	3	337,04	3,06%	3	337,04	4,47%	2	337,03	6,23%
250 × 10	5	112,03	1,60%	6	112,04	0,87%	3	112,04	2,32%	4	112,03	1,40%
250 × 30	5	337,05	1,45%	4	337,05	5,01%	1	337,04	15,69%	3	337,05	12,66%
250 × 50	4	562,08	1,20%	4	562,06	1,98%	2	562,08	2,30%	2	562,06	1,94%
350 × 10	5	157,09	1,25%	2	157,08	8,32%	2	157,07	8,42%	6	157,06	10,65%
350 × 30	5	472,13	1,83%	2	472,07	15,72%	2	472,11	15,47%	4	472,08	1,54%
50 × 50	3	787,09	2,03%	2	787,13	4,71%	2	787,12	4,89%	5	787,09	0,99%
	54	269,55	2,63%	35	269,54	5,14%	28	269,55	5,74%	35	269,54	3,93%

Appendix C

The results of Tabu search algorithms

The for TS methods (with four tabu lists parameter and initiated solution by EDD rule) are compared in terms of quality. In Table C.1, column 'Best' for '$TS_\ell(EDD)$' (with ℓ is a number element of Tabu list) indicates the number of times the method $TS_\ell(EDD)$ outperforms the other methods, column Cpu(s) indicates the average computation time of $TS_\ell(EDD)$ per nine instances, column '$\Delta_\ell(EDD)$' ($\ell \in \{20, 40, 120\}$) indicates the average deviation between $TS_\ell(EDD)$ and the best method between $TS_{20}(EDD)$, $TS_{40}(EDD)$, $TS_{60}(EDD)$ and $TS_{120}(EDD)$.

$$\Delta_\ell(EDD) = \frac{TS_\ell(EDD) - \min(TS_{20}(EDD), TS_{40}(EDD), TS_{60}(EDD), TS_{120}(EDD))}{TS_\ell(EDD)}$$

In Table C.1, we can see that the $TS_{40}(EDD)$ leads to the best results. On average, the deviation between the solutions returned by this method and the best solutions is 1,26%. These values are around 2,75% for $TS_{20}(EDD)$, 2,88% for $TS_{60}(EDD)$ and 1,87% for $TS_{120}(EDD)$.

In Table C.2, the TS methods are initiated solution by EN rule. Column 'Best' for '$TS_\ell(EN)$' (with ℓ is a number element of Tabu list) indicates the number of times the method $TS_\ell(EN)$ outperforms the other methods, column Cpu(s) indicates the average computation time of $TS_\ell(EN)$ per nine instances, column '$\Delta_\ell(EN)$' ($\ell \in \{20, 40, 120\}$) indicates the average deviation between $TS_\ell(EN)$ and the best method between $TS_{20}(EN)$, $TS_{40}(EN)$, $TS_{60}(EN)$ and $TS_{120}(EN)$.

$$\Delta_\ell(EN) = \frac{TS_\ell(EN) - \min(TS_{20}(EN), TS_{40}(EN), TS_{60}(EN), TS_{120}(EN))}{TS_\ell(EN)}$$

As we can see that the $TS_{40}(EN)$ leads to the best results. On average, the deviation between the solutions returned by this method and the best solutions is 0,73%. These values are around 2,86% for $TS_{20}(EN)$, 2,56% for $TS_{60}(EN)$ and 2,33% for $TS_{120}(EN)$.

In Table C.3, the TS methods are initiated solution by NEH rule. Column 'Best' for '$TS_\ell(NEH)$' indicates the number of times the method $TS_\ell(NEH)$ outperforms the other methods, column Cpu(s) indicates the average computation time of $TS_\ell(NEH)$ per nine instances, column '$\Delta_\ell(NEH)$' ($\ell \in \{20, 40, 120\}$) indicates the average deviation between $TS_\ell(NEH)$ and the best method between $TS_{20}(NEH)$, $TS_{40}(NEH)$, $TS_{60}(NEH)$ and $TS_{120}(NEH)$.

$$\Delta_\ell(NEH) = \frac{TS_\ell(NEH) - \min(TS_{20}(NEH), TS_{40}(NEH), TS_{60}(NEH), TS_{120}(NEH))}{TS_\ell(NEH)}$$

We can see that the $TS_{40}(NEH)$ leads to the best results (number of the best solution =54). On average, the deviation between the solutions returned by this method and the best solutions is 2,17%. These values are around 2,35% for $TS_{20}(NEH)$, 0,58% for $TS_{60}(NEH)$ and 1,12% for $TS_{120}(NEH)$.

Table C.1: Comparison of tabu search algorithms are initiated by EDD solution

	$TS_{20}(EDD)$			$TS_{40}(EDD)$			$TS_{60}(EDD)$			$TS_{120}(EDD)$		
$n \times m$	Best	Cpu(s)	$\Delta_{20}(EDD)$	Best	Cpu(s)	$\Delta_{40}(EDD)$	Best	Cpu(s)	$\Delta_{60}(EDD)$	Best	Cpu(s)	$\Delta_{120}(EDD)$
50 × 10	3	22,02	1,96%	7	22,01	0,12%	3	22,01	1,57%	2	22,02	2,13%
50 × 30	1	67,03	1,30%	6	67,02	0,28%	2	67,04	1,49%	0	67,04	2,56%
50 × 50	0	112,07	1,57%	7	112,04	0,26%	1	112,06	1,22%	1	112,05	1,28%
150 × 10	3	67,28	4,55%	4	67,18	2,23%	4	67,18	0,54%	4	67,27	1,49%
150 × 30	3	202,89	1,58%	5	202,28	2,11%	1	202,89	6,96%	3	202,57	5,91%
150 × 50	2	338,62	8,36%	5	337,78	0,24%	1	338,04	7,91%	1	338,22	6,05%
250 × 10	4	113,21	0,58%	7	112,67	0,19%	3	113,50	0,79%	4	113,53	0,59%
250 × 30	2	341,34	1,41%	8	338,47	0,15%	2	340,95	0,89%	3	339,74	0,43%
250 × 50	1	568,16	1,30%	9	565,79	0,00%	1	567,71	0,80%	2	566,53	0,63%
350 × 10	3	160,81	0,83%	9	159,35	0,00%	3	159,79	1,54%	4	159,74	0,74%
350 × 30	1	480,60	9,09%	8	476,19	9,56%	2	478,17	10,29%	4	475,83	0,37%
350 × 50	2	801,11	0,47%	8	793,20	0,01%	1	798,64	0,55%	4	799,11	0,28%
	25	272,93	2,75%	83	271,17	1,26%	24	272,33	2,88%	32	271,97	1,87%

Table C.2: Comparison of tabu search algorithms are initiated by EN solution

	$TS_{20}(EN)$			$TS_{40}(EN)$			$TS_{60}(EN)$			$TS_{120}(EN)$		
$n \times m$	Best	Cpu(s)	$\Delta_{20}(EN)$	Best	Cpu(s)	$\Delta_{40}(EN)$	Best	Cpu(s)	$\Delta_{60}(EN)$	Best	Cpu(s)	$\Delta_{120}(EN)$
50 × 10	4	22,02	0,63%	4	22,01	0,33%	4	22,01	1,00%	3	22,00	0,49%
50 × 30	5	67,06	0,52%	1	67,02	1,30%	1	67,02	1,09%	2	67,03	1,42%
50 × 50	1	112,04	0,50%	2	112,03	0,49%	3	112,04	0,66%	3	112,06	0,70%
150 × 10	2	67,38	5,15%	5	67,19	0,29%	3	67,30	1,43%	5	67,20	1,42%
150 × 30	3	202,97	1,48%	5	202,30	5,12%	2	202,57	6,00%	2	202,62	4,12%
150 × 50	1	338,06	8,80%	3	337,51	0,65%	2	337,71	4,41%	3	338,18	5,93%
250 × 10	3	113,63	0,67%	5	112,89	0,08%	3	113,50	0,70%	7	112,74	0,13%
250 × 30	3	339,52	1,04%	5	338,11	0,11%	2	339,60	0,79%	5	339,38	0,37%
250 × 50	2	564,78	1,52%	6	564,31	0,12%	2	566,71	0,93%	2	566,30	0,77%
350 × 10	4	388,03	1,20%	8	158,26	0,14%	3	159,56	1,59%	7	158,28	0,15%
350 × 30	1	478,83	12,24%	9	476,06	0,00%	1	478,11	11,69%	1	478,27	11,72%
350 × 50	2	802,21	0,55%	8	795,12	0,11%	1	799,13	0,48%	1	901,60	0,75%
	31	291,38	2,86%	61	271,07	0,73%	27	272,11	2,56%	41	280,47	2,33%

Table C.3: Comparison of tabu search algorithms are initiated by NEH solution

	$TS_{20}(NEH)$			$TS_{40}(NEH)$			$TS_{60}(NEH)$			$TS_{120}(NEH)$		
$n \times m$	Best	Cpu(s)	$\Delta_{20}(NEH)$	Best	Cpu(s)	$\Delta_{40}(NEH)$	Best	Cpu(s)	$\Delta_{60}(NEH)$	Best	Cpu(s)	$\Delta_{120}(NEH)$
50 × 10	4	22,01	0,54%	5	22,01	0,35%	3	22,01	0,98%	3	22,01	0,58%
50 × 30	0	67,06	1,33%	7	67,03	0,19%	1	67,02	0,86%	1	67,01	1,47%
50 × 50	1	112,05	0,86%	3	112,02	0,61%	3	112,04	0,61%	2	112,03	0,69%
150 × 10	2	67,22	5,00%	4	67,15	4,31%	4	67,21	1,57%	5	67,12	0,57%
150 × 30	2	202,74	6,73%	2	202,90	7,47%	5	202,66	0,64%	3	202,50	2,32%
150 × 50	2	337,72	9,40%	2	412,21	10,34%	4	337,81	0,60%	1	337,56	6,16%
250 × 10	4	113,10	0,33%	6	112,62	0,47%	4	112,76	0,19%	4	112,81	0,07%
250 × 30	2	339,10	0,60%	3	338,63	0,62%	6	338,27	0,17%	4	339,06	0,45%
250 × 50	1	565,65	1,43%	5	564,99	0,58%	2	564,84	0,96%	4	564,21	0,78%
350 × 10	6	159,08	0,46%	5	159,41	0,28%	8	158,97	0,06%	7	158,74	0,03%
350 × 30	2	479,52	0,36%	6	475,51	0,19%	4	475,70	0,05%	4	474,68	0,08%
350 × 50	1	799,20	1,12%	6	795,23	0,65%	3	791,45	0,28%	3	794,40	0,24%
	27	272,04	2,35%	54	277,48	2,17%	47	270,89	0,58%	41	271,01	1,12%

Bibliography

[Adenso-Díaz, 1996] Adenso-Díaz, B. (1996). An sa/ts mixture algorithm for the scheduling tardiness problem. *European Journal of Operational Research*, 88:516–524.

[Armentano and Ronconi, 1999] Armentano, V. A. and Ronconi, D. P. (1999). Tabu search for total tardiness minimization in flowshop scheduling problems. *Computers and Operations Research*, 26:219–235.

[Bellman, 1957] Bellman, R. (1957). *Dynamic programming*. Princeton University Press.

[Ben-Daya and Al-Fawzan, 1998] Ben-Daya, M. and Al-Fawzan, M. (1998). A tabu search approach for the flow shop scheduling problem. *European Journal of Operational Research*, 109:88–95.

[Billaut, 2006] Billaut, J. C. (2006). New dominance conditions for the $f2 \parallel tbar$ problem. *Tenth International Workshop on Project Management and Scheduling (PMS'06)*, pages 78–82.

[Billaut and Zhang, 2007] Billaut, J. C. and Zhang, T. (2007). A branch and bound algorithm for the two-machine flow shop total tardiness problem. *International Conference on Industrial Engineering and Systems Management (IESM)*.

[Bixby et al., 2000] Bixby, R., Fenelon, M., Gu, Z., Rothberg, E., and Wunderling, R. (2000). Mip: Theory and practice - closing the gap. *In: System Modelling and Optimization: Methods, Theory, and Applications. Kluwer Academic Publishers, Boston*, Volume 174 of IFIP international federation for information processing.:19–49.

[Brucker, 2007] Brucker, P. (2007). *Scheduling Algorithms*. Springer Berlin Heidelberg New York.

[Campbell et al., 1970] Campbell, H. G., Dudek, R. A., and Smith, M. L. (1970). A heuristic algorithm for the n-job m-machine sequencing problem. *Management Science*, 16:169–174.

[Chakraborty and Laha, 2007] Chakraborty, U. K. and Laha, D. (2007). An improved heuristic for permutation flowshop scheduling. *Int. J. Information and Communication Technology*, 1:89–97.

[Chung et al., 2006] Chung, C. S., Flynn, J., and Kirca, O. (2006). A branch and bound algorithm to minimize the total tardiness for m-machine permutation flow shop problems. *European Journal of Operational Research*, 174:1–10.

[Della Croce et al., 2004] Della Croce, F., Ghirardi, M., and Tadei, R. (2004). Recovering beam search: enhancing the beam search approach for combinatorial optimization problems. *Journal of Heuristics*, 10:89–104.

[Della Croce et al., 2011] Della Croce, F., Grosso, A., and Salassa, F. (2011). A matheuristic approach for the total completion time two-machines permutation flow shop problem. *Lecture Notes in Computer Science, 6622 LNCS:*, pages 38–47.

[Della Croce and T'kindt, 2002] Della Croce, F. and T'kindt, V. (2002). A recovering beam search algorithm for the one-machine dynamic total completion time scheduling problem. *Journal of the Operational Research Society*, 53:1275–1280.

[Du and Leung, 1990] Du, J. and Leung, J. Y. T. (1990). Minimizing total tardiness on one machine is np-hard. *Mathematics of Operations Research*, 19:483–495.

[Fisher, 1976] Fisher, M. L. (1976). A dual algorithm for the one machine scheduling problem. *Mathematical Programming*, 11:229–251.

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

[Garey et al., 1976] Garey, M. R., Johnson, D., and Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of operations research*, 1:117–129.

[Garey and Johnson, 1979] Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman.

[Gelders and Sambandam, 1978] Gelders, L. F. and Sambandam, N. (1978). Four simple heuristics for scheduling a flowshop. *International Journal of Production Research*, 16:221–231.

[Glover, 1989] Glover, F. (1989). Tabu search - part i. *ORSA Journal on Computing*, 1:190–206.

[Glover, 1990] Glover, F. (1990). Tabu search - part ii. *ORSA Journal on Computing*, 2:4–32.

[Goldberg, 1989] Goldberg, D. E. (1989). *Genetic algorithms in Search, Optimization and Machine Learning.* Addison-Welsey, Reading Mass.

[Grabowski and Wodecki, 2001] Grabowski, J. and Wodecki, M. (2001). New block properties for the permutation flow-shop problem with application in ts. *Journal of the Operational Research Society*, 52:210–220.

[Grabowski and Wodecki, 2004] Grabowski, J. and Wodecki, M. (2004). A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion. *Computers and Operations Research*, 31:1891–1909.

[Graham et al., 1979] Graham, R. L., Lawler, E. L., Lenstra, J. K., and Rinnooy Kan, A. H. G. (1979). Optimization and approximation in deterministic suquencing and scheduling: a survey. *Proceedings of the Advanced Research Institute on Discrete Optimization*

and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium. Elsevier, 5:287–326.

[Gupta et al., 1993] Gupta, M. C., Gupta, Y., and Kumar, A. (1993). Minimising flow time variance in single machine using genetic algorithms. *European J ournal of Operational Research*, 70:289–303.

[Hasija and Rajendran, 2004] Hasija, S. and Rajendran, C. (2004). Scheduling in flowshops to minimize total tardiness of jobs. *International Journal of Production Research*, 42:2289–2301.

[Holland, 1975] Holland, J. A. (1975). *Adaptation in natural and artificial systems.* University of Michigan Press, Ann Arbor.

[Ignall and Schrage, 1965] Ignall, E. and Schrage, L. (1965). Application of the branch and bound technique to some flow shop scheduling problems. *Operations Research*, 13:400–412.

[Johnson, 1954] Johnson, S. M. (1954). Optimal two and three stage production schedules with setup times induced. *Nan.Res.Log.Quar*, 1:61–68.

[Kan, 1976] Kan, A. H. G. R. (1976). *Machine scheduling problem: Classification, complexity and computations.* MARTINUS NIJHOFF / THE HAGUE.

[Kim, 1993a] Kim, Y. D. (1993a). Heuristics for flowshop scheduling problems minimizing mean tardiness. *Journal of Operational Research Society*, 44:19–28.

[Kim, 1993b] Kim, Y. D. (1993b). A new branch and bound algorithm for minimizing mean tardiness in two-machine flowshops. *Computers and Operations Research*, 20(4):391–401.

[Kim, 1995] Kim, Y. D. (1995). Minimizing total tardiness in permutation flowshops. *European Journal of Operational Research*, 33:541–551.

[Koulamas, 1998] Koulamas, C. (1998). A guaranteed accuracy shifting bottleneck algorithm for the two-machine flowshop total tardiness problem. *Computers and Operations Research*, 25:83–89.

[Land and Doig, 1960] Land, A. H. and Doig, A. (1960). An automatic method of solving discrete programming problems. *JSTOR*, 28:497–520.

[Lenstra and Rinnooy Kan, 1981] Lenstra, J. K. and Rinnooy Kan, A. H. G. (1981). Complexity of vehicle routing and scheduling problems. *Networks*, 11:221–227.

[Lenstra et al., 1977] Lenstra, J. K., Rinnooy Kan, A. H. G., and Brucker, P. (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362.

[Leung, 2004] Leung, J. Y. T. (2004). *Handbook of scheduling: Algorithms, Models, and Performance Analysis.* Chapman and Hall/CRC, Boca Raton, Florida.

[Maniezzo et al., 2010] Maniezzo, V., Stutzle, T., and Voss, S. (2010). *Matheuristics: Hybridizing Metaheuristics and Mathematical Programming.* Annals of Information Systems 10, Springer.

[Montgomery, 2000] Montgomery, D. C. (2000). *Design and analysis of experiments.* 5th ed., New York: Wiley.

[Morton and Pentico, 1993] Morton, T. E. and Pentico, D. (1993). *Heuristic scheduling systems.* New York, John Wiley and Sons.

[Nawaz et al., 1983] Nawaz, M., Enscore, E., and Ham, I. (1983). A heuristic algorithm for the m-machine n-job flow shop sequencing problem. *Omega*, 11:91–95.

[Nowicki and Smutnicki, 1996] Nowicki, E. and Smutnicki, C. (1996). A fast tabu search algorithm for the permutation flowshop problem. *European Journal of Operational Research*, 91:160–175.

[Ogbu and Smith, 1990] Ogbu, F. A. and Smith, D. K. (1990). The application of the simulated annealing algorithm to the solution of the n/ m / cmax flow shop problem. *Computers & Operations Research*, 17:243–253.

[Onwubolu and Mutingi, 1999] Onwubolu, G. C. and Mutingi, M. (1999). Genetic algorithm for minimizing tardiness in flow-shop scheduling. *Production Planning and Control*, 10:462–471.

[Ow, 1985] Ow, P. S. (1985). Focused scheduling in proportionate flowshops. *Management Science*, 31:852–869.

[Ow and Morton, 1988] Ow, P. S. and Morton, T. E. (1988). Filtered beam search in scheduling. *International Journal of Production Research*, 11:91–6.

[Pan et al., 2002] Pan, J. C. H., Chen, J. S., and Chao, C. M. (2002). Minimizing tardiness in a two-machine flow-shop. *Computers and Operations Research*, 29:869–885.

[Pan and Fan, 1997] Pan, J. C. H. and Fan, E. T. (1997). Two-machine flowshop scheduling to minimize total tardiness. *International Journal of Systems Science*, 28 (4):405–414.

[Papadimitriou, 1994] Papadimitriou, C. (1994). *Computational Complexity.* Addison Wesley.

[Parthasarathy and Rajendran, 1998] Parthasarathy, S. and Rajendran, C. (1998). Scheduling to minimize mean tardiness and weighted mean tardiness in flowshop and flowline-based manufacturing cell. *Computers and Industrial Engineering*, 34:531–46.

[Pinedo, 1995] Pinedo, M. (1995). *Scheduling theory, algorithms, and system.* Prentice Hall.

[Pinedo, 2008] Pinedo, M. (2008). *Scheduling theory, algorithms, and system.* Prentice Hall.

[Portmann and Vignier, 2008] Portmann, M. C. and Vignier, A. (2008). *Chapter 4: Genetic algorithm and scheduling.* In Production Scheduling, P. Lopez and F. Roubellat Eds, Wiley-ISTE.

[Potts and Van Wassenhove, 1982] Potts, C. N. and Van Wassenhove, L. (1982). A decomposition algorithm for the single machine total tardiness problem. *Operations Research Letters*, 1:177–81.

[Raman, 1995] Raman, N. (1995). Minimum tardiness scheduling in flow shops: construction and evaluation of alternative solution approaches. *Journal of Operations Management*, 85:131–151.

[Reeves and Yamada, 1998] Reeves, C. R. and Yamada, T. (1998). Genetic algorithms, path relinking and the 'owshop sequencing problem. *Evolutionary Computation*, 6:45–60.

[Ruiz et al., 2006] Ruiz, R., Maroto, C., and Alcaraz, J. (2006). Two newrobust genetic algorithms for the flowshop scheduling problem. *Omega*, 34:461–476.

[Schaller, 2005] Schaller, J. (2005). Note on minimizing total tardiness in a two-machine flow shop. *Computers and Operations Research*, 32(12):3273–3281.

[Sen et al., 1989] Sen, T., Dileepan, P., and Gupta, J. N. D. (1989). The two-machine flowshop scheduling problem with total tardiness. *Computers and Operations Research*, 16(4):333–340.

[Ta et al., 2013a] Ta, Q. C., Billaut, J. C., and Bouquard, J. L. (2013a). An hybrid metaheuristic, an hybrid lower bound and a tabu search for the two-machine flowshop total tardiness problem. Proceedings of the 10th IEEE RIVF International Conference on Computing and Communication Technologies.

[Ta et al., 2013c] Ta, Q. C., Billaut, J. C., and Bouquard, J. L. (2013c). Recovering beam search and matheuristic algorithms for the $f2||\sum t_j$ scheduling problem. pages 218–220. Proceedings of the 11th Workshop on Models and Algorithms for Planning and Scheduling Problems (MAPSP).

[Ta et al., 2014a] Ta, Q. C., Billaut, J. C., and Bouquard, J. L. (2014a). Minimisation de la somme des retards pour un problème d'ordonnancement de type flowshop à deux machines et un problème de livraison intégrés. ROADEF.

[Ta et al., 2014b] Ta, Q. C., Billaut, J. C., and Bouquard, J. L. (2014b). Minimizing total tardiness in the m-machine flow-shop by genetic and matheuristic algorithms. *Journal of Intelligent Manufacturing*, page Published online: 05 February 2015.

[Ta et al., 2013b] Ta, Q. C., Billaut, J. C., and Bouquard, J. L. (février, 2013b). Minimisation de la somme des retards pour un problème de flow shop à deux machines. ROADEF.

[Ta et al., 2013d] Ta, Q. C., Gen, W., Billaut, J. C., and Bouquard, J. L. (2013d). Resolution of the $f2||\sum t_j$ scheduling problem by genetic algorithm and matheuristic. pages 408–412. Proceedings of 2013 International Conference on Industrial Engineering and Systems Management (IESM).

[Taillard, 1990] Taillard, E. (1990). Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47:65–74.

[Taillard, 1993] Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64:278–285.

[Talbi, 2013] Talbi, E. G. (2013). *Hybrid Metaheuristics*. Studies in computational intelligence, Springer.

[T'kindt and Billaut, 2006] T'kindt, V. and Billaut, J. C. (2006). *Multicriteria Scheduling: Theory, Models and Algorithms*. 2nd ed. Springer Berlin Heidelberg New York.

[T'kindt et al., 2003] T'kindt, V., Gupta, J. N. D., and Billaut, J. C. (2003). Two-machine flowshop scheduling with a secondary criterion. *Computers and Operations Research*, 30:505–526.

[Vallada and Ruiz, 2010] Vallada, E. and Ruiz, R. (2010). Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem. *European Journal of Operational Research*, 38:57–67.

[Vallada et al., 2008] Vallada, E., Ruiz, R., and Minella, G. (2008). Minimising total tardiness in the m-machine flowshop problem: A review and evaluation of heuristics and metaheuristics. *Computers & Operations Research*, 35:1350–1373.

[Vop et al., 1999] Vop, S., Martello, S., Osman, I. H., and Roucairol, C. (1999). *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Kluwer, Boston.

[Wagner, 1959] Wagner, R. (1959). An integer linear-programming model for machine scheduling. *Naval Research Logistics Quarterly*, 6(2):131–140.

[Wu et al., 2007] Wu, C. C., Lee, W. C., and Chen, T. (2007). Heuristic algorithms for solving the maximum lateness scheduling problem with learning considerations. *Computers & Industrial Engineering*, 52:124–132.

[Zegordi et al., 1995] Zegordi, S. H., Itoh, K., and Enkawa, T. (1995). Minimising makespan for flow-shop scheduling by combining simulated annealing with sequencing knowledge. *European Journal of Operational Research*, 85:515–531.