PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH
BLIDA SAAD DAHLAB UNIVERSITY
FACULTY OF SCIENCE
DEPARTMENT OF MATHEMATICS

MASTER THESIS IN OPERATIONAL RESEARCH

Theme:

SCHEDULING JOBS ON IDENTICAL MACHINES WITH AGREEMENT CONSTRAINTS

Author:

BENYOUCEF SIDOUMMOU

Defended before the jury composed of:

Mr. Brahim BENMEDJDOUB, B Senior lecturer at USTHB	President
Mr. Karim AMROUCHE, B Senior lecturer at Algiers 3 University	Director
Mr. Mohamed BENDRAOUCHE, A Senior lecturer at Blida 1 University	Co-director
Mrs. Wafaa LABBI, B Senior lecturer at Algiers 3 University	Examiner

Blida University, October 28th, 2018

ACKNOWLEDGEMENT

The success and final outcome of this project required a lot of guidance and assistance from many people and I am extremely privileged to have got this all along the completion of my project. All that I have done is only due to such assistance and I would not forget to thank them.

I would first like to thank my thesis director M. Karim Amrouche and my thesis co-director M. Mohamed Bendraouche for their support and advice along the completion of my project, and for their patience, motivation, enthusiasm, and immense knowledge.

Besides my director and my co-director, I would like to thank my thesis President, M. Brahim Benmedjdoub, and my thesis examiner, Mrs. Wafaa Labbi, for their insightful comments and hard questions.

I must express my very profound gratitude to my brother Mohamed, for providing me the opportunity to achieve this work. This accomplishment would not have been possible without him.

Nobody has been more important to me in the pursuit of this project than the members of my family. I would like to thank my parents, whose love and guidance are with me in whatever I pursue. They are the ultimate role models. Most importantly, I wish to thank my supportive wife, Fahima, and my three wonderful children, Mohamed Amine, Oussama and Yacine, who provide unending inspiration.

Last but not the least, I would like to thank my brothers, sisters, nephews, nieces and all my friends.

ABSTRACT

In this thesis, we consider the "Scheduling with Agreements" problem. The jobs are subjected to agreement constraints modeled by a graph called "Agreement Graph". All the jobs are ready at time zero, the preemption is not allowed and the objective is to minimize the makespan. We consider different types of agreement graph: general graphs, bipartite graphs, trees and chains. This problem is NP-hard in the general agreement graph case. However, it can be solved in a reasonable time for a special classes of graphs. Indeed, we introduce an algorithm solving polynomially the problem in the case of chain-type agreement graph. This algorithm is built on the maximum weight stable set to determine a lower bound for the problem. A mathematical formulation is proposed in both general and bipartite graphs. To approach the optimality and obtain an upper bound for the problem, we introduce heuristics based on lists. And, to be closer to the optimal solution, we propose metaheuristics. All these methods are widely experimented using uniformly-generated instances.

Keywords: *Scheduling with Agreements, Identical Parallel Machines, Makespan, Heuristics, Metaheuristics, Lower Bounds.*

خـــــــلاصة

في هذه الرسالة، نعتبر المسألة "جدولة مع التوافق"، أين تخضع الوظائف لقيود التوافق التي تمثل برسم بياني يسمى "الرسم البياني للتوافق ". جميع الوظائف جاهزة في بداية الجدولة، و لا يسمح بتقسيم الوظائف إلى أجزاء. والهدف هو التقليل إلى أدنى حد ممكن الوقت الإجمالي للجدولة. نعتبر أنواع مختلفة من الرسومات البيانية للتوافق: الرسوم البيانية العامة، الرسومات البيانية الثنائية الأجزاء، الأشجار و السلاسل. هذه المسألة صعبة الحل في حالة الرسم البياني العام للتوافق. ومع ذلك يمكن حلها في وقت معقول لفئات خاصة من الرسومات البيانية. لقد قدمنا خوارزمية حيث صعوبة الحل تكون على شكل كثير الحدود لحل المسألة في حالة الرسم البياني للتوافق على شكل سلسلة. بنيت هذه الخوارزمية على الوزن الأقصى للمجموعة المستقلة لتحديد الحد الأدنى للمسألة. ثم اقترحنا صيغة رياضية في كلتا الحالتين الرسومات البيانية العامة و الثنائية الأجزاء. و للحصول على نتائج أحسن في حل هذه المسألة، قدمنا مناهج تجريبية تستند إلى قوائم. ولكي نكون أقرب إلى الحل الأمثل، فقد اقترحنا مناهج تجريبية عليا. و تم اختبار كل هذه الطرق على نطاق واسع باستخدام أمثلة منشأة عشوائيا بشكل منتظم.

كلمات المفتاح: الجدولة مع التوافق ، الآلات المتوازية والمتطابقة، الوقت الإجمالي للجدولة ، المناهج للتجريبية ، المناهج التجريبية العليا، الحدود الدنيا.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

INTRODUCTION

Scheduling is a decision-making process that is used 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. The resources and tasks in an organization can take many different forms. The resources may be machines in a workshop, runways at an airport or processing units in a computing environment. The tasks may be operations in a production process, take-offs and landings at an airport or executions of computer programs. The objectives can also take many different forms. One objective may be the minimization of the completion time of the last task and another may be the minimization of the number of tasks completed after their respective due dates.

In a parallel-machine scheduling problem, we are given a set of jobs with associated processing times, and a set of identical machines, each of which can process at most one job at a time. The parallel-machine scheduling problem is to assign each job to exactly one machine so as to minimize the maximum completion time.

In a classical parallel-machine scheduling problem, the issue of a simultaneous job processing doesn't arise. However, in many cases, there exist constraints between jobs, called conflict constraints. This problem arise when certain jobs cannot be processed simultaneously on different machines because they share the same resources. This problem is called "Scheduling with Conflicts" or SWC, in short.

This problem has been introduced first by Baker and Coffman (1996) [1] as "Mutual Exclusion Scheduling". Even et al. [14] have studied later a problem called "Scheduling with Conflicts". In these problems, conflict between jobs is modeled by a graph, called "conflict graph", where two jobs related in this graph cannot be processed simultaneously on different machines.

In this thesis, we consider a problem equivalent to "Scheduling with Conflicts" but using the complement of the conflict graph, called "Agreement Graph", introduced by [Bendraouche and Boudhar, 2012][3].

This thesis is composed of six chapters. The first one is dedicated to notions on graph theory, complexity theory and combinatorial optimization, where graph notions used throughout this thesis (trees, rooted trees, chains, bipartite graphs, stable sets...), complexity notions and most common combinatorial problems are highlighted.

In the chapter 2, we introduce the scheduling problems including the parallel-machine problems. In the chapter 3, we define our problem with an example, and give some notations, motivations, a state of the art of the problem, some known previous results and a linear mathematical formulation of the problem for general and bipartite agreement graphs.

In chapter 4, lower bounds based on the maximum weight stable set in general and bipartite agreement graphs have been introduced. Two algorithms based on Greedy strategies to determine an approximate solution of the maximum weight stable set problem have been described and illustrated with 2 examples in a general graph case. A combinatorial algorithm of solving polynomially the problem of maximum weight stable set is detailed and applied to arbitrary bipartite graphs, trees and chains. A polynomial algorithm yielding an optimal schedule in the case of chain-type agreement graph has been described with a proof of its efficiency.

In section 1 of chapter 5, we introduce some heuristics, based on lists, to find a near-optimal solutions. And for approaching more the optimality, some local search methods and metaheuristics, a simulated annealing among them, are detailed under a form of algorithms in sections 2 and 3.

Finally, chapter 6 is dedicated to various experiments on heuristics, metaheuristics and mathematical formulations based on a uniformly generated instances. Tables and graphs showing some interesting results close this thesis.

Chapter 1

Notions on Graph Theory, Complexity and Combinatorial Optimization

I. Notions on Graph Theory

1. Basics

A graph G consists of two sets: a set $V = \{v_1, v_2, \dots, v_n\}$ whose elements are called vertices (or nodes) and a set $E = \{e_1, e_2, \dots, e_m\}$ whose elements are distinct pairs of vertices of G, called edges (or lines). The set V is called the vertex set of G and E is the edge set and the graph G is noted $G = (V, E)$.

Let $G = (V, E)$ be a graph and $e = \{u, v\}$ an edge of G. Since $\{u, v\}$ is 2-element set, we may write $\{v, u\}$ instead of $\{u, v\}$. It is often more convenient to represent the edge e by $e = uv$ or $e = vu$. The vertices u and v are called the extremities of the edge e. We further say that u and v are adjacent in G and that e joins u and v.

Every graph can be represented by a diagram where the vertices are represented by points or circles and the edges by lines.

Example: Let $G = (V, E)$ be a graph, where

$V = \{u_1, u_2, u_3, u_4, u_5, u_6\}$ is the vertex set and

$E = \{u_1, u_2\}, \{u_1, u_3\}, \{u_1, u_4\}, \{u_2, u_3\}, \{u_2, u_4\}, \{u_4, u_6\}$ is the edge set.

u_5 is an isolated vertex.

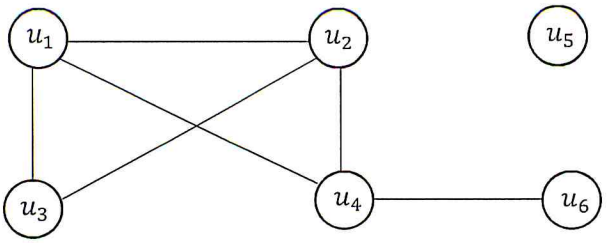


Fig. 1.1- Graph example

If $e = uv$ is an edge of a graph G, then the vertex u and the edge e are said to be incident with each other as are v and e. If two distinct edges say e and f are incident with a common vertex, then they are said to be adjacent edges.

A graph with p-vertices and q-edges is called a (p, q) graph. The $(1, 0)$ graph is called trivial graph. The complete graph K_n is the graph with n vertices and an edge joining every pair of vertices.

2. Degrees

Let $G = (V, E)$ be a graph. The number of edges incident to a vertex v of G is called the degree of a vertex v and is denoted by $d_G(v)$ (or $d(v)$). A vertex having no incident edges is called an isolated vertex (vertex of degree 0). A vertex of degree one, is called a pendent vertex or an end vertex.

The set of neighbors of a vertex v in G is denoted by $N_G(v)$.

An adjacency matrix for a simple graph G whose vertices are explicitly ordered $v_1, v_2, \ldots, v_n$ is the $n \times n$ matrix $A = (a_{ij})$ such that

$$a_{ij} = \begin{cases} 1 & \text{if } v_i \text{ and } v_j \text{ are adjacent} \\ 0 & \text{otherwise} \end{cases}$$

3. Complement of a graph and subgraph

Let $G = (V, E)$ be a graph. The complement of G is the graph, noted $\overline{G} = (V, \overline{E})$, such that $\{x, y\} \in \overline{E}$ if and only if $\{x, y\} \notin E$.

A graph $H = (W, F)$ is called a subgraph of a graph $G = (V, E)$ if $W \subseteq V$ and $F \subseteq E$. If $W = V$, H is called a spanning subgraph.

If $A \subseteq V$, the subgraph induced by A is the subgraph of G noted $G_A = (A, E_A)$, where E_A is the set of all edges having their extremities in A.

4. Chains and trees

In a graph, a chain (or path) of length k in G is a sequence P of distinct vertices of the form $P = \{x_0, x_1, \ldots, x_k\}$. The vertices x_1 and x_k are linked by P and are called its ends; the vertices $x_1, \ldots, x_{k-1}$ are the inner vertices of P (Fig. 1.2b).

We say that a graph $G = (V, E)$ is connected if for any two vertices $x, y \in V$, there is a chain in G joining the vertices x and y. A graph that is not connected is said to be disconnected and is composed of connected components.

A closed chain is a cycle. An acyclic graph is one with no cycles, also called a forest. A connected forest is called a tree (fig. 1.2a). Thus, a forest is a graph whose components are trees. The vertices of degree 1 in a tree are its leaves. A subtree of a graph is a subgraph which is itself a tree. If this tree is a spanning subgraph of the graph G, it is called a spanning tree of the graph G.

Example:

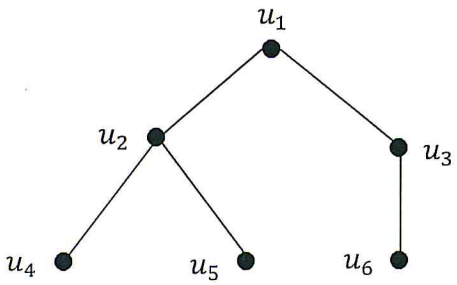


Fig.1.2a – Tree T (rooted at u_1)

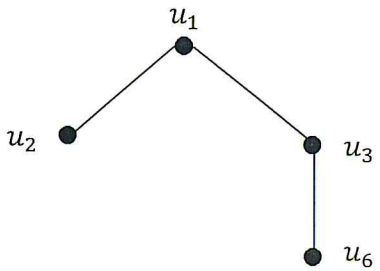


Fig. 1.2b – Chain in T

5. Rooted trees

A rooted tree is a tree with a designated vertex called the root. Designating a root imposes a hierarchy on the vertices of a rooted tree, according to their distance from that root (Fig. 1.2a). If vertex v immediately precedes vertex w on the path from the root to w, then v is parent of w and w is child of v. A leaf in a rooted tree is any vertex having no children. Figure 1.2-a represents a tree rooted at u_1. u_2 and u_3 are nodes of level 2. u_4, u_5 and u_6 are nodes of level 3 and are also leaves.

6. Stable set (or independent set)

A stable set (or independent set) in a graph is a set of vertices no two of which are adjacent. A stable set in a graph is maximum if the graph contains no larger stable set and maximal if the set cannot be extended to a larger stable set. The cardinality of a maximum stable set in a graph G is called the stability number of G and is denoted by $\alpha(G)$.

7. Bipartite graph

A bipartite graph is a graph $G = (V, E)$ whose node set V can be partitioned into two stable subsets S_1 and S_2. G is noted $G = (S_1, S_2; E)$.

If further all the vertices of S_1 are adjacent to all vertices of S_2, then G is called a complete bipartite graph. A bipartite graph cannot contain an odd cycle (a cycle of odd length). Forests and trees both are bipartite graphs.

To illustrate these concepts we introduce some decision problems which play an important role in proving that decision problems are NP-complete.

Partition: Given n integers $a_1, a_2, \ldots, a_n$ and the value $b = \frac{1}{2}\sum_{i=1}^{n} a_i$.

Is there a subset $I \subset \{1, \ldots, n\}$ such that $\sum_{i \in I} a_i = \sum_{i \notin I} a_i = b$?

Satisfiability: Given n Boolean variables, $U = \{u_1, u_2, \ldots, u_n\}$, and a set of m clauses, $C = \{c_1, c_2, \ldots, c_m\}$, or its complement (negation), is there an assignment of truth values to the boolean variables so that every clause is simultaneously true?

3-Dimensional Matching (3DM): We are given a set $N \subseteq W \times X \times Y$ where W, X, and Y are disjoint sets with the same number of q elements.

Does N contain a matching, that is, a subset $M \subseteq N$ with q elements such that no two elements of M agree in any coordinate?

III. Combinatorial Optimization Problem

Definition

The combinatorial optimization problem (COP in short) is defined by a given finite set of feasible solutions S and an objective function $f: S \to \mathbb{R}$, where the objective is to find an element x_0 from S verifying

$$f(x_0) = \min_{x \in S} f(x)$$

This problem is called a minimization problem, and noted $\begin{cases} Min\ f(x) \\ x \in s \end{cases}$

We can also define the maximization problem $\begin{cases} Max\ f(x) \\ x \in s \end{cases}$, which follows naturally from the minimization problem since we have

$$\max_{x \in S} f(x) = -\min_{x \in S}(-f(x))$$

A. Resolution by enumerative algorithms

1. Linear and integer programming formulations

The most basic mathematical program is the Linear Program (LP). The LP refers to an optimization problem in which the objective and the constraints are linear in the decision variables. It can be formulated as follows:

$$Minimize\ Z = cx$$
$$Ax \leq b$$
$$x \geq 0$$

An Integer Program (IP) is basically a linear program with the additional requirement that the variables have to be integers. If only a subset of the variables are required to be integer and the remaining ones are allowed to be real, the problem is referred to as a Mixed Integer Program (MIP). In contrast to the LP, an efficient (polynomial time) algorithm for the IP or MIP does not exist.

2. Dynamic programming

A dynamic programming algorithm has the property that each partial solution is represented by a state to which a value (or cost) is assigned. When two or more partial solutions achieve the same state, one with the lowest value is retained, while the others are eliminated. It is necessary to define the states in such a way that this type of elimination is valid.

3. Branch and bound

The process of solving a problem using a branch and bound algorithm can be conveniently represented by a search tree. Each node of the search tree corresponds to a subset of feasible solutions to a problem. A branching rule specifies how the feasible solutions at a node are partitioned into subsets, each corresponding to a descendant node of the search tree.

B. Resolution by near-optimal methods

The NP-hardness of an optimization problem suggests that it is not always possible to find an optimal solution quickly. Therefore, instead of searching for an optimal solution with enormous computational effort, we may instead use a local search method or an approximation algorithm to generate approximate solutions that are close to the optimum with considerably less investment in computational resources.

1. Pairwise interchange

The pairwise interchange (PI) is a permutation-based local search method used to generate a near-optimal solution of a scheduling problem. It operates by job interchanging on a pre-defined list.

2. Adjacent pairwise interchange

The adjacent pairwise interchange (API) method is the same as PI except that the jobs to be interchanged must be adjacent in the current permutation.

3. Insertion method

Insertion method is a variable neighborhood search method proceeding by inserting a randomly chosen position of a job in front (or back) of another randomly chosen job position.

4. Simulated annealing

In simulated annealing, a probabilistic acceptance rule is used. More precisely, any move that results in an improvement in the objective function value, or leaves the value unchanged, is accepted. On the other hand, a move that increases the objective function value by λ is accepted with probability $exp(-\lambda t)$, where t is a parameter known as the temperature. The value of t changes during the course of the search; typically t starts at a relatively high value and then gradually decreases.

Chapter 2

Scheduling Problems

1. The role of scheduling

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. The resources and tasks in an organization can take many different forms. The resources may be machines in a workshop, runways at an airport, crews at a construction site, processing units in a computing environment, and so on. The tasks may be operations in a production process, take-offs and landings at an airport, stages in a construction project, executions of computer programs, and so on.

2. Scheduling problems

The machine scheduling problems can be described as follows. There are m machines that are used to process n jobs. A schedule specifies, for each machine i ($i = 1, \ldots, m$) and each job j ($j = 1, \ldots, n$), one or more time intervals throughout which processing is performed on j by i. A schedule is feasible if there is no overlapping of time intervals corresponding to the same job (so that a job cannot be processed by two machines at once), or of time intervals corresponding to the same machine (so that a machine cannot process two jobs at the same time), and also if it satisfies various requirements relating to the specific problem type.

3. Machine environment

Different configurations of machines are possible. An operation refers to a specified period of processing by some machine type. We assume that all machines become available to process jobs at time zero.

In a single-stage production system, each job requires one operation, whereas in multi-stage systems the jobs require operations at different stages.

Single-stage systems involve either a single machine, or m machines operating in parallel, or dedicated machines. In the case of parallel machines, each machine has the same function. We consider three main cases:

- **Identical parallel machines:** in which each processing time is independent of the machine performing the job;
- **Uniform parallel machines:** in which the machines operate at different speeds;
- **Unrelated parallel machines**: in which the processing time of a job depends on the machine assignment.

Multi-stage systems comprise several stages, each having a different function. We distinguish three main cases:

- **Flow shop:** There are m machines in series. Each job has to be processed on each one of the m machines. All jobs have to follow the same route, i.e., they have to be processed first on machine 1, then on machine 2, and so on. After completion on one machine a job joins the queue at the next machine.
- **Job shop:** In a job shop with m machines each job has its own predetermined route to follow. A distinction is made between job shops in which each job visits each machine at most once and job shops in which a job may visit each machine more than once.
- **Open shop:** There are m machines. Each job has to be processed again on each one of the m machines. However, some of these processing times may be zero. There are no restrictions with regard to the routing of each job through the machine environment. The scheduler is allowed to determine a route for each job and different jobs may have different routes.

4. Jobs data

The processing requirements of each job j are given by the processing time p_j or p_{ij}. It depends on whether machines are identical or not.

In addition to its processing requirements, a job is characterized by its availability for processing, any dependence on other jobs, and whether interruptions in the processing of its operations are allowed. The availability of each job j may be restricted by its release date r_j that defines when it becomes available for processing,

and/or by its deadline $\tilde{d}_j$ that specifies the time by which it must be completed. Job dependence arises when there are precedence constraints on the jobs or when some jobs are not mutually in agreement, i.e. they cannot be processed at the same time on different machines. In general, all data p_i, p_{ij}, r_j, d_j, $\tilde{d}_j$, w_j are assumed to be integer.

5. Optimality criteria

For each job j, an integer due date d_j and a positive integer weight w_j may be specified.

Given a schedule, we can compute for job j: the completion time, the flow time, the lateness, the earliness, the tardiness, the unit penalty,...

Some commonly used optimality criteria involve the minimization of: the maximum completion time, or makespan C_{max}, the maximum lateness, the maximum cost, the maximum earliness, ...

6. Classification of scheduling problems

Scheduling problems are characterized by a 3-fields classification: α / ß / Υ [21]

a. Machine environment α

The machine environment is characterized by $\alpha = \alpha_1, \alpha_2$ of two parameters.

$\alpha_1 \in \{\emptyset, P, Q, R, O, J, F\}$, where $\emptyset$ denotes the empty symbol (thus, $\alpha = \alpha_2$ if $\alpha_1 = \emptyset$)

- $\emptyset$: single machine
- P: identical parallel machines
- Q: uniform parallel machines
- R: unrelated parallel machines
- O: open shop
- J: job shop
- F: flow shop

and $\alpha_2 \in \{k, \emptyset\}$

- k: fixed number of machines
- $\emptyset$: arbitrary number of machines

b. Jobs characteristics β

The job characteristics are specified by a set β containing at the most 8 elements

$\beta = \beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6, \beta_7, \beta_8$

- $\beta_1 \in (\emptyset, pmtn)$: indicates whether preemption (or job splitting) is allowed;
- $\beta_2 \in (\emptyset, res)$: there are or not specified resource constraints;
- $\beta_3 \in (\emptyset, prec, tree, chain)$: describes precedence relations between jobs. It denotes respectively independent tasks, general precedence constraints, precedence constraints forming a tree or a set of chains;
- $\beta_4 \in (\emptyset, r_i)$: If $\beta_4 = r_i$, then release dates may be specified for each job;

If β_4 does not appear in β, then $r_i = 0$ for all jobs;

- $\beta_5 \in (\emptyset, p_j = p, p \in P)$: specifies restrictions on the processing times or on the number of operations;
- $\beta_6 \in (\emptyset, \tilde{d})$: If $\beta_6 = \tilde{d}_i$, then a deadline $\tilde{d}_i$ is specified for each job J_i, i.e. job J_i must finish not later than time $\tilde{d}_i$;
- $\beta_7 \in$ (∅, s-batch, p-batch): indicates a batching problem. p-batching (s-patching) denotes that the length of a batch is equal to the maximum (sum) of processing times of all jobs in the batch;
- $\beta_8 \in$ (∅ , no-wait): Indicates whether a no-wait property is specified, for dedicated machines.

c. Optimality criteria Υ

We denote the completion time of job J_i by C_i, and the associated cost by $f_i(C_i)$.
There are essentially two types of total cost functions:

$$f_{max}(C) = \max_{1 \le i \le n}\{f_i(C_i)\} \qquad \text{and} \qquad \sum f_i(C) = \sum_{i=1}^{n} f_i(C_i)$$

The scheduling problem is to find a feasible schedule which minimizes the total cost function. The most common objective functions are:

- C_j : completion time of job j_j
- S_j : start time
- W_j : waiting time
- $F_j = C_j - r_j$: flow time
- $L_j = C_j - d_j$: lateness
- $T_j = Max\{C_j - d_j, 0\}$: tardiness
- $E_j = Max\{0, -L_j\}$: earliness
- $U_j = \{0 \; si \; C_j \le d_j \, , 1 \; otherwise\}$: unit penalty
- $C_{max} = Max\{C_j\}$: schedule length, makespan or completion time
- $\bar{F} = \frac{1}{n} \sum_{j=1}^{n} F_j$: mean flow time
- $\bar{F}_w = \frac{1}{n} \sum_{j=1}^{n} w_j F_j \, / \sum_{j=1}^{n} w_j$: weighted mean flow time
- $\sum_{j=1}^{n} w_i \, C_i$: total weighted completion time
- $T_{max} = Max\{T_j\}$: maximum tardiness
- $\sum_{j=1}^{n} T_j$: total tardiness
- $\sum_{j=1}^{n} w_j T_j$: total weighted tardiness
- L_{max} $\{L_i\}$: maximum lateness
- $\bar{T} = \frac{1}{n} \sum_{j=1}^{n} T_j$: mean tardiness
- $T_w = \frac{1}{n} \sum_{j=1}^{n} w_j T_j \, / \sum_{j=1}^{n} w_j$: weighted mean tardiness
- U = $\sum_{j=1}^{n} U_j$: number of tardy jobs

- $U_w = \sum_{j=1}^{n} w_j U_j$: weighted number of tardy jobs.

Example 1: $1|r_j, prec|\sum w_j C_j$ is the problem of scheduling jobs with release dates and precedence constraints on a single machine to minimize the total weighted completion time.

Example 2: $R|pmtn|L_{max}$ is the problem of preemptively scheduling jobs on an arbitrary number of unrelated parallel machines to minimize the maximum lateness.

Example 3: $P|AgreeG = (V, E)|C_{max}$ is the problem of scheduling jobs, related by an agreement graph, on identical parallel machines to minimize the makespan (which is the subject of our thesis).

7. Gantt chart

Gantt chart is a graphical presentation of a schedule of jobs on machines. X-axis chart represents time and rectangular block on y-axis represents machine. A horizontal bar shows each job's start and finish time on particular machine. The job number is inscribed in the rectangle. The length of the rectangle is scaled to represent job's processing time. The start and finish time of a job are indicated at the starting and terminating vertical sides of the job rectangle. Bars representing machines also indicate idle intervals on the machine.

Example

The following Gantt chart shows a schedule of 6 jobs processed on 3 parallel machines.

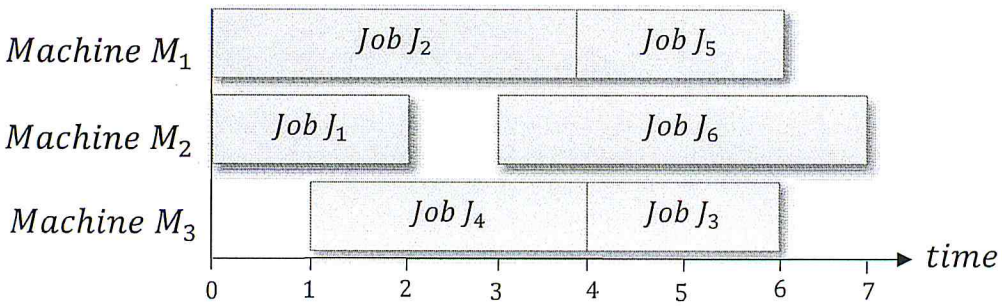


Fig 2.1. Gantt chart example

The makespan of the schedule is $C_{max} = 7$.

Chapter 3

State of the Art and Mathematical Model

1. Problem definition

We define the problem of Scheduling With Agreements (SWA) as follows [Bendraouche and Boudhar, 2012][3]. There are m identical parallel machines. There is a set V of n jobs with processing times $\{p_j\}_{j\in J}$. Agreement between jobs are derived from a graph $G = (V, E)$, called agreement graph, with vertex set $V = \{J_1, J_2, \dots, J_n\}$ and edge set E. Each edge in E models a pair of agreeing jobs that may be scheduled concurrently on different machines. A schedule is an assignment of time intervals on the m machines to the n jobs.

A schedule is feasible if each job J_j is assigned an interval of length p_j on one machine, intervals on the same machine do not overlap and intervals assigned to non-agreeing jobs (jobs not connected with an edge in the agreement graph) do not overlap. Furthermore, we suppose that all jobs are available for processing at time zero, and the preemption (or job splitting) is not allowed. The makespan of a schedule is the completion time of the most loaded machine. The objective is to find a feasible schedule that minimizes the makespan.

2. Notations

As a description of the problem, we use the three-field classification $\alpha|\beta|\gamma$ introduced by graham et al [21]. (See chapter 2).
So we adopt this notations:

The α field: $\alpha = \alpha_1, \alpha_2$

$\alpha_1 = P$, the shop has an identical parallel machines system

$\alpha_2 \in \{\emptyset, m\}$, machines number is arbitrary or fixed to m

The β field: $\beta = \beta_1, \beta_2, \beta_3$

$\beta_1 \in \{\emptyset, AgreeG = (V, E), AgreeG = (S_1, S_1, E)\}$, agreement graph is respectively complete, general or bipartite.

$\beta_2 \in \{\emptyset, r_i\}$, respectively, all jobs are ready at time zero or there is arbitrary ready dates for all jobs.

$\beta_3 \in \{\emptyset, "p_i = p", "p_i \in P"\}$, the processing times are respectively arbitrary, identical equal to p or from a set P.

The γ field: $\gamma = C_{max}$

The aim is to minimize the maximum completion time or makespan.

For instance, $Pm|\ AgreeG = (S_1, S_1, E)\}, p_i = 1|C_{max}$ is the problem with m identical parallel machines where jobs are related by a bipartite agreement graph, and the processing times are all equal to the unit.

According to these notations, our general problem would be expressed as:

$$P|AgreeG = (V, E)|C_{max}$$

which is NP-Hard since it contains, for a complete agreement graph, the well-known problem $P||C_{max}$ also known to be NP-hard.

3. Motivation

The SWA problem can be found in the resource constrained scheduling when the resources are non-sharable. The interest to this problem initially comes from the following problem: in a workshop there are n jobs $J_1, J_2, \ldots, J_n$ to be executed by m workers. Each job J_i requires a time p_i for its treatment and a subset of resources R_i from the set of the available resources $R = \{res_1, res_2, \ldots, res_k\}$. The objective is to execute these jobs in a minimum time. If we consider the workers as machines and associate the agreement graph $G = (V, E)$ in which $J = \{J_1, J_2, \ldots, J_n\}$ is the jobs set and that two jobs agree if they use no resources in common, one can verify that this problem can be modeled as the problem of scheduling with agreements in which the agreement graph is $G = (V, E)$.

Another motivation of this problem is the school exam planning: at school, there are n exams $E_1, \ldots, E_n$ to be scheduled at the end of the half year. Each exam has to be taken by a set of students and lasts either one or two hours. The exams are taken in classrooms whose number is m. Also we suppose that the capacity of a classroom is big enough so that any exam can be taken in any classroom. We seek a schedule which minimizes the length of the examination period. This problem can be modelized as follows: to each exam E_i corresponds a job J_i such that the processing time of J_i equals the time taken by its corresponding exam E_i. We regard the classrooms as being the machines. The agreement graph $G = (V, E)$ is such that $V = \{J_1, J_2, \ldots, J_n\}$ and a pair $\{J_i, J_j\}$ is an edge if and only if there are no students who take both exams E_i and E_j at a time. This problem can be formulated as the SWA problem in which the agreement graph is $G = (V, \mathrm{E})$ with processing times in $\{1,2\}$.

4. State of the art

When restricted to unit processing times, the SWA problem is equivalent to finding a partition of a given graph into a minimum number of cliques, each with size at most m. This problem is equivalent to the mutual exclusion scheduling problem introduced by Baker and Coffman (1996), by considering the complement of the agreement graph, called the conflict graph. Many results concerning this problem can be found in the literature (see, e.g. Bendraouche and Boudhar 2012[3]; Bodlaender and Jansen 1995[9]; Gardi 2009[17]; Hansen, Hertz, and Kuplinski 1993[24]).

Scheduling with agreement graphs or scheduling with agreements (SWA) is equivalent to scheduling with conflict graphs (the complement of the agreement graphs). The latter is known as scheduling with conflicts, which was first studied by Irani and Leung (1996)[24] and then by Chrobak et al. (2001). Even et al. (2009)[4] have also worked with conflict graphs and have considered the SWA problem for fixed m under the name: scheduling with conflicts (SWC in short).

In the case of two machines Even et al. (2009)[14] have proposed a polynomial time algorithm when the processing times are equal to 1 or 2 and have proved that it is APX-hard when $p_i \in \{1, 2, 3, 4\}$. Bendraouche and Boudhar (2012)[3] have established that the SWA problem is strongly NP-hard in the case of two machines when $p_i \in \{1, 2, 3\}$, even for arbitrary bipartite agreement graphs.

In a recent paper (Bendraouche, Boudhar and Oulamara 2015[5]), the authors have thoroughly closed the complexity status of SWA on two machines with two fixed processing times.

Example: We consider the *SWA* problem with 5 jobs and 2 machines.

Jobs processing times and agreement between jobs are depicted respectively in table 3.1 and figure 3.1 below.

Jobs J_j	J_1	J_2	J_3	J_4	J_5
Processing times p_j	2	4	1	3	2

Table 3.1 – Processing times of the example

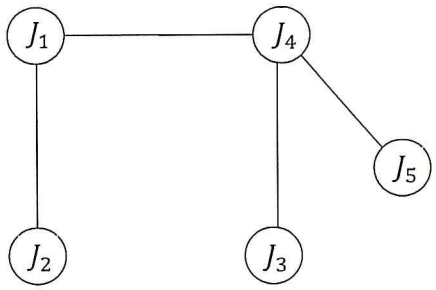


Fig. 3.1 – Agreement graph of the example

The Gantt chart of a feasible schedule of this problem is given in the figure 3.2 below.

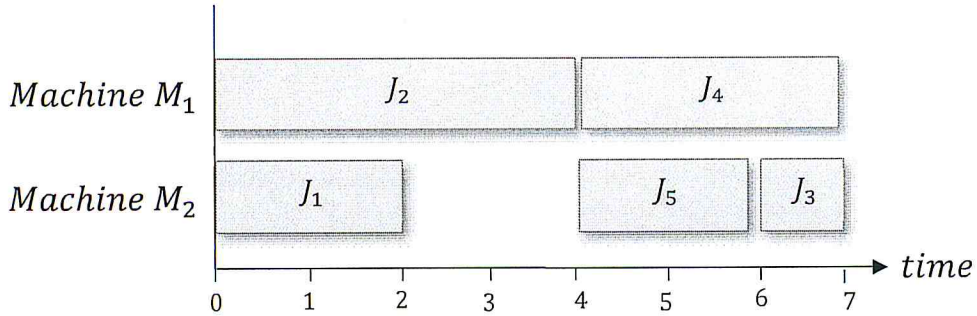


Fig. 3.2 - Gantt chart of the example

5. Some previous known results

Problem	Complexity	Reference
$P\|AgreeG = (V,E)\|C_{max}$	NP-hard	Garey & Johnson (1975) [19]
$P2\|AgreeG = (V,E), p_i \in \{1,2\}\|C_{max}$	Polynomial	Even et al. (2006) [15]
$P2\|AgreeG = (V,E), r_i, p_i = 1\|C_{max}$	NP-hard	Boudhar & Finke (2000) [18]
$P2\|AgreeG = (V,E), r_i \in \{0,r\}, p_i \in \{1,2\}\|C_{max}$	NP-hard	Bendraouche & Boudhar (2012) [3]
$P2\|AgreeG = (S_1,S_2;E), p_i \in \{1,2,3\}\|C_{max}$	NP-hard	Bendraouche & Boudhar (2012) [3]
$P2\|AgreeG = (S_1,S_2;E), r_i \in \{0,r\}\ p_i \in \{1,2\}\|C_{max}$	NP-hard	Bendraouche & Boudhar (2012) [3]
$P2\|AgreeG = (S_1,S_2;E), p_{S_1} = 1\|C_{max}$	Polynomial	Bendraouche & Boudhar (2012) [3]
$P2\|AgreeG = (S_1,S_2;E), p_i \in \{a, 2a+b\}\|C_{max}$, b≠ 0	NP-hard	Bendraouche et al. 2015 [5]
$P2\|AgreeG = (S_1,S_2;\text{tree})\|C_{max}$	Open	
$P2\|AgreeG = (S_1,S_2;\text{binary tree})\|C_{max}$	Open	

Table 3.2 – Some previous known results

6. Mathematical model

A. Case of general agreement graph

We recall the scheduling with agreements problem $P|AgreeG = (V,E)|C_{max}$.

$V = \{J_1, J_2, \dots, J_n\}$ is a set of n jobs with processing times $p_1, p_2, \dots, p_n$ to be processed on m identical parallel machines $M_1, M_2, \dots, M_m$. Let $G = (V,E)$ be a graph, called agreement graph, with vertex set V and edge set E. Each edge in E models a pair of jobs mutually agreeing that may be scheduled concurrently on different machines.

Let $A = (a_{ij})$, $i = \overline{1,n}$, $j = \overline{1,n}$, be the adjacency matrix associated with the graph G.

We define the two matrices of bivalent variables as follows.

Let $X = (x_{jk})$, $j = \overline{1,n}$, $k = \overline{1,m}$, be the jobs affectation matrix.

Let $e = (e_{ij})$, $i,j = \overline{1,n}$, $i \neq j$, be the jobs precedence matrix.

The decision variables are:

$$x_{jk} = \begin{cases} 1 & \text{if job } J_j \text{ is processed on machine } M_k \\ 0 & \text{otherwise} \end{cases}$$

t_i, $i = \overline{1,n}$, is the start time of job J_i

$$e_{ij} = \begin{cases} 1 & \text{if } t_i \leq t_j \\ 0 & \text{otherwise} \end{cases}$$

Since each job J_i is processed on exactly one machine, we have

$\sum_{k=1}^{m} x_{ik} = 1$

Let $M = \sum_{i=1}^{n} p_i$ be the sum of the processing times of all jobs.

The equation involving e_{ij} can be given as

$t_j - t_i \leq M.e_{ij}, \qquad i,j = \overline{1,n},\ i \neq j$

If job J_i and job J_j are processed on the same machine M_k, and J_i is processed before J_j, then the processing time intervals of the two jobs do not overlap. This can be expressed as $t_j \geq t_i + p_i$, $i,j = \overline{1,n}$, $i \neq j$

In other words, if $x_{ik} = 1$ and $x_{jk} = 1$, then $t_j \geq t_i + p_i$

The constraints guaranteeing that the jobs intervals do not overlap are given by

$t_i + p_i - t_j \leq M\left(1 - e_{ij} + 2 - x_{ik} - x_{jk}\right)$ $i,j = \overline{1,n}, i \neq j, k = \overline{1,m}$

For all pair of jobs $\{j_i, j_j\}$ which are not linked in the agreement graph, we have the agreement constraints

$t_j - t_i \leq p_i$ or $t_i - t_j \leq p_j$, $\qquad \forall\, i,j \mid (j_i, j_j) \notin E,\ i \neq j$

And using the adjacency matrix, these constraints would be

$t_j - t_i \geq (1 - a_{ij}).p_i$ or $t_i - t_j \geq (1 - a_{ij}).p_j, \qquad i,j = \overline{1,n},\ i \neq j$

These disjunctive equations could be expressed in a conjunctive form as follows

$t_j - t_i \geq [1 - a_{ij}(2 - x_{ik} - x_{jk})]p_i + (e_{ij} - 1)M \qquad i,j = \overline{1,n}, i \neq j, k = \overline{1,m}$

$t_i - t_j \geq [1 - a_{ij}(2 - x_{ik} - x_{jk})]p_j - e_{ij}M \qquad i,j = \overline{1,n}, i \neq j, k = \overline{1,m}$

Finally, the problem can be formulated as follows.

Formulation ($F1$)

$$
\begin{cases}
Min\ Z = C_{max} & & \\
\sum_{k=1}^{m} x_{ik} = 1 & i = \overline{1,n} & (1)\\
t_j - t_i \geq [1 - a_{ij}(2 - x_{ik} - x_{jk})]p_i + (e_{ij} - 1)M & i,j = \overline{1,n}, i \neq j, k = \overline{1,m} & (2)\\
t_i - t_j \geq [1 - a_{ij}(2 - x_{ik} - x_{jk})]p_j - e_{ij}M & i,j = \overline{1,n}, i \neq j, k = \overline{1,m} & (3)\\
t_i + p_i \leq C_{max} & i = \overline{1,n} & (4)\\
x_{ik} \in \{0,1\} & i = \overline{1,n}, k = \overline{1,m} & (5)\\
e_{ij} \in \{0,1\} & i,j = \overline{1,n}, i \neq j & (6)\\
t_i \geq 0 & i = \overline{1,n} & (7)\\
C_{max} \geq 0 & & (8)
\end{cases}
$$

B. Case of bipartite agreement graph and two machines

Let $G = (S_1, S_2; E)$ be a bipartite graph. Let p and q be respectively the cardinality of its two partitions S_1 and S_2. We have

$A = \begin{bmatrix} C & B \\ E & D \end{bmatrix}$, where

$B = (b_{ij})$, $i \in S_1, j \in S_2$; $C = 0$, $i \in S_1, j \in S_1$; $D = 0$, $i \in S_2, j \in S_2$; $E = B^t$, $i \in S_2, j \in S_1$.

A is the adjacency matrix of the general graph formulation ($F1$).

The formulation in the case of bipartite graph is based essentially on the reduced adjacency $p \times q - matrix\ B = (b_{ij})$, $i = \overline{1,p}$, $j = \overline{1,q}$, with $p + q = n$.

Replacing the matrix A by respectively B, C and D in the equations (2) and (3) in the general graph formulation ($F1$), we obtain the new formulation of the problem ($F2$)

Formulation ($F2$)

$$
\begin{cases}
Min\ Z = C_{max} & & \\
x_{i1} + x_{i2} = 1 & i = \overline{1,n} & (1)\\
t_j - t_i \geq [1 - b_{ij}(2 - x_{ik} - x_{jk})]p_i + M(e_{ij} - 1) & i \in S_1, j \in S_2, k \in \{1,2\} & (2)\\
t_i - t_j \geq [1 - b_{ij}(2 - x_{ik} - x_{jk})]p_j - M.e_{ij} & i \in S_1, j \in S_2, k \in \{1,2\} & (3)\\
t_j - t_i \geq p_i + M(e_{ij} - 1) & i,j \in S_1, i \neq j & (4)\\
t_i - t_j \geq p_j - M.e_{ij} & i,j \in S_1, i \neq j & (5)\\
t_j - t_i \geq p_i + M(e_{ij} - 1) & i,j \in S_2, i \neq j & (6)\\
t_i - t_j \geq p_j - M.e_{ij} & i,j \in S_2, i \neq j & (7)\\
t_i + p_i \leq C_{max} & i = \overline{1,n} & (8)\\
x_{ik} \in \{0,1\} & i = \overline{1,n}, k \in \{1,2\} & (9)\\
e_{ij} \in \{0,1\} & i,j = \overline{1,n}, i \neq j & (10)\\
t_i \geq 0 & i = \overline{1,n} & (11)\\
C_{max} \geq 0 & & (12)
\end{cases}
$$

Chapter 4
Lower Bounds

Maximum weight stable set problem (MWS)

An independent set S in a graph (called also a stable set) is a subset of vertices no two of which are adjacent. (For example, both subsets S_1 and S_2 of the node partitions of a bipartite graph are stable sets). The maximum weight stable set problem, also known as the vertex packing problem, deals with graphs whose vertices are weighted with positive integers, the problem being to find a stable set of maximum total weight. Given a weight function $c : V \to R_+$, we seek to maximize the total weight $c(S) = \sum_{i \in S} p_i$ over the collection of independent sets S.

We consider the following scheduling with agreements problem

$$P|AgreeG = (V, E)|C_{max}$$

If we regard the processing times p_j of the jobs J_j as the node weights in the agreement graph $G = (V, E)$, G will be a weighted graph noted $G = (V, E, p)$,

where $p = (p_1, p_2, \dots, p_n)$ is the processing times vector.

Since the nodes of an independent set are not adjacent mutually, the corresponding jobs cannot be processed simultaneously on different machines in any feasible schedule. Thus, the total weight of an independent set would be a lower bound for the makespan.

A. Case of general agreement graph

The problem of maximum weight stable (MWS) is well known to be NP-Hard in a general graph type. Sakai et al. [30] have provided 2 simple algorithms WGMIN and WGMIN2 based on Greedy strategies to determine a near-optimal solution of the problem. However, this problem can be solved in a polynomial time for some special classes of graphs.

It should be noted that, if $p_1 = p_2 = \ldots = p_n = 1$, we meet the Maximum Independent Set (MIS) problem which is also known to be NP-Hard.

1. Algorithm: GWMIN

Algorithm: GWMIN
Input: Weighted graph $G = (V, E, p)$
Output: Approximation of MWS S in G

1. set $S := \emptyset$; $H := G$;
2. for each vertex u in H, set $c(u) := p(u)/[d_H(u) + 1]$;
3. select a vertex v in $H \mid c(v) = \max_{u \in H} c(u)$;
4. set $S := S \cup \{v\}$; $H := H[V(H) - N_H^+(v)]$;
5. if $V(H) \neq \emptyset$, return to step 2;
   otherwise, STOP: return S.

$d_H(u)$ denote the degree of vertex u in the graph H and $N_H^+(v)$ the close neighborhood of the vertex v in the graph H.

Example: Consider the following weighted graph G

Node i	1	2	3	4	5	6
Weight $p(i)$	3	4	5	1	1	4

Table 4.1 – Node weights of the example

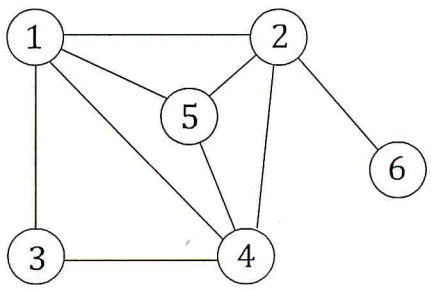


Fig. 4.1 – Graph of the example

Initialization: $S := \emptyset$; $H := G$;
Iteration 1:
Step 2: for each vertex u in H, calculate
$c(1) = p(1)/[d_H(1) + 1] = 3/(4 + 1) = 3/5$
$c(2) = 4/(4 + 1) = 4/5$
$c(3) = 5/(2 + 1) = 5/3$

$c(4) = 1/(4+1) = 1/5$

$c(5) = 1/(3+1) = 1/4$

$c(6) = 4/(1+1) = 2$

$c(v) = max\left(\frac{3}{5}, \frac{4}{5}, \frac{5}{3}, \frac{1}{5}, \frac{1}{4}, 2\right) = 2;\ v = 6;$

$H = H(V - \{6,2\}) = H(\{1,3,4,5\});\ S = \{6\}$

Iteration 2:

for each vertex u in H, calculate

$c(1) = p(1)/[d_H(1)+1] = 3/(3+1) = 3/4$

$c(3) = 5/(2+1) = 5/3$

$c(4) = 1/(3+1) = 1/4$

$c(5) = 1/(2+1) = 1/3$

$c(v) = max(\frac{3}{4}, \frac{5}{3}, \frac{1}{4}, \frac{1}{3}) = 5/3, ;\ v = 3;$

$H = H(V - \{3,4,1\}) = H(\{5\});\ S = \{6,3\}$

Iteration 3:

The graph H is reduced to 1 vertex, that is vertex 5, so we add it to S.

We obtain then the stable set $S = \{6,3,5\}$ of total weight $c(S) = 10$.

2. Algorithm: GWMIN2

Algorithm: GWMIN2
Input: Weighted graph G
Output: Approximation of MWS S in G

1. $S := \emptyset; H := G;$
2. for each vertex u in H, set $c(u) := p(u)/\sum_{w \in N_H^+(u)} p(w);$
3. select a vertex v *in* $H \mid c(v) = \max_{u \in H} c(u);$
4. set $S := S \cup \{v\};\ H := H[V(H) - N_H^+(v)];$
5. if $V(H) \neq \emptyset$, return to step 2;
   otherwise, STOP: return S.

Example

Let's apply GWMIN2 algorithm to the same example as for GWMIN algorithm, and compare the 2 lower bounds obtained.

$S := \emptyset; H := G;$

Iteration 1:

Step 2: for each vertex u in H, calculate $c(u) := p(u)/\sum_{w \in N_H^+(u)} p(w)$

$c(1) = p(1)/(p(1) + p(2) + p(3) + p(4) + p(5)) = 3/(3+4+5+1+1+4) = 3/15 = 1/5$

$c(2) = p(2)/(p(1) + p(2) + p(4) + p(5) + p(6) = 4/13$

$c(3) = p(3)/((p(1) + p(3) + p(4)) = 1/3$

$c(4) = p(3)/((p(1) + +p(2) + p(3) + p(4) + p(5)) = 1/14$

$c(5) = p(5)/((p(1) + +p(2) + p(4) + p(5)) = 1/9$

$c(6) = 4/(4 + 4) = 1/2$

$c(v) = max\left(\frac{1}{5}, \frac{4}{13}, \frac{1}{3}, \frac{1}{14}, \frac{1}{9}, \frac{1}{2}\right) = 1/2; v = 6$

$H = H(V - \{6, 2\}) = H(\{1, 3, 4, 5\}; S = \{6\}$

Iteration 2:

$c(1) = p(1)/(p(1) + p(3) + p(4) + p(5)) = c(3) = 3/10$

$c(3) = p(3)/(p(1) + p(3) + p(4)) = 5/9$

$c(4) = p(4)/(p(1) + p(3) + p(4) + p(5)) = 1/10$

$c(5) = p(5)/(p(1) + p(4) + p(5)) = 1/5$

$c(v) = max\left(\frac{3}{10}, \frac{5}{9}, \frac{1}{10}, \frac{1}{5}\right) = \frac{5}{9}; v = 3$

$H = H(V - \{3, 4, 1\}) = H(\{5\}); S = \{6, 3\}$

Iteration 3:

The graph H is reduced to 1 vertex, that is vertex 5, we add it then to S.

So we obtain the stable set $S = \{6, 3, 5\}$ of total weight $c(S) = 10$.

Note that the two algorithms give the same value of the maximum weight stable set.

3. Lower bounds for a general agreement graph

Let LB_0 be the classical lower bound of the makespan for the parallel machines problem $P||C_{max}$, that is

$$LB_0 = \left\lceil max\left(\sum_{i=1}^{n} p_i/m, \ \max_{1 \le i \le n} p_i\right)\right\rceil,$$

where m denotes the machines number.

Since $P|AgreeG = (V, E)|C_{max}$ contains the $P||C_{max}$ problem (complete agreement graph), then LB_0 could be considered as a lower bound for our problem.

Let LB_1 and LB_2 be the lower bounds yielded respectively by WGMIN and WGMIN2 algorithms. The overall lower bound would be

$$LB = max\{LB_0, LB_1, LB_2\};$$

Continuing with the example above, we have

GWMIN algorithm gives the first lower bound $LB_1 = 10$.

GWMIN2 algorithm gives the second lower bound $LB_2 = 10$.

Now, let's compute the classical lower bound of $P||C_{max}$ problem

$LB_0 = \left\lceil max\left(\sum_{i=1}^{n} p_i/n, \ \max_{1 \le i \le n} p_i\right)\right\rceil = \lceil max(18/2, \ 5)\rceil = 9.$

Hence, the overall lower bound is

$$LB = max(LB_0, LB_1, LB_2) = max(9, 10, 10) = 10.$$

B. Case of bipartite agreement graph and two machines

Faigle et al. [15] have provided a combinatorial algorithm to solve the problem of maximum weight stable set in a bipartite graph in polynomial time. They proved that the complexity of the algorithm is $O(n^4)$ (where n denotes the nodes number of the graph). In this chapter, we present the algorithm remodeled in an explicit and structured way, and give an example of *SWA* problem, with 2 machines and a bipartite agreement graph, where the resolution of the problem of maximum weight stable set is considered to compute a lower bound for the *SWA* problem.

1. Maximum weight stable set problem in bipartite graphs

The algorithm we present in this chapter for the resolution of the maximum weight stable set problem in bipartite graphs is a combinatorial algorithm [15]. It proceeds as follows. We consider a spanning tree in the graph and determine a maximal stable set relative to this tree. If this solution is not stable in the original graph, we find a restricting edge which has not yet been considered during the computation, add it to the tree, delete another edge to obtain a new tree and compute a solution in the modified tree. The algorithm will end with a solution that is optimal.

Two algorithms considered as subroutines will be used in the main algorithm: the one for finding a spanning tree in a graph, and the other for determining a maximum weight stable set in a tree.

2. Spanning tree algorithm

Algorithm: ST (Spanning Tree)
Input: Graph $G = (V, E)$
Output: Spanning tree $T = (V', E')$

1. fix an arbitrary node, say 1;
2. set $S := \{1\}$, $V' := \{1\}$, $E' := \emptyset$;
3. for each x in S, in order
   for each y in $V \backslash V'$
   if $(x, y) \in E$, then $E' := E' \cup (x, y)$, $V' := V' \cup \{y\}$;
   next y
   if no edges were added, STOP: return T;
   next x
4. $S :=$ children of S; return to step 3.

A spanning tree is a tree covering all nodes. There are 2 methods for determining a spanning tree in a connected graph: Breadth-First Search and Depth-First search. The Breadth-First Search algorithm proceeds as follows. Start with arbitrarily chosen vertex of the graph as the root, add all edges incident to the vertex along with the other vertex connected to each edge. The new vertices added become the vertices

at level 1 in the spanning tree. For each vertex at level 1, visited in order, add each edge incident to this vertex and the other vertex connected to the edge to the tree as long as it does not produce a simple circuit. The new vertices added become the new vertices at level 2 in the spanning tree. Repeat the procedure until all vertices in the graph have been added.

3. MWST Algorithm (maximum weight stable set in trees).

MWST algorithm uses a dynamic programming approach to determine a maximum weight stable set in a tree T. It proceeds as follows.

We fix an arbitrary node r in T. Then, for each $v \in T$ and the subtree T_v rooted at v, we compute:

- $\omega(v)$: Total weight $c(S_v)$ of a maximum weight stable set S_v in T_v.
- $\omega^-(v)$: Total weight $c(S_v)$ of a maximum weight stable set S_v^- in T_v under the restriction $v \notin S_v$.
- $\omega^+(v)$: Total weight $c(S_v)$ of a maximum weight stable set S_v^+ in T_v under the restriction $v \in S_v$.

A maximum weight stable set S_v either contains v or does not contain v, Therefore,

$$\omega(v) = max\{\omega^+(v), \omega^-(v)\} \quad (1)$$

A maximum weight stable set S_v in T_v under the restriction $v \notin S_v$ decomposes into maximum weight stable sets in the subtrees of the sons of v. Let $S(v)$ be the set of the sons of v in the tree T. So we have

$$\omega^-(v) = \sum_{w \in S(v)} \omega(w) \quad (2)$$

A maximum weight stable set S_v in T_v under the restriction $v \in S_v$ decomposes into v itself and maximum weight stable sets in the subtrees of the sons of v under the restriction that the sons themselves are not part of the set. Hence

$$\omega^+(v) = p(v) + \sum_{w \in S(v)} \omega^-(w) \quad (3)$$

Using recursively the equations 1, 2 and 3, we can compute the values $\omega(v)$, $\omega^+(v)$ and $\omega^-(v)$ for all nodes form the leaves to the root.

To construct the solution S in T, we first initialize $S := \emptyset$ and proceed from the root to the leaves as follows. For each node v, we perform the test:

If $\omega^+(v) < \omega^-(v)$, there is no optimal solution containing the node v. Then we do not add v to S.

If $\omega^+(v) > \omega^-(v)$, the optimal solution must include the node v. So we add v to S.

If $\omega^+(v) = \omega^-(v)$, there are some maximum weight solutions which contain the node v and some which do not. So we can equally add it to S or not.

In the situation where we have added the node v to S, its sons cannot belong to S. So we continue the construction of the optimal stable set S with the grandchildren of v.

In the algorithm, the Boolean variable $\varphi(v)$ guarantee that for a node v belonging to a stable set S, its sons cannot be part of this stable set. For that it takes the false value. φ is initialized to true for all nodes.

Algorithm: MWST (Maximum weight stable set in a tree)
Input: Tree T
Output: Maximum weight stable set S

a. fix an arbitrary node r as the root of the tree T;
b. for each node v in T, set $\varphi(v) := true$;
   set $S := \emptyset$, $c(S) := 0$;
c. from the leaves up to the root, and for each node v, set
   $\omega^+(v) := p(v) + \sum_{w \in S(v)} \omega^-(w)$;
   $\omega^-(v) := \sum_{w \in S(u)} \omega(w)$;
   $\omega(v) := \max\{\omega^+(v), \omega^-(v)\}$;
d. from the root down to the leaves, and for each node v, do
   if $\varphi(v) = true$ and $\omega^+(v) \geq \omega^-(v)$ then
     $S := S \cup \{v\}$, $c(S) := c(S) + p(v)$;
     for each $w \in S(v)$, set $\varphi(w) := false$;
   end if
e. end. Return S.

4. Maximum weight stable set algorithm in arbitrary bipartite graphs

Let $G = (S_1, S_2; E)$ be a bipartite graph.

Let $S(v)$ denote the set of the sons of node v in a tree.

We assume that the bipartite graph G is connected (as otherwise the problem decomposes naturally into subproblems on the respective connected components).

The algorithm proceeds conceptually as follows:

1. compute a spanning tree T in G using ST algorithm;
2. find a maximum weight stable set S in T using MWST algorithm;
3. find an edge e from E having its two ends in S. If there is no such an edge, Stop: S is feasible in G and hence optimal;
4. add the edge e to T and create (exactly one) circuit C in $T \cup \{e\}$;
5. delete another edge f having its two endpoints in $T \setminus S$ from the circuit C and obtain again a spanning tree T;
6. return to step 2.

During the algorithm, the rule of choosing a leaving edge avoid a spanning tree to be considered twice. This property guarantees that the algorithm does not cycle and terminates after a finite number of steps (since the number of spanning trees is finite).

Algorithm: MWSB (Maximum weight stable set in a bipartite graph)
Input: bipartite graph $G = (S_1, S_2; E)$
Output: maximum weight stable set S of weight $c(S)$

1. compute a spanning tree T in G using ST algorithm;
2. find a maximum weight stable set S in T using MWST algorithm;
3. find an edge $e = (x, y) \in E \mid x \in S$ and $y \in S$;
   if there is no such an edge, STOP: return S;
4. set $T := T \cup \{e\}$,
   let C be the circuit created in $T \cup \{e\}$ by adding e to T;
5. find an edge $f = (a, b) \in C \mid a \notin S$ and $b \notin S$;
   set $T := T \backslash \{f\}$, return to step 2.

Example: Let $G = (S_1, S_2; E)$ be the following weighted bipartite graph.

Node i	1	2	3	4	5	6	7	8
Weight $p(i)$	5	3	5	6	5	4	4	1

Table 4.2 – Node weights of the example

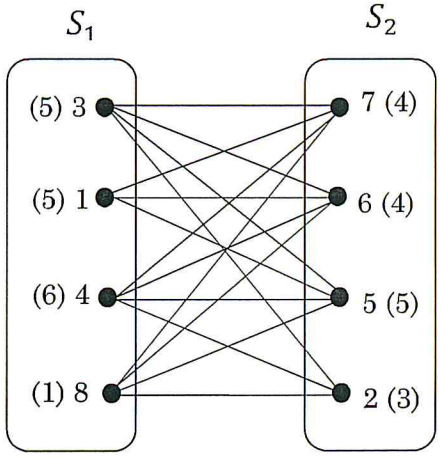


Fig. 4.2 – Graph of the example

Iteration 1

Step 1. Use the ST algorithm to compute a spanning tree T (fig. 4.1).

Step 2. In The MWST Algorithm:

Step a. Fix the node 1 as the root of the spanning tree;

Step b. $\varphi(1) = \varphi(2) = \cdots = \varphi(8) = true$; $S := \emptyset$; $c(S) := 0$;

Step c. level 4 node is 2 (note that, in a tree, a leaf has no sons);

$\omega^+(2) = 3$; $\omega^-(2) = 0$; $\omega(2) = max(3, 0) = 3$;

Level 3 nodes are 1, 3, and 4;

$\omega^+(8) = p(8) + \omega^-(2) = 1 + 0 = 1$;

$\omega^-(8) = \omega(2) = 3$;

$\omega(8) = max(3,1) = 3$;

$\omega^+(3) = 5, \omega^-(3) = 0, \omega(3) = max(5,0) = 5$;

$\omega^+(4) = 6, \omega^-(4) = 0, \omega(4) = max(6,0) = 6$;

Level 2 nodes are 5, 6 and 7;

$\omega^+(5) = p(5) + \omega^-(3) + \omega^-(4) + \omega^-(8) = 5 + 0 + 0 + 3 = 8;$

$\omega^-(5) = \omega(3) + \omega(4) + \omega(8) = 5 + 6 + 3 = 14;$

$\omega(5) = \max(8,14) = 14;$

$\omega^+(6) = 4, \omega^-(6) = 0, \omega(6) = 4;$

$\omega^+(7) = 4, \omega^-(7) = 0, \omega(7) = 4;$

Level 1 node is 1;

$\omega^+(1) = p(1) + \omega^-(5) + \omega^-(6) + \omega^-(7) = 5 + 14 + 0 + 0 = 19;$

$\omega^-(1) = \omega(5) + \omega(6) + \omega(7) = 14 + 4 + 4 = 22;$

$\omega(1) = max(22,19) = 22;$

Node i	1	2	3	4	5	6	7	8
$\omega^+(i)$	19	3	5	6	8	4	4	1
$\omega^-(i)$	22	0	0	0	14	0	0	3
$\omega(i)$	22	3	5	6	14	4	4	3

Table 4.3 – Summary of iteration 1

Step d: for the root (level 1), node 1, we have:

since $\varphi(1) = true$ and $\omega^+(1) = 19 < \omega^-(1) = 22$, then ignore node 1;

and pass to the next level (level 2, nodes 5, 6 and 7);

φ = true, for all nodes of level 2;

$\omega^+(5) < \omega^-(5)$, node 5 cannot belong to S;

$\omega^+(6) > \omega^-(6)$, $S = \{6\}$, $c(S) = 4$;

$\omega^+(7) > \omega^-(7)$, $S = \{6,7\}$, $c(S) = 4 + 4 = 8$;

Now, we pass to level 3 (nodes 3, 4 and 8);

We have $\varphi(3) = \varphi(4) = \varphi(8) = true$;

$\omega^+(3) > \omega^-(3)$, $S = \{6,7,3\}$, $c(S) = 8 + 5 = 13$;

$\omega^+(7) > \omega^-(7)$, $S = \{6,7,3,4\}$, $c(S) = 13 + 6 = 19$;

$\omega^+(8) < \omega^-(8)$, node 8 is ignored;

Level 4, node 2;

$\omega^+(2) > \omega^-(2)$, $S = \{6,7,3,4,2\}$, $c(S) = 19 + 3 = 22$;

At the end of step d, a stable set $S = (6, 7, 3, 4, 2)$ in the tree T is constructed (fig. 4.3), but S may not be stable in the original graph G. So, we proceed with adding and deleting edges from the spanning tree;

Iteration 2

Step 3: Edge $(6,3) \in E$, but $(6,3) \notin S$, so we add it to T. The circuit $C = (1,5,3,6)$ is created.

Step 4: Vertices 1 and $5 \notin S$, we remove then the edge (1,5) from T.

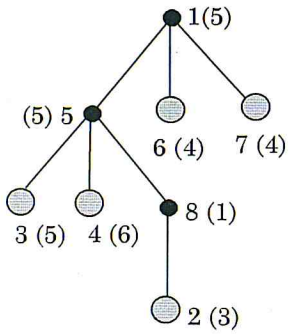


Fig. 4.3 – Stable set of iteration 1

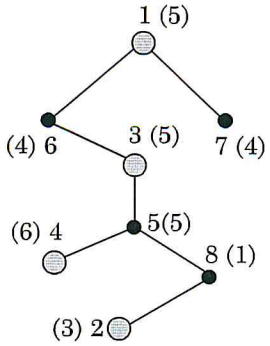


Fig. 4.4 – Stable set of iteration 2

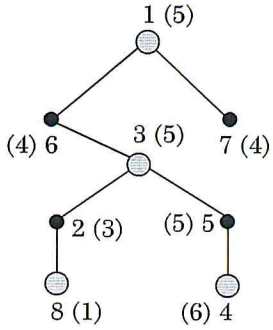


Fig. 4.5 – Stable set of iteration 3

Steps a, b and c in the MWST algorithm give these results.

Node i	1	2	3	4	5	6	7	8
$\omega^+(i)$	19	3	14	6	8	13	4	1
$\omega^-(i)$	18	0	9	0	9	14	0	3
$\omega(i)$	19	3	14	6	9	14	4	3

Table 4.4 – Summary of iteration 2

Step d:

$\varphi(1) = true$ *and* $\omega^+(1) = \omega^-(1), \mathcal{S} = \{1\}, c(\mathcal{S}) = 5, \varphi(6) = \varphi(7) =$ false;

We pass to node 3;

$\omega^+(3) > \omega^-(3), \mathcal{S} = \{1,3\}, c(\mathcal{S}) = 5 + 5 = 10, \varphi(2) = \varphi(5) =$ false;

$\omega^+(4) > \omega^-(4), \mathcal{S} = \{1,3,4\}, c(\mathcal{S}) = 10 + 6 = 16;$

$\omega^+(8) < \omega^-(8)$, we ignore node 8;

$\omega^+(2) > \omega^-(2), \mathcal{S} = \{1,3,4,2\}, c(\mathcal{S}) = 16 + 3 = 19$ (fig. 4.4).

Iteration 3

Step 3: Edge $(3,2) \in E$, but $\notin \mathcal{S}$, so we add it to T. The circuit $C = (3,2,8,5)$ is created.

Step 4: The vertices 8 and $5 \notin \mathcal{S}$, we remove then the edge (8,5) from T;

Steps a, b and c give the results summarized in the table below.

Node i	1	2	3	4	5	6	7	8
$\omega^+(i)$	17	3	12	6	5	13	4	1
$\omega^-(i)$	17	1	9	0	6	12	0	0
$\omega(i)$	17	3	12	6	6	13	4	1

Table 4.5 – Summary of iteration 3

Step d:

$\varphi(1) = true$ *and* $\omega^+(1) = \omega^-(1), \mathcal{S} = \{1\}, c(\mathcal{S}) = 5, \varphi(6) = \varphi(7) =$ false;

We pass to node 3;

$\omega^+(1) > \omega^-(1), \mathcal{S} = \{1,3\}, c(\mathcal{S}) = 5 + 5 = 10, \varphi(5) =$ false;

$\omega^+(4) > \omega^-(4), \mathcal{S} = \{1,3,4\}, c(\mathcal{S}) = 10 + 6 = 16;$

$\omega^+(5) < \omega^-(5)$, we ignore node 5;

$\omega^+(8) > \omega^-(8)$, $S = \{1,3,4,8\}$, $c(S) = 16 + 1 = 17$;

Step 3: there is no edge verifying the condition, then $S = \{1,3,4,8\}$ is optimal(fig. 4.5).

5. MWSC Algorithm (Maximum weight stable set in a chain)

A chain is a tree with one root and one leaf. All other nodes have exactly one son (successor) and one parent (predecessor) each.

Let $C = (u_1, u_2, , \dots, u_n)$ be a chain.

The algorithm for determining a maximum weight stable set in a chain uses the following recursive formula:

$$\omega(i) = \max_{1 \le i \le n} \{\omega(i-2) + p_i, \omega(i-1)\}$$

Where $\omega(i)$ is the maximum weight stable set in the sub-chain $(u_1, u_2, , \dots, u_i)$.

6. Algorithm yielding an optimal schedule in a chain

Let $C = (u_1, u_2, , \dots, u_n)$ be a chain.

Let $S = \{s_1, s_2, \dots, s_p\}$ be the maximum weight stable set obtained by MWSC, where jobs are in the order of C. Let $c(S)$ be the weight of the stable set S.

Let $\tilde{S} = C \backslash S = \{\tilde{s}_1, \tilde{s}_2, \dots, \tilde{s}_q\}$, in order, be the complement of S in C.

Algorithm: OSC
Input: Chain C, stable set S in C
Output: Optimal schedule
1. schedule all jobs of the set S on machine M_1 without idle times and in order;
2. for all jobs $\tilde{s}_i \in \tilde{S}$, in order, do
   find the earliest time t such that job $\tilde{s}_i$ is in agreement with all jobs being processed on machine M_1 in the time interval $[t, t + p_i]$, schedule job $\tilde{s}_i$ on machine M_2 at time t;
3. end.

Theorem

The OSC algorithm is polynomial and yields an optimal schedule of makespan $C_{max} = c(S)$.

Proof:

- The algorithm is polynomial since it is a list algorithm.
- We shall prove that $C(s_p) \geq C(\tilde{s}_q)$.

Let C be a chain, S a maximum weight stable set in C and $\tilde{S}$ the complement of S in C.

Let $S(u_i)$, $C(u_i)$ and $p(u_i)$ denote respectively the start time, the completion time and the processing time of job u_i, so

$C(u_i) - S(u_i) = p(u_i)$.

The chain $\mathcal{C}$ may take one of the following forms:

1. **Case $\mathcal{C} = (s_1\ \tilde{s}_1\ s_2\ \tilde{s}_2 \dots \tilde{s}_{p-1}\ s_p)$**

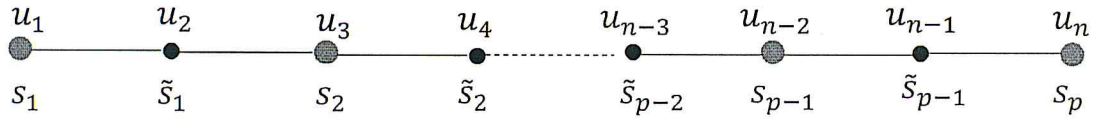


Fig. 4.6 –Type 1 chain

We have $p(\tilde{s}_i) \le p(s_i) + p(s_{i+1}), i = \overline{1, p-1}$, (otherwise, $S \cup \{\tilde{s}_i\}\backslash\{s_i, s_{i+1}\}$ would be a stable set of higher weight, contradiction with the fact that S is a maximum weight stable set),

Also from the maximum weight stable set properties, one can generalize as follows

$\sum_{k=i}^{j} p(\tilde{s}_k) \le \sum_{k=i}^{j+1} p(s_k), i = \overline{1, p-1},\ j = \overline{i, p-1}$.

Consider the first idle time on machine M_2, say after job $\tilde{s}_i$. We have

- $S(\tilde{s}_1) = S(s_1) = 0$,
- $p(\tilde{s}_1) \le p(s_1) + p(s_2) \implies C(\tilde{s}_1) \le C(s_2)$,
- $p(\tilde{s}_1) + p(\tilde{s}_2) \le p(s_1) + p(s_2) + p(s_3) \implies C(\tilde{s}_2) \le C(s_3)$,
- ...
- $p(\tilde{s}_1) + p(\tilde{s}_2) + \dots + p(\tilde{s}_i) \le p(s_1) + p(s_2) + p(s_3) + \dots + p(s_i) \implies$
  $\implies C(\tilde{s}_i) \le C(s_{i+1})$

Suppose that the second idle time is inserted just after job $\tilde{s}_j$. We have

- $S(\tilde{s}_{i+1}) = S(s_{i+1})$
- $p(\tilde{s}_{i+1}) \le p(s_{i+1}) + p(s_{i+2}) \implies C(\tilde{s}_{i+1}) - S(\tilde{s}_{i+1}) \le C(s_{i+2}) - S(s_{i+1}) \implies C(\tilde{s}_{i+1}) \le C(s_{i+2})$
- $p(\tilde{s}_{i+1}) + p(\tilde{s}_{i+2}) \le p(s_{i+1}) + p(s_{i+2}) + p(s_{i+3})$
  $\implies C(\tilde{s}_{i+2}) - S(\tilde{s}_{i+1}) \le C(s_{i+3}) - S(s_{i+1}) \implies C(\tilde{s}_{i+2}) \le C(s_{i+3})$
  ...
- $p(\tilde{s}_{i+1}) + p(\tilde{s}_{i+2}) + \dots + p(\tilde{s}_j) \le p(s_{i+1}) + p(s_{i+2}) + \dots + p(\tilde{s}_{j+1}) \implies$
  $\implies C(\tilde{s}_j) - S(\tilde{s}_{i+1}) \le C(s_{j+1}) - S(s_{i+1}) \implies C(\tilde{s}_j) \le C(s_{j+1})$

And so on until the last idle time, say after job l. We have then

- $S(\tilde{s}_{l+1}) = S(s_{l+1})$
- $p(\tilde{s}_{l+1}) \le p(s_{l+1}) + p(s_{l+2}) \implies$
- $\implies C(\tilde{s}_{l+1}) - S(\tilde{s}_{l+1}) \le C(s_{l+2}) - S(s_{l+1}) \implies C(\tilde{s}_{l+1}) \le C(s_{l+2})$
- ...
- $p(\tilde{s}_{l+1}) + p(\tilde{s}_{l+2}) + \dots + p(\tilde{s}_{p-1}) \le p(s_{l+1}) + p(s_{l+2}) + \dots + p(s_p) \implies$

$$\Rightarrow C(\tilde{s}_{p-1}) - S(\tilde{s}_{l+1}) \leq C(s_p) - S(s_{l+1})$$

$$\boxed{\Rightarrow C(\tilde{s}_{p-1}) \leq C(s_p)}$$

2. Case $\mathcal{C} = (\tilde{s}_1\, s_1\, \tilde{s}_2 \ldots \tilde{s}_p\, s_p \tilde{s}_{p+1})$

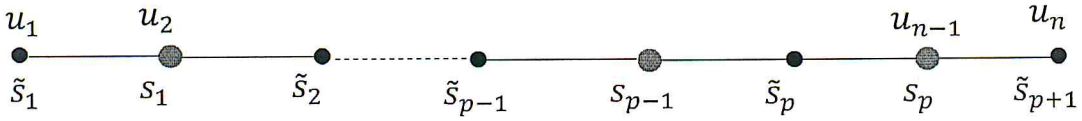


Fig. 4.7 –Type 2 chain

From the maximum weight stable set properties

$\sum_{k=1}^{i} p(\tilde{s}_k) \leq \sum_{k=1}^{i} p(s_k), i = \overline{1, p-1},$ (1)

$\sum_{k=i+1}^{j} p(\tilde{s}_k) \leq \sum_{k=i}^{j} p(s_k),\ i = \overline{1, p-1}, j = \overline{i+1, p-1}$ (2)

$\sum_{k=i+1}^{p+1} p(\tilde{s}_k) \leq \sum_{k=i}^{p} p(s_k),\ i = \overline{1, p}$ (3)

$\sum_{k=1}^{p+1} p(\tilde{s}_k) \leq \sum_{k=1}^{p} p(s_k)$ (4)

We consider the first idle time on machine M_2, say after job $\tilde{s}_i$. From (1) we have

$p(\tilde{s}_1) \leq p(s_2) \Rightarrow C(\tilde{s}_1) \leq C(s_2)$,

$p(\tilde{s}_1) + p(\tilde{s}_2) \leq p(s_1) + p(s_2) \Rightarrow C(\tilde{s}_2) \leq C(s_2)$,

...

$p(\tilde{s}_1) + p(\tilde{s}_2) + \cdots + p(\tilde{s}_i) \leq p(s_1) + p(s_2) + \cdots + p(s_i) \Rightarrow C(\tilde{s}_i) \leq C(s_i)$

Suppose now that the second idle time is inserted just after the job $\tilde{s}_j$. We have $S(\tilde{s}_{i+1}) = S(s_i)$. From (2), we have

$p(\tilde{s}_{i+1}) + p(\tilde{s}_{i+2}) + \cdots + p(\tilde{s}_j) \leq p(s_i) + p(s_{i+1}) + \cdots + p(s_j)$

$\Rightarrow C(\tilde{s}_j) - S(\tilde{s}_{i+1}) \leq C(s_j) - S(s_i) \Rightarrow C(\tilde{s}_j) \leq C(s_j)$

And so on until the last idle time, say after the job $\tilde{s}_l$. We have $S(\tilde{s}_{l+1}) = S(s_l)$.

From (2), we have $p(\tilde{s}_{l+1}) \leq p(s_l) + p(s_{l+1}) \Rightarrow C(\tilde{s}_{l+1}) - S(\tilde{s}_{l+1}) \leq C(s_{l+1}) - S(s_l)$

$\Rightarrow C(\tilde{s}_{l+1}) \leq C(s_{l+1})$

...

$p(\tilde{s}_{l+1}) + p(\tilde{s}_{l+2}) + \cdots + p(\tilde{s}_p) \leq p(s_l) + p(s_{l+1}) + \cdots + p(s_p)$

$\Rightarrow C(\tilde{s}_p) - S(\tilde{s}_{l+1}) \leq C(s_p) - S(s_l) \Rightarrow C(\tilde{s}_p) \leq C(s_p)$

And from (3), $p(\tilde{s}_{l+1}) + p(\tilde{s}_{l+2}) + \cdots + p(\tilde{s}_{p+1}) \leq p(s_l) + p(s_{l+1}) + \cdots + p(s_p)$

$\Rightarrow C(\tilde{s}_{p+1}) - S(\tilde{s}_{l+1}) \leq C(s_p) - S(s_l)$

$$\boxed{\Rightarrow C(\tilde{s}_{p+1}) \leq C(s_p)}$$

3. Case $\mathcal{C} = (\tilde{s}_1\, s_1\, \tilde{s}_2 \ldots \tilde{s}_p\, s_p)$

This case is obtained from the case 2 by removing the last vertex $\tilde{s}_{p+1}$ and the equations (3) and (4).

4. **Case $\mathcal{C} = (s_1\ \tilde{s}_1\ s_2\ \tilde{s}_2 \dots s_p\ \tilde{s}_p)$**

This case is obtained by considering the reversal of the chain of type 3.

5. **Case $\mathcal{C} = (\dots s_i \tilde{s}_j\ \tilde{s}_{j+1} s_{i+1} \dots)$**

We consider the sub-chains $\mathcal{C}_1 = (\dots s_i\ \tilde{s}_j)$ and $\mathcal{C}_2 = (\tilde{s}_{j+1}\ s_{i+1} \dots)$.

It is clear that $\tilde{s}_j$ is in agreement only with s_i, and $\tilde{s}_{j+1}$ is in agreement only with s_{i+1}, so it must occur an idle time after $\tilde{s}_j$, then $\tilde{s}_{j+1}$ and s_{j+1} would start processing at the same time.

Hence, the proof, for this case, can be performed separately for both sub-chains $\mathcal{C}_1$ and $\mathcal{C}_2$ using the corresponding case for each sub-chain.

Claim 1: $\mathcal{C}$ may contain a series of at most 2 vertices of $\tilde{S}$.

Proof: Suppose that $\mathcal{C}$ contains a series of 3 vertices of $\tilde{S}$, say $\tilde{s}_{i-1}$, $\tilde{s}_i$ and $\tilde{s}_{i+1}$. $S \cup \{\tilde{s}_i\}$ is also a stable set since the 2 adjacent vertices of the vertex $\tilde{s}_i$ are not part of the stable set S, and this stable set has a higher weight than S. Contradiction with the fact that S is a maximum weight stable set. ■

Claim 2: $\mathcal{C}$ may not begin or end with more than 1 vertex of $\tilde{S}$.

Proof: Suppose that $\mathcal{C}$ is of the form $(\tilde{s}_1\ \tilde{s}_2 \dots)$, respectively of the form $(\dots \tilde{s}_{q-1}\ \tilde{s}_q)$, $S \cup \{\tilde{s}_1\}$, respectively $S \cup \{\tilde{s}_q\}$, would be a stable set of higher weight. Contradiction. ■

Chapter 5

Near-Optimal Methods

In this chapter, some heuristics, local search methods and metaheuristics are used to deal with the SWA problem, all based on lists.

I. Heuristics

1. Definitions

The agreement number of a job J_j, noted $Dg(J_j)$, is the number of jobs in agreement with job J_j. It is also the degree of the node J_j in the agreement graph.

The iterative agreement number of a job J_j, noted $dg(J_j)$, is the number of non-scheduled jobs in agreement with job J_j. It is also the degree of the node J_j in the modified agreement graph obtained by removing iteratively nodes corresponding to scheduled jobs.

2. Notations

Let LPT denote the sort order based on "Longest Processing Time first".

Let SPT denote the sort order based on "Shortest Processing Time first".

Let $RAND$ be the random sort.

Let p denote, as usually, the processing time.

Let $Dg+$ and $Dg-$ denote the sort order of the jobs list based respectively on increasing and decreasing agreement number of jobs.

Let $dg+$ and $dg-$ denote the sort order of the jobs list based respectively on increasing and decreasing iterative agreement number of jobs.

The lists introduced in this chapter, are sorted according to 1 or 2 criteria. Thus, jobs are sorted according to a first criterion, and then, those with the same value are

eventually ordered according to a second sort criterion. For example, "$LPT|Dg+$" means that the selection is based first on the LPT rule and then jobs with the same processing time are sorted according to increasing node degree of the corresponding job. According to these notations, we give the list of the 14 heuristics.

Heur.	$HL1$	$HL2$	$HL3$	$HL4$	$HL5$	$HL6$	$HL7$
Descr.	LPT	SPT	$LPT\|Dg+$	$LPT\|dg+$	$Dg+$	$Dg-$	$dg+$
Heur.	$HL8$	$HL9$	$HL10$	$HL11$	$HL12$	$HL13$	$HL14$
Descr.	$dg-$	$Dg+\|LPT$	$dg+\|LPT$	$Dg/p+$	$dg/p+$	$Dg.dg/p+$	$RAND$

Table 5.1 – Heuristics list

3. List-algorithm

Let V be the set of jobs and U the set of unscheduled jobs. A non-scheduled job J_j is said to be available at time t if there exists a free machine at this time, and if jobs yet scheduled and having a part of their processing in the interval $[t, t+p_j]$ are all in agreement with J_j. This algorithm is of complexity $O(n)$.

Algorithm: LS (list algorithm)

Input: jobs list (for example, LPT list)

Output: feasible schedule

1. set $U := V$, $t := 0$;
2. find the earliest time t at which a job from U becomes available;
   let $\overline{U}$ be the set of jobs of U available at time t;
3. choose a job J_j from $\overline{U}$ of higher priority, schedule it on a machine at time t;
   set $U := U \backslash \{J_j\}$;
4. if $U = \emptyset$, stop: all jobs are scheduled;
   Otherwise return to step 2.

Example: We retake the same example of chapter 3 ($m = 2, n = 5$).

Job J_j	J_1	J_2	J_3	J_4	J_5
Proc. times	2	4	1	3	2

Table 5.2 – Processing times of the example

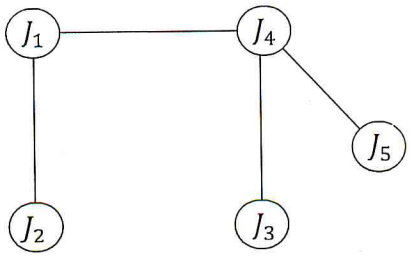


Fig. 5.1 – Agreement graph of the example

Iteration 1:

We sort the jobs list according to any rule, say LPT rule, we obtain

$U = \{J_2, J_4, J_1, J_5, J_3\}$; $t := 0$;

At time zero, all jobs are available, $\overline{U} = U$.

Since all jobs are available and all machines are free at time $t = 0$, schedule the job of higher priority that is J_2 on any machine, say machine M_1 (fig. 5.2).

$U = U - \{J_2\} = \{J_4, J_1, J_5, J_3\}$.

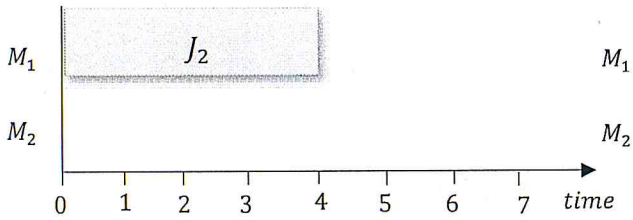


Fig. 5.2 - Gantt chart of iteration 1

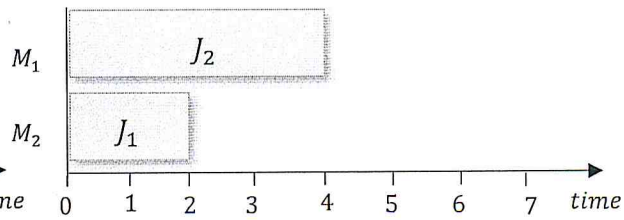


Fig 5.3 - Gantt chart of iteration 2

Iteration 2: Since the machine M_2 is free at time zero and the job J_1 agrees with job J_2, then the earliest time is $t = 0$.

$\overline{U} := \{J_1\}$ (J_1 is the only job available at time zero).

Schedule job J_1 on machine M_2 (fig. 5.3).

$U := \{J_4, J_5, J_3\}$;

Iteration 3: In the interval [2,4], a part of the job J_2 is being scheduled, and none of the non-scheduled jobs is in agreement with J_2. The earliest time at which a job becomes available should be after the completion of J_2, $t = 4$. At this time, all jobs are available. We choose the one of higher priority, J_4, and schedule it on M_1 (fig. 5.4).

$U := \{J_5, J_3\}$;

Iterations 4 and 5: Both jobs J_3 and J_5 agree with job J_4, and there is a free machine M_2 at time $t = 4$. So we schedule J_5 and then J_3 in the priority order on M_2 at times 4 and 6. $U = \emptyset$. Stop: all jobs are scheduled (fig. 5.5).

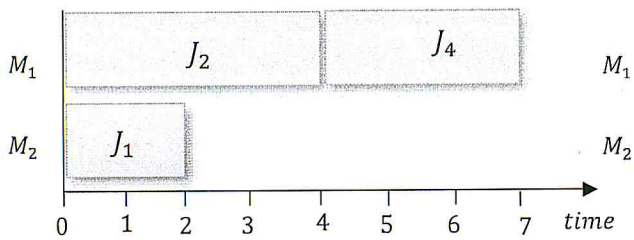


Fig 5.4 - Gantt chart of iteration 3

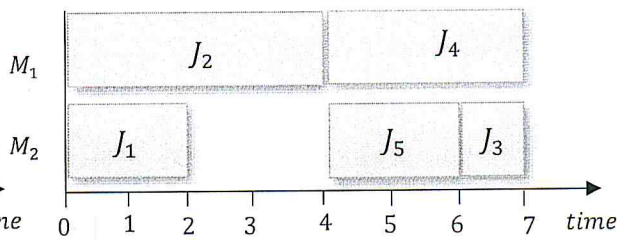


Fig 5.5 - Gantt chart of iteration 5

II. Local search methods

1. Pairwise Interchange (PI)

The pairwise interchange (PI) is a permutation-based method used to generate a near-optimal solution of a scheduling problem, and is considered as a local search method. It operates by job interchanging on a pre-defined list. It proceeds as follows. Get an initial permutation, and consequently an initial schedule (using LPT heuristic, for example) giving an initial value of the makespan. Select randomly a pair of jobs from the initial permutation, operate interchange between them to move from the initial permutation to a new permutation to obtain a new schedule.
If the new schedule yields a better value of the makespan, the new permutation is accepted and replace the initial one for the next iteration. Otherwise, it is rejected, and a new pair of jobs will be generated. This operation is repeated until a lower bound is hit or after a fixed number of iterations without improvement is reached.

Algorithm: PI (Pairwise Interchange)
Input: Initial Jobs list
Output: Jobs list yielding better or equal schedule

1. get an initial job permutation π_{ini} (use LPT heuristic, for example), run the List algorithm on π_{ini} to obtain an initial feasible schedule π_{ini} of makespan C_{ini};
   set $k := 0$, $\pi_{best} := \pi_{ini}$, $C_{best} := C_{ini}$;
2. if $k \geq limit$, stop: return π_{best}, C_{best};
3. set $k := k + 1$;
4. generate 2 random discrete numbers in $[1, n]$, say i, j and get a new permutation π_{new} by interchanging $\pi_{best}(i)$ and $\pi_{best}(j)$;
5. run the List algorithm on π_{new} to obtain a new feasible schedule of makespan C_{new};
6. if $C_{new} = LB$, stop: $\pi_{opt} := \pi_{new}$, $C_{opt} = C_{new}$;
7. if ($C_{new} > C_{best}$) or ($C_{new} = C_{best}$ and $rand[0,1] < 0.5$),
   set $\pi_{best} := \pi_{new}$, $C_{best} = C_{new}$, $k := 0$;
8. go to step 2.

2. Adjacent Pairwise Interchange (API)

The adjacent pairwise interchange (API) method is the same as PI except that the jobs to be interchanged must be adjacent in the current permutation.

Algorithm: API (Adjacent Pairwise Interchange)
Input: Initial Jobs list
Output: Jobs list yielding better or equal schedule

1. get an initial job permutation π_{ini} (use LPT heuristic, for example), run the List algorithm on π_{ini} to obtain an initial feasible schedule π_{ini} of makespan C_{ini};
   set $k := 0$, $\pi_{best} := \pi_{ini}$, $C_{best} := C_{ini}$;
2. if $k \geq limit$, stop: return π_{best}, C_{best};
3. set $k := k + 1$;
4. generate a random discrete number in $[1, n-1]$, say i, and get a new permutation π_{best} by interchanging positions of $\pi_{ini}(i)$ *and* $\pi_{ini}(i+1)$;
5. run the List algorithm on π_{new} to a obtain a new feasible schedule of makespan C_{new};
6. if $C_{new} = LB$, stop: $\pi_{opt} := \pi_{new}$, $C_{opt} = C_{new}$;
7. if ($C_{new} > C_{best}$) or ($C_{new} = C_{best}$ and $rand[0,1] < 0.5$),
   set $\pi_{best} := \pi_{new}$, $C_{best} = C_{new}$, $k := 0$;
8. go to step 2.

3. Insertion Method

Insertion Method is a variable neighborhood search method proceeding by inserting a randomly chosen position of a job in front (or back) of another randomly chosen job position.

Algorithm: IM (Insertion Method)
Input: Initial Jobs list
Output: Jobs list yielding better or equal schedule

1. get an initial job permutation π_{ini} (use LPT heuristic, for example), run the List algorithm on π_{ini} to obtain an initial feasible schedule π_{ini} of makespan C_{ini};
   set $k := 0$, $\pi_{best} := \pi_{ini}$, $C_{best} := C_{ini}$;
2. if $k \geq limit$, stop: return π_{best}, C_{best};
3. set $k := k + 1$;
4. generate 2 random discrete numbers in $[1, n]$, say i, j and get a new permutation π_{best} by inserting $\pi_{ini}(i)$ in front of $\pi_{ini}(j)$.
5. run the LS on π_{new} to a obtain a new feasible schedule of makespan C_{new};
6. if $C_{new} = LB$, stop: $\pi_{opt} := \pi_{new}$, $C_{opt} = C_{new}$;
7. if ($C_{new} > C_{best}$) or ($C_{new} = C_{best}$ and $rand[0,1] < 0.5$),
   set $\pi_{best} := \pi_{new}$, $C_{best} = C_{new}$, $k := 0$;
8. go to step 2.

III. Metaheuristics

1. Simulated Annealing (SA)

The algorithm SA proposed proceeds as follows.

First, choose an initial permutation by any heuristic. Choose randomly 2 job positions. A new permutation in the neighborhood is obtained by interchanging the positions of the 2 selected jobs (a permutation reachable by only one move is called a neighbor).

In SA, solutions are accepted according to the magnitude of increase in the objective function and the temperature. Thus, $P(accept) = exp(-\lambda . \Delta L/L)$, where λ is a control parameter (temperature) and ΔL is the change in the objective function (makespan). If the makespan increases after the random pairwise interchange, the new permutation is accepted if $P(accept) > r$, where r is a uniform random number between 0 and 1[27].

Algorithm: SA (Simulated Annealing)
Input: Initial jobs list
Output: Jobs list yielding better or equal schedule

1. get an initial job permutation π_{ini} (use LPT heuristic, for example); run the List algorithm on π_{ini} to obtain an initial feasible schedule of makespan C_{ini};
   initialize β, $limit$; set $\pi_{best} = \pi_{ini}$, $\pi_{old} = \pi_{ini}$, $k := 0$, $t := 0$;
2. if $k \geq limit$, stop: return π_{best}, C_{best};
3. set $k := k + 1$; $t := t + 1$; $\lambda := \beta . t$;
4. generate 2 random discrete numbers in $[1, n]$, say i, j and get a new permutation π_{new} by interchanging the positions of $\pi_{old}(i)$ and $\pi_{old}(j)$;
5. run the List algorithm on π_{new} to obtain a new feasible schedule of makespan C_{new};
6. if $C_{new} = LB$, stop, $\pi_{opt} := \pi_{new}$, $C_{opt} := C_{new}$;
7. if $C_{new} < C_{old}$, set $\pi_{best} := \pi_{new}$, $\pi_{old} := \pi_{new}$, $k := 0$, go to step 2;
8. set $P = exp(-\lambda . (C_{new} - C_{old})/C_{old})$;
9. if $P < rand[0,1]$, set $\pi_{old} := \pi_{new}$;
10. go to step 2

2. Harmony Search (HS)

The steps of HS are described below.

Step 1. Initialize algorithm parameters

The HS parameters are initialized in this step, including harmony memory size (HMS), harmony memory considering rate (HMCR), pitch adjusting rate (PAR),

distance bandwidth (bw) and the number of improvisations (NI). Here, HMS is the number of solutions in the harmony memory. $HMCR$ and PAR are parameters used to search for globally and locally improved solutions, respectively.

Step 2. Initialize the harmony memory

The initial HM matrix, shown below, is filled with HMS number of harmonies. Each harmony, represented as an n-dimensional real-valued vector, is randomly generated by using a uniform distribution.

Step 3. Improvise a new harmony

Let $H_{new} = \{h_{new}(1), h_{new}(2), \dots, h_{new}(n)\}$ be a new harmony, which is generated by using three rules: memory consideration, pitch adjustment and random selection.

In the memory consideration, each decision variable $h_{new}(j), j = \overline{1,n}$, is chosen randomly from the j^{th} column of the HM matrix with the probability of $HMCR$. And is generated randomly in the range $[LB_j, UB_j]$ as follows: $h_{new}(j) = LB_j + r.(UB_j - LB_j)$ with probability $1 - HMCR$, where r is a [0,1] –uniformly generated random number. Furthermore, every component obtained by the memory consideration is adjusted by the pitch adjustment rule with the proportion of PAR.

$$h_{new}(j) = h_{old}(j) \pm r.bw$$

Step 4. Update the harmony memory

If the fitness value of the new harmony is better than that of the worst one in the HM, the new generated harmony is included into the HM and the worst harmony is excluded from the HM.

Due to the continuous nature of Harmony Search, it cannot be used in its original form for the scheduling problem. Thus, in the discrete case, a convenient mapping must be found which convert harmonies to solutions. We adopt then a permutation-based encoding method, and a list scheduling rule is used to convert the harmony vectors to a solution of the problem. More specifically, a harmony is represented as a permutation of integers $\pi = (\pi_1, \pi_2, \dots, \pi_n)$, where n is the number of jobs and π_i denotes the order of the i^{th} job to be scheduled.

To initialize HM, some of the HMS harmonies may be chosen from the heuristic lists as initial sequences and others randomly. Next, a novel improvisation process is used to generate a new harmony.

Each component of the harmony vector is chosen randomly from the j^{th} column of the HM matrix with the probability of $HMCR$, and then, with a probability of PAR, is adjusted by this pitch adjustment

$$h_{new}(j) = h_{old}(j) \pm b, \quad b = \overline{0, bw}$$

And each component of the harmony vector is chosen randomly in the range $[0, n]$ with a probability $1 - HMCR$, and won't be adjusted.

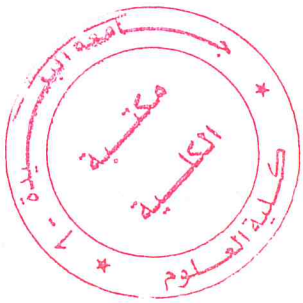
But the new harmony obtained may not be a permutation of job sequences since it contains eventually duplicate elements. Thus, it should be repaired to be a feasible solution. A repair process is used by replacing each duplicate integer by an integer that has not been assigned to any job.

Algorithm: HS (Harmony Search)
Input: Initial jobs list
Output: Jobs list yielding better or equal schedule
Let $C(\pi)$ denote the makespan of the sequence π.

1. initialize $HMS, HMCR, PAR, bw, NI$; set $k = 0$;
2. get HMS number of harmonies (sequences $\pi_1, \pi_2, \dots, \pi_{HMS}$) from heuristic lists and/or randomly;
3. set $K = K + 1$;
4. for $i = 1$ to n do
   if $rand[0,1] < HMCR$ then
   $r = Int\ rand[1, HMS]$, $\pi(i) = \pi_r(i)$;
   if $rand[0,1] < PAR$, then $b = Int\ rand[0, bw], \pi(i) = \pi(i) \pm b$;
   else $\pi(i) = Int\ rand[1, n]$;
   next i
5. for $i = 1$ to n
   for $j = i + 1$ to n
   if $\pi(i) = \pi(j)$ then
   find $k\ in\ [1, n] \mid k$ does not appear in π sequence;
   set $\pi(j) := k$;
   next j
   next i
6. run the List algorithm on π to obtain a feasible schedule of makespan $C(\pi)$;
7. find the sequence $\pi_{min} \mid C(\pi_{min}) = \min_{1 \le i \le HMS} C(\pi_i)$;
8. Find the sequence $\pi_{max} \mid C(\pi_{max}) = \max_{1 \le i \le HMS} C(\pi_i)$;
9. if $C(\pi) \le C(\pi_{max})$, set $\pi_{max} := \pi$;
10. if $C(\pi_{min}) = LB$, STOP: set $\pi_{opt} := \pi_{min}$, return π_{opt};
11. if $K = NI$, STOP: return π_{min};
12. return to step 4.

Chapter 6

Numerical Experiments



In this chapter, various experiments on heuristics, metaheuristics and exacts methods are performed on a computer equipped with Intel I5 CPU and 8 GB of RAM using "Visual Basic 6.0" and "Cplex" development software. All instances are uniformly generated.

I. Heuristics experiments

In the previous chapter, we have described 14 heuristics, all based on lists, to deal with the *SWA* problem. In the present chapter, we carry out some experiments on these heuristics and try to determine their performance for different problem configurations. Pretests have shown that 4 heuristics ($HL2$, $HL6$, $HL8$ and $HL14$) give very bad results in almost all cases, so we rule out this heuristics and keep only the 10 remaining.

Two types of agreement graph, general and bipartite, are considered, each with different values of density (d): low ($d_1 = 20\%$), medium ($d_2 = 50\%$) and high ($d_3 = 80\%$),).

In the general agreement graph case, jobs number (n) is chosen from the set $\{20, 100, 500, 1000\}$, machines number (m) from $\{2, 3, 5, 10, 20\}$ and processing times (p_i) from $\{I_1 = [1, 10], I_2 = [30, 50]\}$. In the bipartite agreement graph case, jobs numbers are from $\{20, 50, 100\}$ and processing times from $\{I_1, I_2\}$.

A. Experiments in the general agreement graph case

All combinations of heuristics number, jobs number, machines number, graph density and processing times range give a total of more than 100 tables of 10 columns and 4 rows, which is too big to take in this thesis. So we have considered only tables with mean values (14 tables) detailed as follows.

- for $n = 20$, we consider the average of the combination of 3 values of m (2,3,5), two processing times intervals $I_1$and I_2, 3 graph densities d_1, d_2 *and* d_3, which give a total of $3x2x3 = 18$ combinations, and for each combination we have generated 1000 instances (that is 18000 instances).
- for $n = 100$, we consider the average of the combination of 4 values of m (2,3,5,10), two processing times intervals I_1, I_2 and 3 graph densities d_1, d_2 and d_3, which give a total of $4x2x3 = 24$ combinations, and for each combination we have generated 500 instances (that is 12000 instances).
- for $n = 500$, m is chosen from {2,3,5,10,20}, p_i in I_1 or I_2 and d from $\{d_1, d_2, d_3\}$. The instances number is $5x2x3x100 = 3000$.
- for $n = 1000$, same parameters as for 500, except 50 instances are generated, which give a total of $5x2x3x50 = 1500$.

Total instances: $18000 + 12000 + 3000 + 1500 = 34500$.

The 4 performance criteria considered are:

- *Best Cmax*: The number of times where the heuristic gives the best value of *Cmax* (in percentage);
- *Opt. Cmax*, The number of times where the heuristic gives an optimal value of *Cmax* (in percentage);
- *Mean. dev.*: The average deviation of the heuristic compared with the lower bound (in percentage, and calculated as $100x(Cmax - LB)/LB$);
- *Max. dev.*: The maximum deviation of the heuristic compared with the lower bound (in percentage).

Note that the last column of each table refers to the best values among the heuristics.

1. Average performance by jobs number

n = 20	*HL1*	*HL3*	*HL4*	*HL5*	*HL7*	*HL9*	*HL10*	*HL11*	*HL12*	*HL13*	*Best*
Best Cmax	21.46	19.16	24.79	21.53	20.03	24.90	33.78	31.84	35.22	**36.84**	
Opt. Cmax	10.21	9.04	12.59	5.28	5.27	6.56	15.45	13.11	16.51	**16.83**	26.73
Mean Dev.	17.97	18.54	17.41	16.41	16.68	15.91	15.51	15.34	15.23	**14.92**	10.43
Max Dev.	50.68	52.83	50.62	50.13	48.56	49.29	48.44	**45.81**	47.00	46.83	35.68
n = 100	*HL1*	*HL3*	*HL4*	*HL5*	*HL7*	*HL9*	*HL10*	*HL11*	*HL12*	*HL13*	*Best*
Best Cmax	14.78	11.59	22.80	10.44	12.44	12.13	28.65	20.63	29.03	**31.75**	
Opt. Cmax	11.20	9.06	17.48	1.93	3.51	2.21	17.71	10.58	16.76	**16.99**	25.92
Mean Dev.	44.23	44.87	43.71	42.78	42.78	42.38	42.24	42.04	41.99	**41.32**	37.35
Max Dev.	64.66	64.18	62.53	62.79	61.84	61.61	61.00	59.78	59.48	**57.86**	52.06

$n = 500$	*HL1*	*HL3*	*HL4*	*HL5*	*HL7*	*HL9*	*HL10*	*HL11*	*HL12*	*HL13*	*Best*
Best Cmax	17.00	13.91	**38.84**	4.52	11.52	5.05	34.61	20.15	35.17	38.03	
Opt. Cmax	13.30	10.85	**28.58**	1.63	4.07	1.66	21.50	12.47	21.92	21.47	32.14
Mean Dev.	54.19	54.32	53.87	55.07	54.55	54.98	54.46	53.52	53.22	**53.07**	51.83
Max Dev.	60.14	59.85	59.44	61.70	60.34	61.50	60.36	59.53	59.15	**58.54**	56.30
$n = 1000$	*HL1*	*HL3*	*HL4*	*HL5*	*HL7*	*HL9*	*HL10*	*HL11*	*HL12*	*HL13*	*Best*
Best Cmax	18.17	12.63	**44.63**	5.00	9.67	3.33	32.73	20.03	41.83	40.50	
Opt. Cmax	13.77	11.20	**33.83**	1.00	4.27	1.23	23.90	13.40	25.33	25.83	36.90
Mean Dev.	48.45	48.61	48.23	49.69	49.39	49.69	49.24	48.14	47.89	**47.80**	47.12
Max Dev.	50.09	50.15	49.72	51.74	51.30	51.85	50.92	49.84	49.51	**49.38**	48.01

Table 6.1, *Average Performance by Jobs Number*

Remarks

For n less than 500, *HL*13 is the best heuristic for all performance criteria. But for a bigger number, *HL*4 becomes better than all others according to all performance criteria. Globally, the optimality is reached more often when n gets bigger (around 37% for $n = 1000$).

It should be noted that, for each heuristic or globally, the mean and max deviation are inversely proportional to the best and optimal Cmax.

2. Average performance by graph density

$d = 20$	*HL1*	*HL3*	*HL4*	*HL5*	*HL7*	*HL9*	*HL10*	*HL11*	*HL12*	*HL13*	*Best*
Best Cmax	7.52	4.97	19.19	10.43	13.59	10.34	19.79	16.44	29.33	**31.46**	
Opt. Cmax	1.60	1.05	8.54	1.10	3.23	1.29	7.69	2.48	9.30	**9.46**	14.24
Mean Dev.	78.05	78.44	77.68	77.91	77.87	77.78	77.57	76.13	75.93	**75.64**	73.10
Max Dev.	96.66	96.15	95.48	95.33	95.28	96.12	95.60	93.47	93.33	**91.69**	87.28
$d = 50$	*HL1*	*HL3*	*HL4*	*HL5*	*HL7*	*HL9*	*HL10*	*HL11*	*HL12*	*HL13*	*Best*
Best Cmax	16.09	11.82	32.01	8.01	13.45	9.23	31.09	19.86	33.44	**35.18**	
Opt. Cmax	10.10	7.30	**22.39**	1.77	4.27	2.13	18.46	10.32	19.48	19.77	29.32
Mean Dev.	41.89	42.21	41.51	42.01	41.74	41.71	41.56	40.70	40.58	**40.24**	37.48
Max Dev.	55.18	55.42	54.30	56.21	55.01	55.38	54.80	52.98	53.19	**52.84**	47.91
$d = 80$	*HL1*	*HL3*	*HL4*	*HL5*	*HL7*	*HL9*	*HL10*	*HL11*	*HL12*	*HL13*	*Best*
Best Cmax	28.75	24.57	**49.77**	8.96	10.99	9.97	46.01	30.30	43.21	43.69	
Opt. Cmax	25.29	22.09	**41.94**	3.58	5.01	4.11	34.18	24.13	32.81	32.76	48.94
Mean Dev.	11.43	11.79	11.03	11.24	10.99	11.01	10.24	10.60	10.36	**10.08**	8.21
Max Dev.	19.24	19.99	18.60	20.38	18.56	18.96	17.38	17.42	17.10	**17.03**	12.96

Table 6.2, *Average Performance by graph density*

Remarks

For a small or average graph density, *HL*13 and *HL*12 get by far ahead of all other heuristics. But when the graph is getting thicker, *HL*4 gets ahead of them (around 50% best heuristic for a graph dense at 80%). The overall optimality increases with the graph density to reach 49% for a graph density of 80%.

3. Average performance by machines Number

$m = 2$	*HL1*	*HL3*	*HL4*	*HL5*	*HL7*	*HL9*	*HL10*	*HL11*	*HL12*	*HL13*	*Best*
Best Cmax	39.46	33.95	**68.31**	9.61	17.56	10.67	48.91	34.49	52.78	52.14	
Opt. Cmax	36.46	31.00	**61.62**	7.06	13.81	7.81	44.34	30.53	47.43	46.50	76.75
Mean Dev.	2.03	2.16	1.83	2.30	2.08	2.21	1.76	1.70	1.55	**1.54**	0.86
Max Dev.	6.69	6.76	6.43	7.84	6.92	7.63	5.95	6.13	5.86	**5.63**	3.99
$m = 3$	*HL1*	*HL3*	*HL4*	*HL5*	*HL7*	*HL9*	*HL10*	*HL11*	*HL12*	*HL13*	*Best*
Best Cmax	17.85	14.57	32.25	6.63	14.45	7.94	44.75	24.71	45.45	**46.03**	
Opt. Cmax	14.18	11.25	24.56	1.35	3.64	2.15	26.83	17.73	26.31	**27.43**	31.81
Mean Dev.	5.03	5.22	4.76	5.22	4.73	5.07	4.20	4.34	4.01	**3.95**	2.54
Max Dev.	14.18	14.71	13.67	15.75	13.84	15.48	13.49	12.79	12.29	**11.58**	8.37
$m = 5$	*HL1*	*HL3*	*HL4*	*HL5*	*HL7*	*HL9*	*HL10*	*HL11*	*HL12*	*HL13*	*Best*
Best Cmax	10.12	7.31	30.73	11.11	12.10	11.60	30.49	20.14	32.01	**33.48**	
Opt. Cmax	4.85	3.41	**18.56**	1.26	1.32	1.34	14.06	7.08	14.05	14.47	23.39
Mean Dev.	21.00	21.42	20.53	19.70	19.56	19.41	18.98	18.87	18.68	**18.35**	15.75
Max Dev.	37.87	39.35	37.75	37.03	35.39	36.02	36.06	**34.27**	34.71	35.10	28.90
$m = 10$	*HL1*	*HL3*	*HL4*	*HL5*	*HL7*	*HL9*	*HL10*	*HL11*	*HL12*	*HL13*	*Best*
Best Cmax	6.24	3.11	15.74	8.65	8.08	7.93	21.02	11.42	22.44	**23.51**	
Opt. Cmax	0.02	0.00	6.09	0.00	0.00	0.00	7.00	0.07	**6.14**	6.13	9.07
Mean Dev.	66.19	66.69	65.83	66.18	66.08	66.00	65.71	64.69	64.37	**63.88**	60.82
Max Dev.	78.84	77.78	77.42	78.09	78.00	79.05	76.54	75.34	75.73	**74.52**	69.31
$m = 20$	*HL1*	*HL3*	*HL4*	*HL5*	*HL7*	*HL9*	*HL10*	*HL11*	*HL12*	*HL13*	*Best*
Best Cmax	8.78	5.11	11.17	8.33	7.83	9.72	6.06	16.17	15.39	**21.17**	
Opt. Cmax	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Mean Dev.	156.78	157.52	156.14	157.82	157.92	157.39	157.69	154.82	154.97	**154.11**	149.38
Max Dev.	180.98	180.32	178.40	181.39	180.95	179.64	181.11	177.38	177.43	**175.23**	168.47

Table 6.3, Average Performance by machines number

Remarks

For 2 machines, *HL4* is the best heuristic (best Cmax at 68% and Opt Cmax at 61%). For 3 machines or more, *HL13* and *HL12* become the best (For *HL13*, the rate of best Cmax is equal to 46%, for $m = 3$, and then it decreases to reach 21% for $m = 20$). The global optimality starts at 76% for $m = 2$ and ends at zero for $m = 20$. The global mean deviation is less than 1% for 2 machines, but it grows rapidly to attain 60% for 10 machines and around 150% for 20 machines.

4. Average performance by processing times range

$p \in [1, 10]$	*HL1*	*HL3*	*HL4*	*HL5*	*HL7*	*HL9*	*HL10*	*HL11*	*HL12*	*HL13*	*Best*
Best Cmax	25.63	19.43	47.84	8.48	11.96	10.10	44.48	34.67	53.84	**57.28**	
Opt. Cmax	18.91	15.68	36.15	3.75	7.48	4.49	37.57	22.99	38.72	**39.56**	44.89
Mean Dev.	40.59	41.12	40.10	42.08	41.76	41.77	41.14	39.47	39.49	**39.13**	37.16
Max Dev.	54.07	54.76	53.17	56.82	55.22	55.99	54.73	52.18	52.39	**51.73**	48.28
$p \in [30, 50]$	*HL1*	*HL3*	*HL4*	*HL5*	*HL7*	*HL9*	*HL10*	*HL11*	*HL12*	*HL13*	*Best*
Best Cmax	8.68	7.72	17.88	9.73	13.21	9.56	**19.88**	9.66	16.62	16.10	
Opt. Cmax	5.18	4.26	**11.06**	0.49	0.69	0.49	2.42	1.57	2.18	1.62	15.15
Mean Dev.	46.99	47.17	46.71	45.36	45.31	45.22	45.11	45.47	45.08	**44.83**	42.04
Max Dev.	59.98	59.60	59.09	57.79	57.34	57.63	57.12	57.06	56.68	**55.98**	50.49

Table 6.4 – Average performance by processing times range

Remarks

*HL*13 is the best heuristic for small values of processing times, $p_i \in I_1$, and *HL*10 is the best for bigger values of p_i, from I_2.

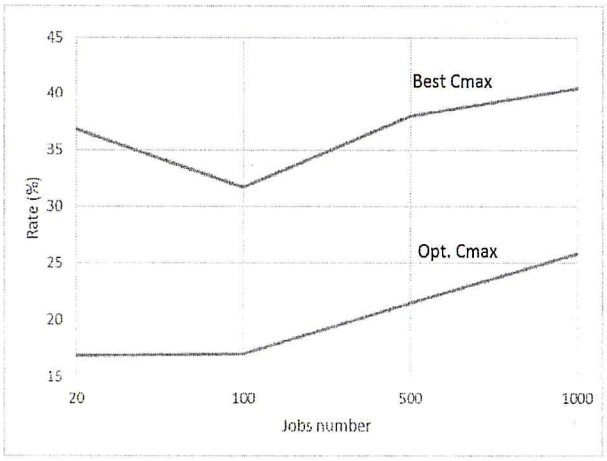
5. Overall average performance

General	*HL*1	*HL*3	*HL*4	*HL*5	*HL*7	*HL*9	*HL*10	*HL*11	*HL*12	*HL*13	*Best*
Best Cmax	17.18	13.56	32.57	9.39	12.63	10.18	31.59	22.18	34.70	**36.23**	
Opt. Cmax	11.79	9.73	**23.10**	2.16	4.04	2.53	19.36	11.89	19.78	19.92	29.66
Mean Dev.	45.34	45.72	44.95	45.27	45.10	45.04	44.67	43.99	43.81	**43.49**	41.00
Max Dev.	59.23	59.42	58.33	59.50	58.47	59.01	58.11	56.71	56.70	**56.00**	51.29

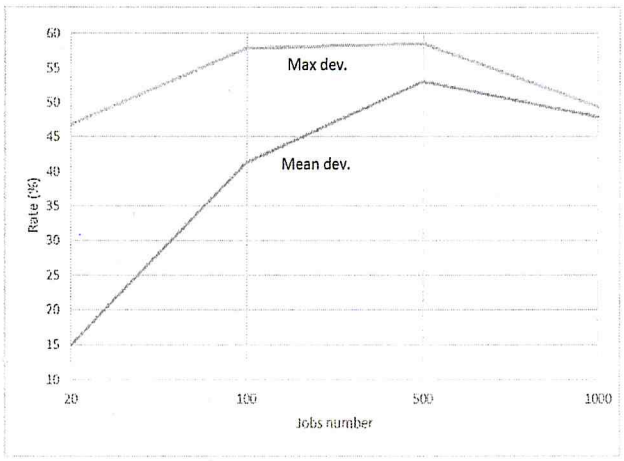
Table 6.5 – *Heuristics Overall Performance for general graph*

Remarks

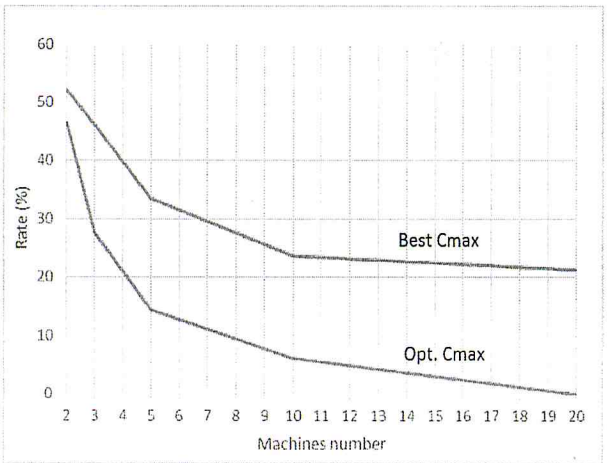
From the table 6.5, one can observe that *HL*13 is globally the best heuristic with regard of the 4 criteria: Best Cmax (36%), optimal Cmax (20%), mean deviation (43%) and maximum deviation (56%). The overall optimal Cmax is around 30%, mean deviation 41% and maximum deviation 51%. On the graphs below, we can observe the variation of all 4 criteria of *HL*13 with regard to jobs number, machines number and graph density.



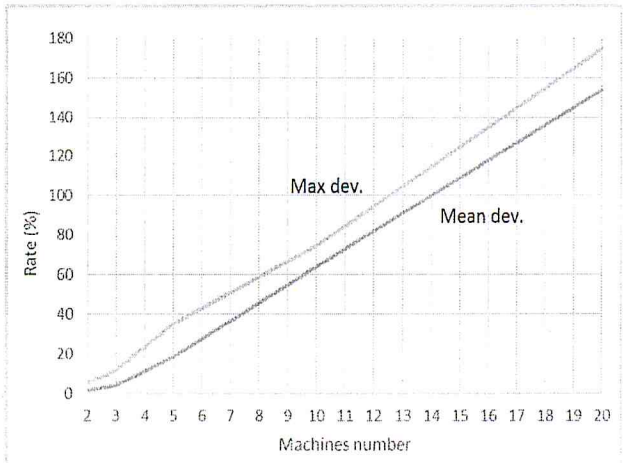
Graph 6.1a – Var. of HL13 Cmax/n



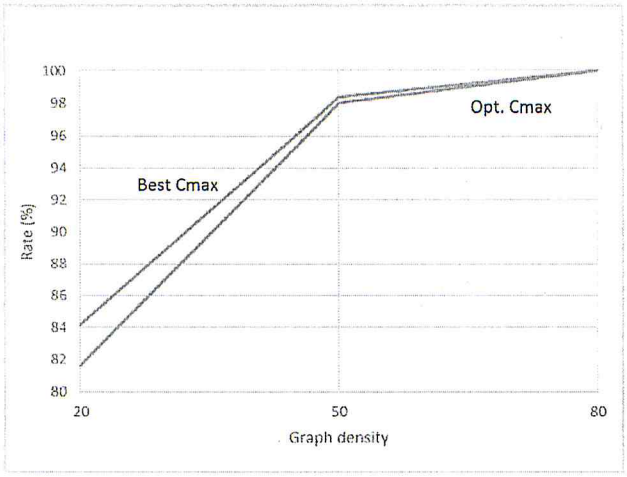
Graph 6.1b – Var. of HL13 dev./n



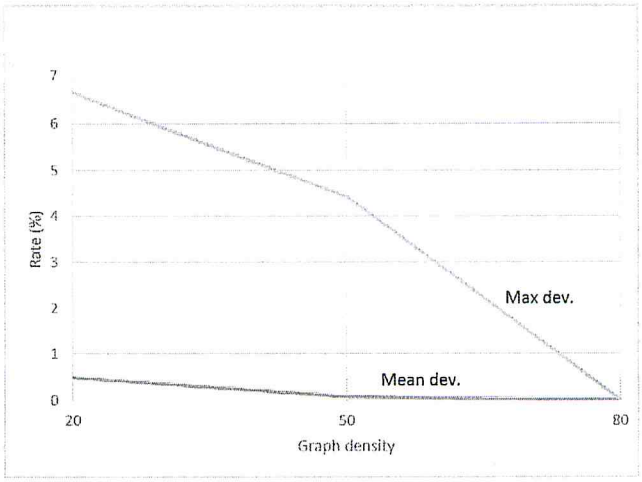
Graph 6.2a – Var. of HL13 Cmax/m



Graph 6.2b – Var. of HL13. dev./m



Graph 6.3a – Var. of HL13 Cmax/d



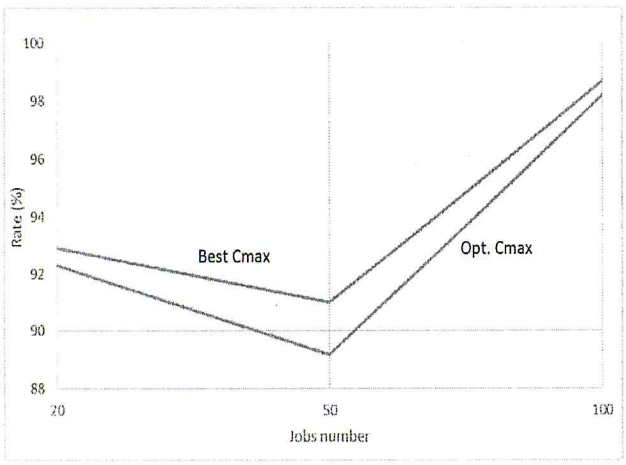
Graph 6.3b – Var. of HL13 dev./d

B. Experiments in the bipartite agreement graph case ($m = 2$)

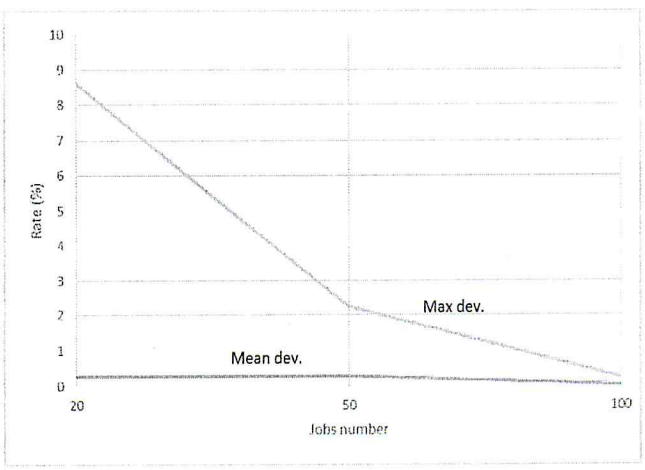
1. Heuristics performance by jobs number

$n = 20$	*HL1*	*HL3*	*HL4*	*HL5*	*HL7*	*HL9*	*HL10*	*HL11*	*HL12*	*HL13*	*Best*
Best Cmax	71.00	69.12	73.10	92.32	90.28	**94.07**	93.00	91.57	91.83	92.92	
Opt. Cmax	70.77	68.95	72.92	91.70	89.72	**93.37**	92.35	90.88	91.23	92.30	98.67
Mean Dev.	1.62	1.82	1.42	0.30	0.37	**0.22**	0.25	0.28	0.27	0.27	0.02
Max Dev.	16.57	16.73	17.13	9.37	8.82	8.73	**7.18**	9.85	7.73	8.62	2.43
$n = 50$	*HL1*	*HL3*	*HL4*	*HL5*	*HL7*	*HL9*	*HL10*	*HL11*	*HL12*	*HL13*	*Best*
Best Cmax	67.17	64.50	68.00	85.50	87.17	88.00	85.67	84.67	86.83	**91.00**	
Opt. Cmax	66.83	64.33	67.83	84.67	86.17	85.83	84.67	83.33	85.67	**89.17**	94.50
Mean Dev.	1.47	1.70	1.35	0.45	0.43	0.40	0.43	0.38	0.37	**0.28**	0.10
Max Dev.	6.88	7.90	6.17	5.98	2.92	4.15	2.82	3.80	2.60	**2.25**	1.40
$n = 100$	*HL1*	*HL3*	*HL4*	*HL5*	*HL7*	*HL9*	*HL10*	*HL11*	*HL12*	*HL13*	*Best*
Best Cmax	78.00	77.00	82.33	92.17	**98.83**	90.83	98.67	91.67	**98.83**	98.67	
Opt. Cmax	77.83	76.83	82.17	92.00	**98.33**	90.67	98.00	91.50	**98.33**	98.17	99.33
Mean Dev.	0.40	0.43	0.30	0.10	**0.02**	0.08	**0.02**	0.07	**0.02**	**0.02**	0.00
Max Dev.	2.87	3.48	2.12	2.03	0.67	1.53	0.57	1.20	0.38	**0.22**	0.12

Table 6.6 – Heuristics Performance by jobs number



Graph 6.4a – Var. of HL13 Cmax/n



Graph 6.4b – Var. of HL13 dev./n

Remarks

We can see that *HL*13 is the best heuristic regardless of jobs number. Best Cmax rate exceeds 90%. The optimality of Cmax is almost always reached either for a small number of jobs or bigger. It happens lightly less often for an average number of jobs (around 50). Globally, the optimality is attained in more than 98% of instances for a small and big value of n ($n = 20$ and $n = 100$), and around 95% for a medium value of n ($n = 50$).

2. Heuristics performance by graph density

$d = 20$	*HL*1	*HL*3	*HL*4	*HL*5	*HL*7	*HL*9	*HL*10	*HL*11	*HL*12	*HL*13	*Best*
Best Cmax	35.57	33.62	39.83	75.32	77.97	78.63	78.85	74.63	79.85	**84.18**	
Opt. Cmax	35.02	33.30	39.48	73.90	76.30	75.80	76.92	72.65	77.92	**81.60**	92.93
Mean Dev.	2.78	3.15	2.48	0.70	0.73	0.58	0.63	0.62	0.58	**0.50**	0.12
Max Dev.	14.50	15.63	13.63	9.07	7.62	7.15	6.90	7.97	6.87	**6.65**	3.30
$d = 50$	*HL*1	*HL*3	*HL*4	*HL*5	*HL*7	*HL*9	*HL*10	*HL*11	*HL*12	*HL*13	*Best*
Best Cmax	83.08	80.23	85.42	95.07	98.32	94.28	**98.48**	94.38	97.67	98.40	
Opt. Cmax	82.90	80.05	85.25	94.87	97.92	94.08	**98.10**	94.18	97.33	98.03	99.57
Mean Dev.	0.65	0.73	0.55	0.15	0.08	0.12	**0.07**	0.12	**0.07**	**0.07**	0.00
Max Dev.	8.25	8.63	8.35	6.02	4.78	6.23	3.67	4.73	**3.58**	4.43	0.65
$d = 80$	*HL*1	*HL*3	*HL*4	*HL*5	*HL*7	*HL*9	*HL*10	*HL*11	*HL*12	*HL*13	*Best*
Best Cmax	97.52	96.77	98.18	99.60	**100.00**	99.98	**100.00**	98.88	99.98	**100.00**	
Opt. Cmax	97.52	96.77	98.18	99.60	**100.00**	99.98	**100.00**	98.88	99.98	**100.00**	100.00
Mean Dev.	0.05	0.07	0.03	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.00
Max Dev.	3.57	3.85	3.43	2.30	**0.00**	1.03	**0.00**	2.15	0.27	**0.00**	0.00

Table 6.7 – Heuristics perforamnce by graph density

Remarks

*HL*13 is again the best heuristic. All the heuristics give better solutions when the graph is thicker. For *HL*13, Best Cmax, Opt Cmax , mean deviation and max deviation are, respectively, 84%, 81%, 0.5% and 6 %. The global opt Cmax rate is about 92% for a graph of 20% of density, it exceeds 99.5% for a graph moderately dense and hit the 100% for 80% of density and beyond.

3. Heuristics performance by processing times range

$p \in [1, 10]$	*HL*1	*HL*3	*HL*4	*HL*5	*HL*7	*HL*9	*HL*10	*HL*11	*HL*12	*HL*13	*Best*
Best Cmax	75.46	72.93	78.97	88.19	91.11	90.02	90.90	87.87	91.32	**92.92**	
Opt. Cmax	75.01	72.63	78.67	87.51	90.30	88.73	89.74	86.81	90.20	**91.78**	96.63
Mean Dev.	0.96	1.13	0.77	0.38	0.36	0.30	0.31	0.30	0.28	**0.24**	0.06
Max Dev.	8.41	8.89	7.90	6.53	4.98	4.84	**3.89**	4.74	4.24	4.54	1.62
$p \in [30, 50]$	*HL*1	*HL*3	*HL*4	*HL*5	*HL*7	*HL*9	*HL*10	*HL*11	*HL*12	*HL*13	*Best*
Best Cmax	68.66	67.48	69.99	91.80	93.08	91.91	93.99	90.73	93.68	**95.47**	
Opt. Cmax	68.61	67.44	69.94	91.40	92.51	91.18	93.60	90.33	93.29	**94.64**	98.37
Mean Dev.	1.37	1.50	1.28	0.19	0.19	0.17	0.16	0.19	0.16	**0.13**	0.02
Max Dev.	9.13	9.86	9.04	5.06	3.29	4.77	3.16	5.16	2.90	**2.84**	1.01

Table 6.8 – Heuristics Performance by processing times range

Remarks

HL13 is the best heuristic for both processing times intervals, and is better for big values. The global optimality is met more often for big values of p_i.

4. Bipartite overall heuristics performance

Bipartite	*HL1*	*HL3*	*HL4*	*HL5*	*HL7*	*HL9*	*HL10*	*HL11*	*HL12*	*HL13*	*Best*
Best Cmax	72.06	70.21	74.48	89.99	92.09	90.97	92.44	89.30	92.50	**94.19**	
Opt. Cmax	71.81	70.04	74.31	89.46	91.41	89.96	91.67	88.57	91.74	**93.21**	97.50
Mean Dev.	1.16	1.32	1.02	0.28	0.27	0.23	0.23	0.24	0.22	**0.19**	0.04
Max Dev.	8.77	9.37	8.47	5.79	4.13	4.81	**3.52**	4.95	3.57	3.69	1.32

Table 6.9 – Heuristics bipartite overall Performance

Remarks

As in the general agreement graph case, *HL13* is also the best heuristic regarding of all 4 criteria. The global Cmax is optimal at a rate of 97 %.

As in the general graph, we can observe graphically the behavior of *HL13* through the 4 criteria compared with the jobs number, machines number and graph density.

5. Overall heuristics performance

All	*HL1*	*HL3*	*HL4*	*HL5*	*HL7*	*HL9*	*HL10*	*HL11*	*HL12*	*HL13*	*Best*
Best Cmax	44.62	41.88	55.77	49.69	52.36	50.57	62.02	55.74	63.60	**65.21**	
Opt. Cmax	41.80	39.88	48.70	45.81	47.72	46.24	55.52	50.23	55.76	**56.56**	63.58
Mean Dev.	23.25	23.52	22.99	22.78	22.69	22.64	22.45	22.12	22.01	**21.84**	20.52
Max Dev.	34.00	34.39	33.40	32.65	31.30	31.91	30.81	30.83	30.14	**29.84**	26.30

Table 6.10 – Heuristics overall performance

Remarks

As a conclusion, *HL13* is the best ever heuristic in average at 65% for best Cmax, 56% for opt Cmax, 22% for mean deviation and 26% for maximum deviation. Globally, the optimality is reached at 63%. The global mean and maximum deviation are respectively 20% and 26%.

II. Metaheuristics and local search experiments

Local search methods and metaheuristics on which experiments have been performed are: Pairwise Interchange (*PI*), Adjacent Pairwise Interchange (*API*), Insertion Method (*IM*), Simulated Annealing (*SA*) and Harmony Search (*HS*).

The parameters used for the configuration of the metaheuristics are:

- *LB*: Lower Bound (chap. 4 and 5)
- *UB*: Upper Bound, minimum value yielded by the heuristics (chap. 5)
- *Av dev/LB* : Average deviation of the heuristics compared with the lower bound calculated as: $100x(H - LB)/LB$
- *Av dev/UB* : Average deviation of the heuristics compared with the upper bound calculated as: $100x(UB - H)/UB$

- *Av UB/LB*: Average deviation of the upper bound compared with the lower bound calculated as: $100x(UB - LB)/LB$
- *Inst*: Instances number
- m, n: Respectively machines number and jobs number
- d: Graph density
- For the 4 first methods, we have set the maximum iterations number without improvement as a stop criterion to 2000
- For the simulated annealing, the parameter beta is set to 2
- For the harmony search, the parameters are set as follows: harmony memory size ($HMS = 10$), harmony memory considering rate ($HMCR = 0.95$), pitch adjustment rate ($PAR = 0.2$), distance bandwidth ($bw = 1$) and the number of improvisations ($NI = 3000$).

A. General agreement graph case

$n = 20$ (*total of* 50 *instances x* 8)

$m = 2$ $d = 30$	$p_i \in [1,20]$			*AD UB/LB*: 1.90%		$p_i \in [30,50]$			*AD UB/LB*: 1.04%	
	PI	***API***	***IM***	***SA***	***HS***	***PI***	***API***	***IM***	***SA***	***HS***
Av dev/LB	0.024	0.024	0.024	0.024	0.024	0.099	0.099	0.099	0.104	0.099
Av dev/UB	1.390	1.294	1.388	1.266	1.390	0.617	0.593	0.626	0.533	0.616
$m = 2$ $d = 50$	$p_i \in [1,20]$			*AD UB/LB*: 0.17%		$p_i \in [30,50]$			*AD UB/LB*: 0.31%	
	PI	***API***	***IM***	***SA***	***HS***	***PI***	***API***	***IM***	***SA***	***HS***
Av dev/LB	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Av dev/UB	0.165	0.165	0.165	0.165	0.165	0.303	0.303	0.303	0.303	0.303
$m = 3$ $d = 30$	$p_i \in [1,20]$			*AD UB/LB*: 8.02%		$p_i \in [30,50]$			*AD UB/LB*: 14.72%	
	PI	***API***	***IM***	***SA***	***HS***	***PI***	***API***	***IM***	***SA***	***HS***
Av dev/LB	0.00	0.00	0.00	0.00	0.00	0.42	0.42	0.35	0.42	0.42
Av dev/UB	2.21	2.12	2.30	1.87	2.11	4.10	3.59	3.75	3.63	4.05
$m = 3$ $d = 50$	$p_i \in [1,20]$			*AD UB/LB*: 7.05%		$p_i \in [30,50]$			*AD UB/LB*: 5.31 %	
	PI	***API***	***IM***	***SA***	***HS***	***PI***	***API***	***IM***	***SA***	***HS***
Av dev/LB	0.006	0.006	0.006	0.009	0.003	0.066	0.073	0.051	0.044	0.044
Av dev/UB	4.412	3.853	4.171	4.275	4.570	3.544	3.053	3.225	3.021	3.463

Table 6.11 – *General graph*, $n = 20$

Remarks

❖ $m = 2$,

- $d = 30\%$,

For $I2$, LB is very close to UB ($av.dev.UB/LB = 1.04\%$) and all methods give the same value except SA, but for $I1$, UB is less close to LB and all metaheuristics are the same.

- $d = 50\%$,

All methods reach the optimality either for $I1$ or for $I2$.

❖ $m = 3$,

- $d = 30\%$,

For $I2$, the deviation of UB is 8% and IM is the best, but for $I1$, the deviation of UB is 14% and all metaheuristics are optimal.

- $d = 50\%$,

HS is the best for $I1$ (0.003% of deviation) and both SA and HS are the best for $I2$ (0.044% of deviation).

$n = 100$ (*total of* 30 *instances x* 10)

$m = 2$ d =30	$p_i \in [1,20]$			AD UB/LB: 0%		$p_i \in [30,50]$			AD UB/LB: 0.08%	
	PI	***API***	***IM***	***SA***	***HS***	***PI***	***API***	***IM***	***SA***	***HS***
Av dev/LB	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Av dev/UB	0.000	0.000	0.000	0.000	0.000	0.080	0.080	0.080	0.080	0.080
$m = 3$ d =30	$p_i \in [1,20]$			AD UB/LB: 10.27%		$p_i \in [30,50]$			AD UB/LB: 8.03%	
	PI	***API***	***IM***	***SA***	***HS***	***PI***	***API***	***IM***	***SA***	***HS***
Av dev/LB	0.174	0.235	0.220	0.240	0.275	0.217	0.148	0.195	0.228	0.235
Av dev/UB	4.027	3.772	3.552	3.097	1.172	3.246	2.987	3.068	2.436	1.248
$m = 3$ d =50	$p_i \in [1,20]$			AD UB/LB: 0.14%		$p_i \in [30,50]$			AD UB/LB: 0.46%	
	PI	***API***	***IM***	***SA***	***HS***	***PI***	***API***	***IM***	***SA***	***HS***
Av dev/LB	0.000	0.000	0.000	0.014	0.015	0.000	0.004	0.000	0.011	0.001
Av dev/UB	0.141	0.141	0.128	0.085	0.056	0.453	0.446	0.457	0.367	0.367
$m = 5$ d =30	$p_i \in [1,20]$			AD UB/LB: 46.80%		$\boldsymbol{p_i \in [30,50]}$			AD UB/LB: 43.21%	
	PI	***API***	***IM***	***SA***	***HS***	***PI***	***API***	***IM***	***SA***	***HS***
Av dev/LB	2.143	2.313	2.034	2.206	2.235	1.322	1.335	1.228	1.390	1.363
Av dev/UB	4.918	4.799	4.585	3.912	2.883	4.748	5.106	4.579	4.877	3.591
$m = 5$ d =50	$p_i \in [1,20]$			AD UB/LB: 9.62%		$p_i \in [30,50]$			AD UB/LB: 8.20%	
	PI	***API***	***IM***	***SA***	***HS***	***PI***	***API***	***IM***	***SA***	***HS***
Av dev/LB	0.376	0.265	0.290	0.384	0.431	0.253	0.206	0.196	0.210	0.402
Av dev/UB	3.708	3.378	3.653	2.834	1.157	3.731	3.718	3.455	2.946	1.912

Table 6.12 – *General graph*, n=100

Remarks

❖ **$m = 2$,**

- $d = 30\%$,

For $I2$, LB is very close to UB ($av.dev.UB/LB = 0.08\%$) and all methods give the optimal value. For $I1$, UB is optimal for all instances.

❖ **$m = 3$,**

- $d = 30\%$,

For $I2$, PI is the best metaheuristic, but for $I1$, API is the best

- $d = 50\%$,

HS is the best for $I1$ (0.003% of deviation) and both SA and HS are the best for $I2$ (0.044 of deviation).

❖ **$m = 5$,**

- $d = 30\%$,

IM is the best metaheuristic (2.034% for I1 and 1.228% for I2), and the average deviation of UB is 46% for I1 and 43% for I2.

- $d = 50\%$,

API is the best metaheuristic for I1 (2.265%) and IM for I2 (0.196%).
The average deviation of UB is 9.62 % for I1 and 8.2 % for I2.

$n = 500$ (*total of* 10 *instances x* 4)

$m = 5$ $p_i \in [1,20]$	$d = 30$		*AD UB/LB*: 9.81%			$d = 50$		*AD UB/LB*: 0.28%		
	PI	***API***	***IM***	***SA***	***HS***	***PI***	***API***	***IM***	***SA***	***HS***
Av dev/LB	0.899	0.907	0.934	0.897	0.998	0.011	0.021	0.021	0.032	0.059
Avr dev/UB	1.743	1.790	1.773	1.456	0.025	0.179	0.161	0.161	0.028	0.000
$m = 10$ $p_i \in [1,20]$	$d = 30$		*AD UB/LB*: 112.9%			$d = 50$		*AD UB/LB*: 41.59%		
	PI	***API***	***IM***	***SA***	***HS***	***PI***	***API***	***IM***	***SA***	***HS***
Av dev/LB	12.238	11.844	12.199	12.050	12.462	4.314	4.240	4.211	4.277	4.545
Av dev/UB	2.438	2.349	1.729	1.826	0.062	1.750	2.089	1.555	1.662	0.277

Table 6.13 – *Gen*eral *graph*, $n = 500$

Remarks

- ❖ $m = 5$,
  - $d = 30\%$,

  Best metaheuristic: *SA* (0.897%), *Av dev UB/LB*: 9.81%
  - $d = 50\%$,

  Best metaheuristic: *PI* (0.011%), *Av dev UB/LB*: 0.28%
- ❖ $m = 10$,
  - $d = 30\%$,

  Best metaheuristic: *API* (11.844%), *Av dev UB/LB*: 112%
  - $d = 50\%$,

  Best metaheuristic: *IM* (4.211%), *Av dev UB/LB*: 41%

B. Bipartite agreement graph case ($m = 2$)

$n = 50$ (30 *instances x*2)

$d = 10$	$p_i \in [1,20]$ |		*AD UB/LB*: 0.28%			$p_i \in [30,50]$ |		*AD UB/LB*: 0.54%		
	PI	***API***	***IM***	***SA***	***HS***	***PI***	***API***	***IM***	***SA***	***HS***
Av dev/LB %	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Av dev/UB %	0.28	0.28	0.28	0.20	0.28	0.52	0.51	0.50	0.42	0.50

Table 6.14 – *Bip*artite *graph*, $n = 50$

$n = 100$ (30 *instances x*2)

d =10	$p_i \in [1,20]$ |		*AD UB/LB*: 1.06%			$p_i \in [30,50]$ |		*AD UB/LB*: 0.26%		
	PI	***API***	***IM***	***SA***	***HS***	***PI***	***API***	***IM***	***SA***	***HS***
Av dev/LB %	0.00	0.00	0.00	0.01	0.00	0.00	0.02	0.00	0.00	0.02
Av dev/UB %	0.97	0.81	0.89	0.57	0.38	0.26	0.21	0.25	0.14	0.05

Table 6.15 – *Bip*artite *graph*, $n = 100$

Remarks

In the bipartite agreement graph case, beyond 10% of graph density, the lower bound is closely approached by the heuristics, even mostly reached. In case where the upper bound don't hit the lower one, all the metaheuristics reach rapidly the optimum. The tables above show, for 10% of graph density, the average deviation of the metaheuristics. It seems to be useless to consider the cases of a graph density which are greater than 10% because there would be zeros everywhere in the results tables. The same thing happens when the jobs number is less than 50 or greater than 100.

III. Exact method experiments, Cplex implementation

A. General agreement graph case

1. Performance depending on *LB* and *UB*

For the exact resolution of the *SWA* problem, we have adopted the Cplex 12.6 software. Pretests have shown that the resolution of a classic formulation cannot be obtained in a reasonable time when the jobs number exceeds 16. So we introduce, in the formulation, two new constraints involving the lower and the upper bound as defined in chapter 4.

Beside the formulation $F1$ of chapter 4, we define 2 other formulations as follows.

$$F1a \equiv \begin{cases} F1 \\ C_{max} \geq LB \end{cases} \qquad \text{and} \qquad F1b \equiv \begin{cases} F1 \\ C_{max} \geq LB \\ C_{max} \leq UB \end{cases}$$

Where LB is the lower bound (chapter 4), and UB is the upper bound obtained by the best of the 10 heuristics (Chapter 5).

We have generated 10 instances for a configuration of 2 machines, $p_i \in [1, 20]$ and a general agreement graph of 50% of density. 3 formulations are used: $F1$, $F1a$ and $F1b$. The mean running times obtained are shown in the tables and graphs below.

Jobs number n	10	11	12	13	14
Mean time (sec.)	4.23	10.52	129.44	297.27	1147.00

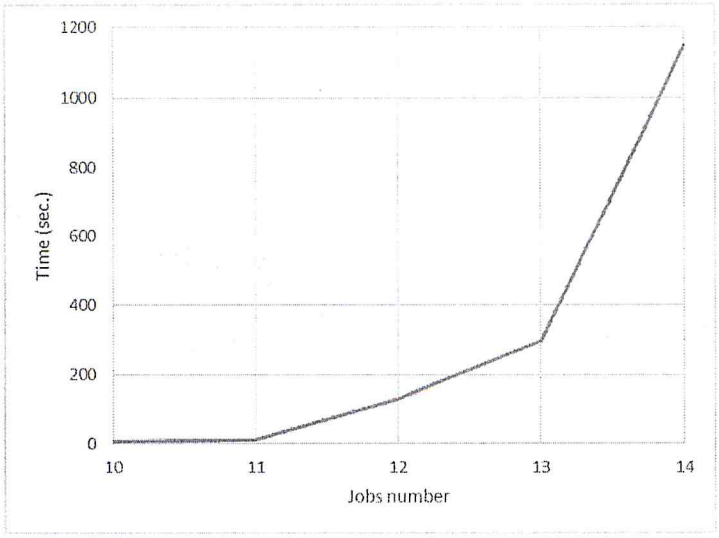
Table 6.16 – *Formulation* $F1$

Jobs number n	12	16	20	24	28	32	36	40
Mean time (sec.)	5.32	3.24	6.88	64.97	36.35	110.77	329.86	990.49

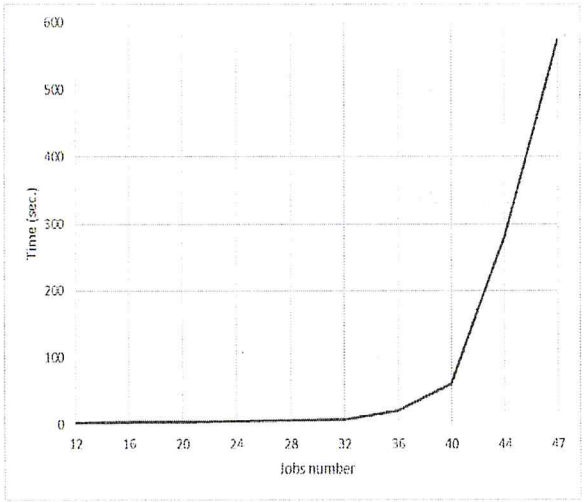
Table 6.17 – *Formulation* $F1a$

Jobs number n	12	16	20	24	28	32	36	40	44	47
Mean time (sec.)	3.19	3.52	3.89	5.31	6.61	7.80	20.76	60.28	279.31	574.59

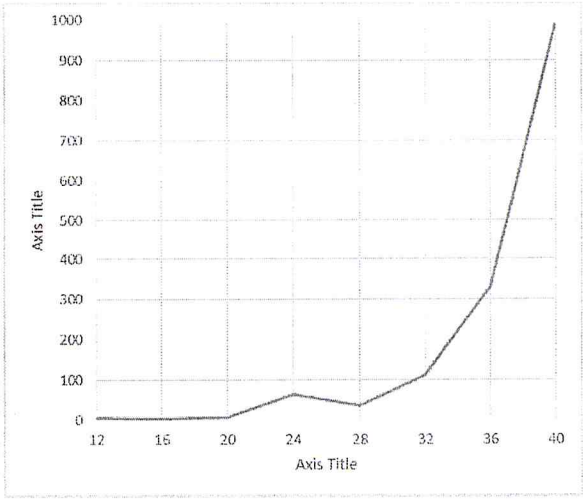
Table 6.18 – *Formulation* $F1b$



Graph 6.5 – formulation F1



Graph 6.6 – formulation F1b



Graph 6.7 – formulation F1a

Remarks

We can see that, for this configuration, the problem resolution in a reasonable time is possible for a maximum of 14 jobs, but by inserting the constraint involving the lower bound, the maximum number of jobs for which the system is solvable is 3 times greater, reaching 40. And this number is extended to 47 if, in addition, we insert the constraint corresponding to the upper bound.

2. Performance depending on the graph density

To perform experiments depending of graph density, we use the following configuration: $m = 2, p_i \in [1,50]$ and $d \in \{10\%, 30\%, 50\%, 70\%, 90\%\}$. We produce randomly 5 instances in each case corresponding to each density value. We obtain the results shown in the table and graphs below.

Remarks

The maximum number of jobs for the original formulation F1 attains the maximum of 17 jobs at around 30% of graph density. It decreases when the graph becomes more or less dense.

b. Using formulation F1a

$d = 10\%$

Jobs number n	10	15	17
Mean time (sec.)	3.06	49.75	168.67

Table 6.20a – $d = 10\%$

$d = 30\%$

Jobs number n	10	15	20
Mean time (sec.)	2.42	7.36	27.91

Table 6.20b – $d = 30\%$

$d = 50\%$

Jobs number n	10	15	20	25	28	29
Mean time (sec.)	5.77	7.81	10.33	21.79	31.40	72.79

Table 6.20c – $d = 50\%$

$d = 70\%$

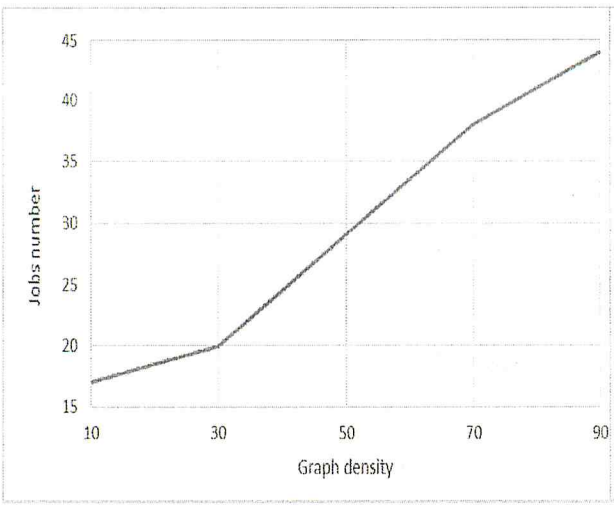
Jobs number n	10	15	18	20	25	30	35	38
Mean time (sec.)	18.28	3.63	5.82	7.08	24.15	147.01	54.87	769.11

Table 6.20d – $d = 70\%$

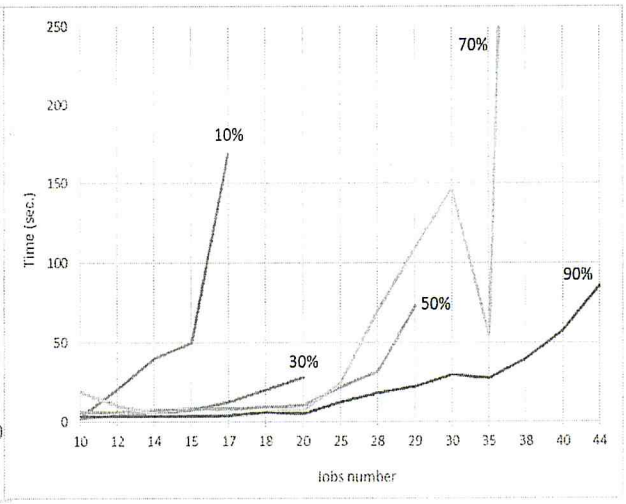
$d = 90\%$

Jobs number n	10	15	18	20	25	30	35	40	44
Mean time (sec.)	3.22	3.47	5.90	5.14	12.39	29.46	27.16	57.65	85.76

Table 6.20e – $d = 90\%$



Graph 6.10 – Max jobs/graph density



Graph 6.11 – Run. time./n (all d) - F1a

B. Bipartite agreement graph case

Comparison between *F1* and *F2* depending on graph density

We consider the *SWA* problem where $p_i \in [1,50]$ and the agreement graph is of bipartite type (thus $m = 2$). The aim is to find the maximum jobs number for which the system is solvable in a reasonable time, and compare the results for the two formulations *F1* (used for a general graphs) and *F2* (used for bipartite graphs), so we generate 5 instances and consider the average for each of 5 graph density values. We obtain these results.

$d = 10\%$	$n = 10$		$n = 11$		$n = 12$	
Formulation	*F1*	*F2*	*F1*	*F2*	*F1*	*F2*
Mean time (sec.)	8.28	10.58	196.69	186.73	777.50	791.10

Table 6.21a – $d = 10\%$

$d = 30\%$	$n = 10$		$n = 11$		$n = 12$		$n = 13$		$n = 14$	
Formulation	*F1*	*F2*	*F1*	*F2*	*F1*	*F2*	*F1*	*F2*	*F1*	*F2*
Mean time	2.35	2.28	9.14	7.72	5.14	6.36	173	169	121	157

Table 6.21b – $d = 30\%$

$d = 50\%$	$n = 10$		$n = 11$		$n = 12$		$n = 13$		$n = 14$		$n = 15$		$n = 16$	
Formulation	*F1*	*F2*	*F1*	*F2*	*F1*	*F2*	*F1*	*F2*	*F1*	*F2*	*F1*	*F2*	*F1*	*F2*
Mean time	3.03	3.16	2.64	2.61	14.74	8.59	30.44	18.60	585	559	373	187	6806	1379

Table 6.21c – $d = 50\%$

$d = 70\%$	$n = 10$		$n = 11$		$n = 12$		$n = 13$		$n = 14$		$n = 15$	
Formulation	*F1*	*F2*	*F1*	*F2*	*F1*	*F2*	*F1*	*F2*	*F1*	*F2*	*F1*	*F2*
Mean time	2.20	2.16	7.62	6.37	8.17	11.13	58	51	86	74	223	266

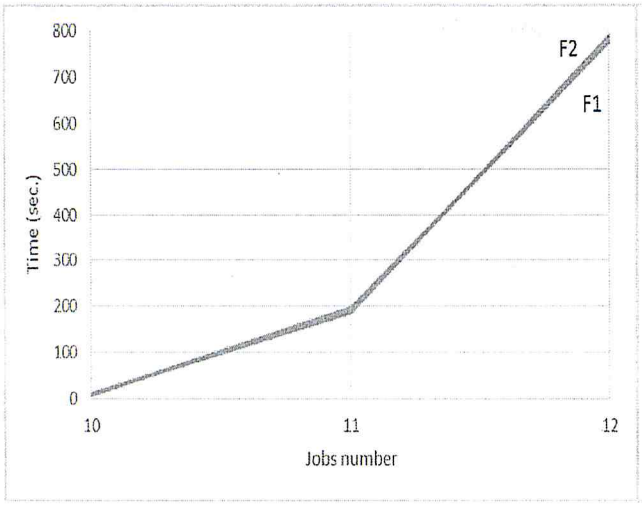
Table 6.21d – $d = 70\%$

$d = 90\%$	$n = 10$		$n = 11$		$n = 12$		$n = 13$		$n = 14$	
Formulation	*F1*	*F2*	*F1*	*F2*	*F1*	*F2*	*F1*	*F2*	*F1*	*F2*
Mean time	3.14	2.53	5.51	5.95	3.77	3.05	61	90	771	472

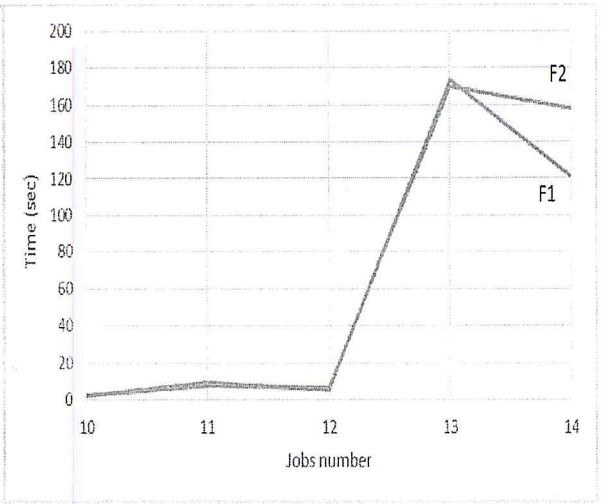
Table 6.21e – $d = 90\%$

Remarks

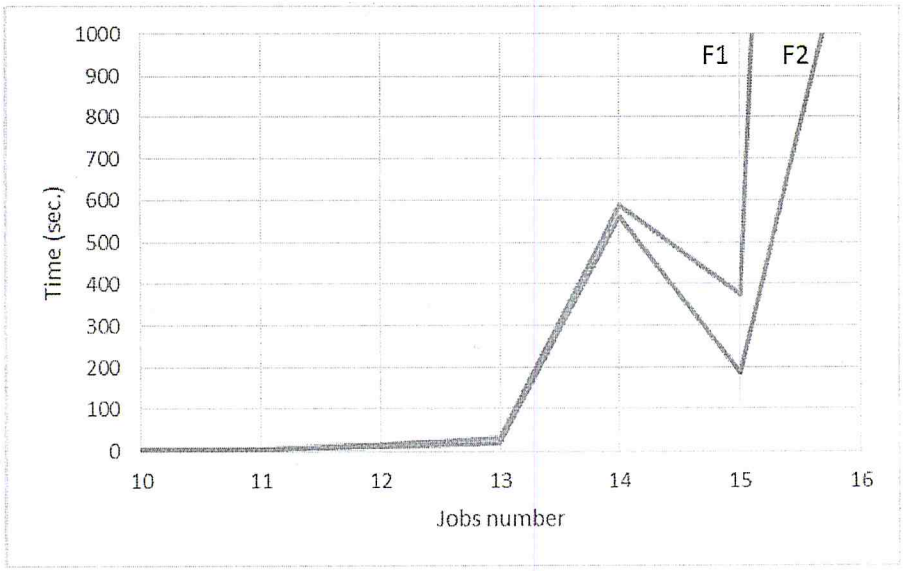
1. The maximum jobs number for which the system is solvable in a reasonable time is the same for the two formulations, and is at the maximum for a graph of medium density. This number decreases when the graph becomes more or less thick.
2. The formulation F2 is much more efficient than F1 for a graph of medium density, and particularly for big values of jobs number. All this remarks are illustrated in the graphs below.



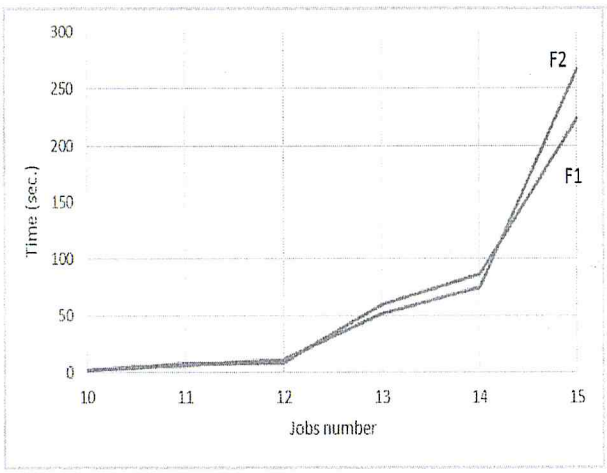
Graph 6.12a – d=10%



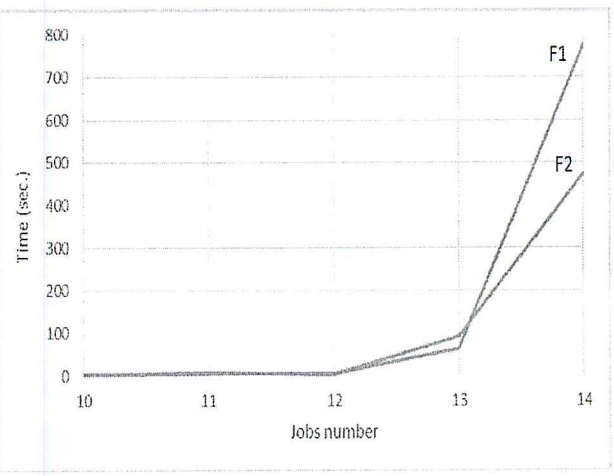
Graph 6.12b – d=30%



Graph 6.12c – d=50%



Graph 6.12d – d=70%



Graph 6.12e – d=90%

CONCLUSION

In this thesis, we have approached the scheduling with agreements problem, where a graph, called agreement graph, models the agreement between jobs. We have considered different types of agreement graph: general graphs, bipartite graphs, trees and chains. This problem is NP-hard for the general agreement graph case. However, it can be solved in a reasonable time for special classes of graphs. Indeed, we have devised an algorithm solving in a polynomial time the problem in the case of chain-type agreement graph. This algorithm is built on the maximum weight stable set to determine a lower bound for the problem. An efficient algorithm to solve the latter in a polynomial time is given for the bipartite agreement graph case.

A mathematical formulation has been proposed in both general and bipartite graphs with and without insertion of constraints involving lower and upper bounds.

To approach the optimality and obtain an upper bound for the problem, 14 heuristics, based on lists, have been introduced whose one seems to stand out from the rest. And, to be closer to the optimal solution, we have proposed 5 metaheuristics and local search methods taking their initial solutions from the best heuristics.

All these methods have been widely experimented using uniformly-generated instances. Empirical results are analyzed, compared and graphically represented.

REFERENCES

1. Baker B.S., E.G. Coffman. Mutual Exclusion Scheduling. Theoretical Computer Science 1996; 162: 225-243.
2. Bendraouche M. Ordonnancement sur Machines Identiques avec Graphe de Concordance. Thèse de Doctorat. USTHB. Décembre 2011. 02/2011 - D/MT.
3. Bendraouche M., M. Boudhar. Scheduling jobs on identical machines with agreement graph. Computers and Operations Research 2012; 39 382–390.
4. Bendraouche Mohamed & Mourad Boudhar (2015): Scheduling with agreements: new results, International Journal of Production Research, DOI:10.1080/00207543.2015.1073860.
5. Bendraouche Mohamed, Mourad Boudhar, Ammar Oulamara. Scheduling: Agreement graph vs resource constraints. European Journal of Operational Research 240 (2015) 355–360.
6. Blazewicz J. Complexity of computer scheduling algorithms under resource constraints. Proc. I Meeting AFCET-SMF on Applied Mathematics, Palais eau (France) 1978; 169-178.
7. Blazewicz J., J.K Lenstra, A.H.G Rinnooy Kan. Scheduling subject to resource constraints, classification and complexity. Discrete Applied Mathematics 1983; 5 No 1:11-24. 92
8. Blazewicz Jacek. Moshe Dror. Jan Weglarz. Mathematical programming formulations for machine scheduling: A survey. European Journal of Operational Research 51 (1991) 283-300 North-Holland.
9. Bodlaender H.L., K. Jansen, and G.J.Woeginger. Scheduling with incompatible jobs. Discrete Applied Mathematics 1994, 55: 219-232.
10. Brenda S. Baker, Edward G. Coffman, Jr. Mutual exclusion scheduling. Theoretical Computer Science 162 (1996) 225-243
11. Brucker Peter. Sigrid Knust. Complex Scheduling. Cataloging-in-Publication Data Library of Congress Control Number: 2005938499.
12. Chen Bo Chris N. Potts. Gerhard J. Woeginger. A Review of Machine Scheduling: Complexity, Algorithms and Approximability. Handbook of Combinatorial Optimization D.-Z. Du and P.M. Pardalos (Eds.) pp. 21-169 c 1998 Kluwer Academic Publishers.
13. Chen Jing, Liu Guang-Liang, LU Ran. Discrete Harmony Search Algorithm for Identical Parallel Machine Scheduling Problem. Proceedings of the 30th Chinese Control Conference July 22-24, 2011, Yantai, China.
14. Even G., M.M. Halldorson, L. Kaplan, D. Ron. Scheduling with conflicts: online and offline algorithms. Journal of scheduling 2009; 12: 199-224.
15. Faigle Ulrich, Gereon Frahling. A combinatorial algorithm for weighted stable sets in bipartite graphs. Discrete Applied Mathematics 154 (2006) 1380 – 1391.
16. Fatemi S. M. T. Ghomi & F. Jolai Ghazvini (1998). A pairwise interchange algorithm for parallel machine scheduling, Production Planning & Control: The Management of Operations, 9:7, 685-689, DOI: 10.1080/095372898233687

17. Gardi F. Mutual exclusion scheduling with interval graphs or related classes Part I. Discrete Applied Math 2009; 157: 19-35.
18. Garey M.R. and R.L Graham. Bounds for multiprocessor scheduling with resource constraints. SIAM Journal on Computing 1975; 4: 187–200.
19. Garey M.R., D.S. Johnson. Complexity results for multiprocessing scheduling under resource constraints. SIAM J. on Computing 1975; 4: 397-411.
20. Golumbic M.C. Algorithmic Graph Theory and Perfect Graphs. Computer Science and Applied Mathematics Series, Academic Press, New York, NY 1980.
21. Graham R.E., E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling : a survey, Ann. Discrete Math. 1979; 4, 287-326.
22. Graham R.L. Bounds for certain multiprocessing anomalies. Bell System Technical Journal 1966; 45:1563-1581.
23. Gupta J. N. D. & A. J. Ruiz-Torres (2001) A LISTFIT heuristic for minimizing makespan on identical parallel machines, Production Planning & Control: The Management of Operations, 12:1, 28-36, DOI: 10.1080/09537280150203951.
24. Hansen P., A. Hertz, J. Kuplinski. Bounded vertex colorings of graphs. Discrete Math 1993; 111: 305-312.
25. IRANI S. and V. LEUNG. Scheduling with Conflicts on Bipartite and Interval Graphs. Journal of Scheduling 6: 287-307, 2003.
26. Jiang Hua, Liping Zheng, Yun Bao, and Yanxiu Liu. Independent Task Scheduling by Hybrid Algorithm of Harmony Search and Variable Neighborhood Search. College of Computer Science, Liaocheng University, Shandong Liaocheng, P. R. China 25205 China.
27. Kirkpatrick S., C. D. Gelatt, Jr., M. P. Vecchi. Optimization by Simulated Annealing. Science. 13 May 1983, Volume 220, Number 4598
28. Lee Wen-Chiung. Chin-Chia Wu. Peter Chen. A simulated annealing approach to makespan minimization on identical parallel machines. Int J Adv Manuf Technol (2006) 31: 328–334.
29. Lenstra J.K., A.H.G. Rinnooy Kan and P. Brucker. Complexity of machine scheduling problems. Studies in integer programming (Proc. Workshop, Bonn, 1975). Ann. Of Discrete Math 1977; 1, 343-362, North Holland, Amsterdam.
30. Sakai S., M. Togasaki, K. Yamazaki. A note on greedy algorithms for maximum weighted independent set problem. Discrete Applied Mathematics 2003; 126: 313- 322.
31. Sakarovitch M. Optimisation combinatoire: Théorie des graphes. Hermann, Paris, 1984. 95
32. Sevkli Mehmet, Hatice Uysae. A Modified Variable Neighborhood Search for Minimizing the Makespan on Identical Parallel Machines.
33. Wang Xiaolei. Xiao-Zhi Gao. Kai Zenger. An Introduction to Harmony Search Optimization Method. Springer Briefs in Applied Sciences and Technology. Computational Intel.