Scheduling jobs within time windows on identical parallel machines: New model and algorithms

Virginie Gabrel

LAMSADE – University of Paris Dauphine, Place du Mal de Lattre de Tassigny, 75775 Paris Cedex 16, France

Abstract

This article analyses the problem of scheduling non-preemptive jobs processed within time windows on \( k \) identical parallel machines. Each job can be completed on a sub-set of machines. The problem of determining if a particular set of jobs can be completed by the available machines is \( \text{NP} \)-complete. A new model and heuristics are proposed to solve this problem in two particular cases: first, the case in which each job has to be completed at fixed start and end times; second, the case in which each job can be completed within a time window larger than its processing time. Our approach deals with graph theory. It is based primarily on the independent set and partition into cliques concepts.

Keywords: Fixed job scheduling; Graphs; Heuristics

0. Introduction

Parallel-machine scheduling problems include a large number of different problems (see [3] for a complete state of the art). Parameters of these problems are related to job characteristics (preemptive, precedence constraints, due date and ready time . . .), to multiple-machine environment characteristics (identical or non-identical serial machines, identical, uniform or unrelated parallel machines, flow shop, job shop, open shop) and to optimality criteria considered (flow time, maximum lateness, total tardiness, makespan). We focus on the problem of scheduling on \( k \) identical parallel machines \( m \) non-preemptive jobs each with a given processing time and an interval for the start time.

Two variations of this scheduling problem are considered: for the first, each job has fixed start and end times (the interval for each job's start time is a point); for the second, each job can be completed within a time window larger than its processing time. Moreover for both, we consider an additional constraint: each job can be processed only on a sub-set of machines. Clearly, the main issue we are dealing with here is: given \( m \) jobs with given processing times, time interval for each job's start time and a particular sub-set of available machines to complete each job, is it possible to carry out all jobs once (and only once) on \( k \) identical parallel machines, such that there is at most one job at a time performed on each machine? When the interval for each job's start time is a point, the problem is called the Fixed
job Scheduling Problem (FSP); otherwise it is called the Variable job Scheduling Problem (VSP).

FSP and VSP have been studied by various authors (see [1,5,11,15]). These problems have practical applications, for example, in airport aircraft maintenance process (see [14,16]) or in drawing up schedules for bus-drivers (see [7]). We are interested in a new application for planning the production of low-orbit Earth sensing satellites (see [8,9]).

FSP is NP-complete when each job can be performed by at least three of the $k$ available machines (see [1]). To solve FSP in this case, a new modelling based on graph theory is proposed. In this model FSP can be formulated as a maximum independent set problem. The proposed model presents some properties which enable us to define efficient algorithms to set bounds for the maximum number of jobs that can be performed, and thus to solve the FSP in an approximate way. We follow this approach to solve instances of FSP in the context of production planning for low-orbit Earth sensing satellites. Our computational results show that simple heuristics proposed to solve FSP in this context are very promising. When the proposed approach is followed to solve FSP in a general context, the results are not as good, but still satisfactory. Furthermore, proposed model and algorithms can be used to solve VSP.

In this paper, we will first consider FSP in providing problem definitions and background. Then proposed model and heuristics (based on partition into cliques) for FSP are presented. After having presenting the computational results, we will specify how to apply the previous results for solving VSP.

1. Main results about scheduling jobs with fixed start and end times

Let us consider the problem of scheduling non-preemptive jobs with fixed start and end times on identical parallel machines. Associated with each job, there is a sub-group of machines that can complete it (this specifies a job-machine mapping). Each job has to be processed once or not at all, and a machine can perform at most one job at a time.

Given $m$ non-preemptive jobs to be completed on $k$ identical parallel machines at fixed start and end times and a job-machine mapping, the decision problem we are interested in, called FSP, is the following: is it possible to complete all the jobs? Arkin and Silverberg have proved in [1] that FSP is NP-complete when the number of machines is not fixed; otherwise this problem becomes tractable. In this polynomial complexity case, the authors proposed an $O(m^{k+1})$ algorithm based on a dynamic programming formulation. Obviously, this algorithm becomes rapidly inefficient as $k$ increases.

When the number of machines is not fixed and depends on FSP instances, we proved in [9] that FSP is also tractable in polynomial time if and only if each job can be processed on, at most, two machines.

The purpose of our study is to define approximation algorithms to solve FSP in NP-Complete cases, that is to say when the number of machines depends on the instances considered and the job-machine mapping contains more than two machines per jobs.

Practical problems that lead to solving FSP instances appear in the aircraft maintenance process problem at an airport (see [14,16]), in the bus driver scheduling problem (see [7]) or in air-traffic control (see [4]) – in Section 5, we discuss this last application. In production planning problem for low-orbit Earth sensing satellites, we must also deal with instances of FSP (see [8] and [9]).

2. New model and formulation

Given $m$ jobs with fixed start and end times, $k$ identical parallel machines and a job-machine mapping, we denote the start and end times of job $i$ by $s_i$ and $e_i$, respectively. $M_i$ is the subset of machines that can complete $i$. A pair (job $i$, machine $j$) such that $j$ belongs to $M_i$ is called an execution mode of the job $i$.

To modelize FSP, we define the set of all execution modes, denoted by $X$ (with $|X| = n$),
and two binary relations, the conflict relation and the exclusion relation, defined on \( X \times X \).

**Definition 2.1.** Given the two execution modes \( x_{il} \) and \( x_{jp} \) (\( x_{il} \) representing the processing of job \( i \) on machine \( l \); \( x_{jp} \) representing the processing of job \( j \) on machine \( p \)), \((x_{il}, x_{jp})\) belongs to the conflict relation, denoted by \( x_{il} \text{C} x_{jp} \), if and only if \( x_{il} \) and \( x_{jp} \) are carried out on the same machine at a same moment, or in other words:

\[
x_{il} \text{C} x_{jp} \quad \text{with} \quad i \neq j \quad \text{and} \quad s_i < s_j < e_i \text{ or } s_i < s_j < e_i.
\]

This binary relation is non-reflexive, symmetric and non-transitive.

**Definition 2.2.** \((x_{il}, x_{jp})\) belongs to the exclusion relation, denoted by \( x_{il} \text{E} x_{jp} \), if and only if they are two execution modes for the same job (i.e. \( i = j, l \neq p \)).

This binary relation is non-reflexive, symmetric and transitive.

From these binary relations, we define two undirected graphs, called the conflict graph (denoted by \( G_c = [X, U_c] \)) and the exclusion graph \( (G_e = [X, U_e]) \) respectively. These two graphs have the same set of vertices representing the set of execution modes defined above. The two sets of edges belonging to the conflict and exclusion graphs respectively represent the conflict and exclusion relations defined on \( X \times X \).

A sub-set of execution modes with, at most, one execution mode per job is represented by an independent set (see, e.g. [2,12] for definition) in \( G_e \). A sub-set of execution modes with at most one execution mode at a time on each machine is represented by an independent set in \( G_e \). Thus a sub-set of execution mode that constitutes a feasible jobs schedule is represented by an independent set in a graph composed of both edges of \( G_e \) and \( G_c \). This graph is denoted by \( G = [X, U] \) (with \( U = U_c \cup U_e \)) and called the incompatibility graph. The incompatibility graph is a 1-graph since the intersection between \( U_c \) and \( U_e \) is empty (if two execution modes are in conflict they are completed on the same machine, thus they cannot be two execution modes for the same job, and vice versa).

Thus a maximum independent set in \( G \) represents a feasible jobs schedule that maximizes the number of jobs performed.

Consequently to solve an instance of FSP, we must determine the stability number (that is to say the maximum independent set cardinality) of the associated incompatibility graph. If the stability number of \( G \) is equal to \( m \) (the number of jobs to be completed), the answer to FSP is ‘yes’; otherwise, the stability number of \( G \) is inferior to \( m \) and implies a ‘no’ answer to FSP.

3. **Approximation algorithms**

Given \( n \) execution modes, each of them representing the decision to perform one of the \( m \) jobs on one of the \( k \) available machines, we must determine the stability number of the associated incompatibility graph \( G \) in order to assess the maximal number of jobs that can be completed. As the problem is NP-complete when the job-machine mapping includes more than two machines per job, algorithms are proposed to find an upper bound and a lower bound for the stability number of \( G \).

3.1. **To determine an upper bound for the stability number of \( G \)**

As an independent set in \( G \) is also an independent set in both \( G_c \) and \( G_e \), there is an obvious upper bound for the stability number of \( G_c \), namely: \( \alpha(G) \leq \min(\alpha(G_c); \alpha(G_e)) \).

This upper bound can be computed in polynomial time. This is clear for \( \alpha(G_c) \) because \( G_c \) is composed of \( m \) connected components, each of them being a clique (the vertices belonging to a maximum clique in \( G_c \) represent all the execution modes for the same job). So, \( \alpha(G_c) \) is equal to \( m \). For \( G_e \), on the other hand, we observe that it is composed of \( k \) disjoint sub-graphs. Each of them is induced by the set of vertices representing all the jobs that can be carried out by the same machine. Each of these sub-graphs is an interval graph (see, e.g. [2] for definition) since a vertex
represents a time interval to complete a job on a machine, and two vertices are joined in $G_e$ if and only if they represent two overlapping time intervals on the same machine. Since the stability number of an interval graph can be computed in polynomial time (in determining the maximal size of a clique in its complementary graph which is a comparability graph – for definition see [2]), $\alpha(G_c)$ can be computed in polynomial time.

The computational results presented in Section 4 indicate that this upper bound for the stability number of $G$ is not precise enough in particular cases. We would therefore propose an algorithm to determine a better upper bound. First, let us observe that if $\theta(G)$ denotes the number of cliques which partition $X$ in $G$, $\alpha(G)$ is inferior or equal to $\theta(G)$ (see, e.g. [2,12]). Thus, the cardinality of a partition into cliques of $X$ in $G$ constitutes another upper bound for the stability number of $G$.

So an algorithm is proposed to determine in polynomial time a partition into cliques of $X$ in $G$ whose cardinality is as near as possible to the minimum. There are several reasons for this approach: first, the clique concept appears quite naturally in our modelling, $G_e$ being a collection of disjoint cliques; moreover, we exhibit some properties of $G$ that allow us to simplify the determination of a partition into cliques in $G$.

Before presenting these properties, we should observe that a minimum partition into cliques of $X$ in $G_e$ or in $G_c$ can be computed in a polynomial time. It is obvious for $G_e$. And, since $G_c$ is composed of $k$ disjoint interval graphs for which there is a polynomial algorithm to determine a minimum partition into cliques (see [6]), a minimum partition into cliques of $X$ in $G_c$ can be computed in polynomial time.

Furthermore, we exhibit two particular properties of $G$:

**Property 3.1.** A clique in $G$ is either a clique in $G_e$ or a clique in $G_c$.

**Property 3.2.** There is a minimum partition into cliques of $X$ in $G$ such that each clique in $G_e$ that belongs to $L_i$ is maximum in $G_e$.

For the proofs of these properties, let us first recall that an $i$-clique is a clique of cardinality equal to $i$.

**Proof of Property 3.1.** Three cases must be considered.

- Concerning the 1-cliques in $G$. As a vertex of $G$ is a vertex of both $G_e$ and $G_c$, every 1-clique in $G$ is a 1-clique in $G_e$ and $G_c$.
- Concerning the 2-cliques in $G$. As an edge of $G$ is an edge either of $G_e$ or $G_c$, every 2-clique in $G$ is a 2-clique either in $G_e$ or in $G_c$.
- Concerning the 3-cliques in $G$. Given a triangle $\{i, j, k\}$ in $G$, two sub-cases have to be distinguished:
  (i) $(ij) \in U_c$ and $(jk) \in U_e$. As the exclusion relation is symmetric and transitive, if $(ik)$ and $(jk)$ belong to $U_c$ then $(ij)$ belongs to $U_e$. In this case, $(ij)$ cannot be an edge of $G_e$ since two execution modes for the same job completed on two different machines cannot be in conflict.
  (ii) $(ij), (ik) \in U_c$ and $(jk) \in U_e$. As $(ij)$ belongs to $U_e$, $i$ and $j$ represent two execution modes performed on the same machine. The same is true for $i$ and $k$. Consequently, $i, j$ and $k$ represent three execution modes completed on the same machine. This implies that $(jk)$ cannot be an edge of $G_e$.

The cases considered above are sufficient for the proof. Indeed, given an $i$-clique in $G$ (with $i$ superior to 3), if it contains edges of $G_e$ and $G_c$, there will of necessity be at least one triangle containing both edges of $G_e$ and $G_c$. This is not possible, as proved above. $\Box$

**Proof of Property 3.2.** Let us consider a minimum partition into cliques of $X$ in $G$ denoted by $L_i$ and a clique $L_i$ in $G_e$ belonging to $L$. If $L_i$ is not maximum in $G_c$, there is necessarily a maximum clique in $G_e$ denoted by $L'_i$ such as $L_i$ is included in $L'_i$ since $G_e$ consists of a collection of disjoint cliques. Let us suppose without loss of generality that: $L_i = L'_i \setminus \{j, k\}$. As $L$ is minimum, $j$ and $k$ cannot make up a proper clique of $L_i$; $j$ and $k$ necessarily belong to two distinct cliques in $G_c$. Let us denote these cliques by $L'_j$ and $L'_k$. Conse-
Algorithm A1. Determination of a partition into cliques of vertices in the incompatibility graph.

\[ V \leftarrow X \]
\[ V' \leftarrow \emptyset \]
\[ L' = \{L'_1, \ldots, L'_m\} \]

While \( V \neq \emptyset \), do.

\[ L^e(V) \leftarrow \text{CLIQUES}(G^e(V)) \]

For all \( L'_i \in L^e(V) \), do

\( \text{If } |L'_i| = 1 \)
\( \text{If } L'_i \subseteq L'_j \text{ such as } |L'_j| > 1 \)
\( L' \leftarrow L' \cup L'_i \)

End if

End for

If \( L' = \emptyset \)
\( L \leftarrow L^e(V) \cup L^e(V') \)
\( V \leftarrow \emptyset \)

Else

\( V \leftarrow V \setminus V(L') \)
\( V' \leftarrow V' \cup V(L') \)

\( V(L') \) contains vertices belonging to the cliques in \( L' \)

End if

End while

If \( |L| \geq |L'| \)
\( P \leftarrow L' \)

Else
\( P \leftarrow L \)

End if

Consequently, it is possible to rule out \( j \) and \( k \) from \( L'_j \) and \( L'_k \) respectively and to add them in \( L_i \) without changing the cardinality of \( L \).

These properties imply the existence of a minimum partition into cliques of \( X \) in \( G \) composed of some cliques in \( G^e \) or/and some maximum cliques in \( G^e \).

For determining a partition into cliques of \( X \) in \( G \), Algorithm A1 is proposed. At the beginning, a minimum partition into cliques of \( X \) in \( G^e \), denoted by \( L' = \{L'_1, \ldots, L'_m\} \), is determined (for all \( i \), the vertices of \( L'_i \) represent all the execution modes for the job \( i \)). At iteration \( i \), \( V \) denotes the set of vertices considered, \( G^e(V) \) and \( G^e(V') \) the conflict and exclusion sub-graphs induced by \( V \). \( L^e(V) \) and \( L^e(V') \) denote the minimum partition into cliques of \( V \) in \( G^e(V) \) and \( G^e(V') \) respectively. At the first step of iteration \( i \), \( L^e(V) \) is computed with the procedure called CLIQUES applied to \( G^e(V) \). CLIQUES applied to a graph \( H = [S, A] \) allows us to determine a minimal partition into cliques of \( S \) in \( H \) if and only if \( H \) is the complementary graph of a comparability graph (this procedure is presented in [6]). If \( L^e(V) \) does not contain any 1-clique, this iteration is the last one and the retained partition, denoted by \( L \), is the union of \( L^e(V) \) and \( L^e(X \setminus V) \). Otherwise, we consider each 1-clique belonging to \( L^e(V) \). If the vertex that belongs to a 1-clique is included in a clique of \( L^e \) whose cardinality is superior to 1, let \( L'_i \) be this clique, the vertices belonging to \( L'_i \) are deleted from \( V \). In fact, this means accepting \( L'_i \) into the final partition. When all 1-cliques have been treated, if \( V \) has not been modified, this iteration is the last one and \( L \) is defined as previously explained, otherwise a new iteration \( (i + 1) \) begins.

At the last iteration, the partition chosen denoted by \( P \) is the one with the smallest cardinality, either \( L \) or \( L' \).

The complexity of Algorithm A1, is of \( O(mn^2) \), \( m \) denotes the number of jobs and \( n \) the number of execution modes. At each iteration, the step which requires the highest number of operations is the determination of a minimum partition into cliques of \( V \) in \( G^e(V) \) with the procedure called CLIQUES. The complexity of CLIQUES is of \( O(n^2) \) on graphs with \( n \) vertices (see [6]). In the worst case, the algorithm requires \( m \) iterations to determine a partition into cliques of \( X \) in \( G \). So the complexity of algorithm 1 is of \( O(mn^2) \).

3.2. To determine a lower bound of the stability number of \( G \)

A lower bound for the stability number of \( G \) is simply the cardinal of an independent set in \( G \). Thus, we propose algorithms, more precisely three greedy heuristics, to determine independent sets whose cardinality is as near as possible to the maximum.

The first algorithm, \( A_1 \), is a classical one. Let us denote by \( V \) the set of vertices that can belong to an independent set. At the first iteration, \( V \) contains all the vertices of \( X \). Then at each iteration, the vertex of smallest degree in \( G(V) \) (the incompatibility sub-graph induced by \( V \) is
Algorithm $A_2$. Second greedy heuristic to determine an independent set in $G$.

$V \leftarrow X$
$S_2 \leftarrow \emptyset$

While $V \neq \emptyset$

$V_{\min} = \{i \in V : d_{G(V)}(i) = \min_{j \in V} [d_{G(V)}(j)]\}$
$d_{G(V)}(i) = \min_{j \in V_{\min}} [d_{G(V)}(j)]$
$S_2 \leftarrow S_2 \cup \{i\}$
$V \leftarrow V \setminus \{i \cup I_{G(V)}(i)\}$
End while

Algorithm $A_3$. Third greedy heuristic to determine an independent set in $G$.

$V \leftarrow X$
$L = \{L_1, \ldots, L_q\}$

While $V \neq \emptyset$

$L_{\min} = \{j \in L_q : \forall L_j \in L : |L_j| = \min_{k \in \{1, \ldots, q\}} [|L_k|]\}$
$d_{G(V)}(i) = \min_{j \in L_{\min}} [d_{G(V)}(j)]$
$S_3 = S_3 \cup \{i\}$
$V \leftarrow V \setminus \{i \cup I_{G(V)}(i)\}$
$V_j = L_j \setminus \{i \cup I_{G(V)}(j)\}, \forall j \in \{1, \ldots, q\}$
End While

Then, the stability number of $G$ can be bounded in the following way:

$$\max\{|S_1|, |S_2|, |S_3|\} \leq \alpha(G) \leq |P|.$$ 

4. Computational results

4.1. FSP in the context of production planning for low-orbit Earth observation satellites

We are mainly concerned with problems resulting from production planning for low-orbit Earth observation satellites. In fact, in this production planning problem, we have to solve the following decision problem, called Feasibility Pictures Problem (denoted by FPP): is it possible to satisfy the users' demand for pictures within a given period? To satisfy a demand for a picture means here to take a shot of a particular Earth landbelt.

During a period equal to $T$, the low-orbit Earth observation satellite considered makes a fixed number of revolutions around the Earth (such a revolution is called an orbit). The two intersections of orbit tracks are set approximately at the North Pole and the South Pole. In fact, the camera (with visible optical) on board such a satellite can photograph the Earth only from half-orbits during which the satellite flies from the South Pole to the North Pole. This camera photographs Earth landbelts (at most one at a time).

Thanks to satellite trajectory and camera production capacities, the same landbelt can be photographed from a sub-set of orbits described by the satellite during $T$. The shot of a landbelt from an orbit induces a fixed start and end time. When a shot start time represents the time required by the satellite for flying from the South Pole to the latitude of the first point to be photographed, start and end times of a picture to be performed are equal from whatever orbit the shot is taken.

So the $k$ orbits described by the satellite during $T$ can be thought of as $k$ identical parallel machines, and the $m$ landbelts to be photographed as $m$ non-preemptive jobs with fixed start and end time. Thus an instance of FPP
corresponds exactly to an instance of FSP. Thus, it is possible to follow the approach presented above to solve FPP: an incompatibility graph \( G \) is associated with an instance of FPP, and proposed algorithms are used to bound the stability number of \( G \).

For example, let us consider an instance of FPP in which 4 earth landbelts must be photographed during the 3 days to come by a satellite with one camera. During this period, this satellite describes 3 orbits from which it is possible to photograph one of the considered landbelts. The landbelt \( g_1 \) can be photographed from orbits 1, 2 and 3; the landbelts \( g_2, g_3 \) and \( g_4 \) can be photographed from orbits 2 and 3 (see Fig. 1). An execution mode denoted by \( x_{ij} \) represents the fact to photograph a landbelt \( i \) from an orbit \( j \) (at a given time). So after having defined the set \( X \) of execution modes and the conflict and exclusion relations on \( X \times X \), we can assess the maximum number of landbelts that can be photographed in determining an upper bound and a lower bound for the stability number of the associated incompatibility graph.

For example (see Fig. 1), with the proposed algorithms, we obtain:

\[
P = \{(x_{11}, x_{12}, x_{13}), (x_{22}, x_{32}, x_{42}), (x_{23}, x_{33}, x_{43})\},
\]

\[
S_1 = \{x_{11}, x_{22}, x_{33}\},
\]

\[
S_2 = S_3 = \{x_{32}, x_{43}, x_{11}\}.
\]

Thus, the stability number of \( G \), or equivalently the maximum number of landbelts that can be photographed, is equal to 3.

To assess the efficiency of the proposed algorithms in this context, we have made some computational experiments. Results obtained on instances of FPP randomly generated are explained in detail in [8]. They are very promising since the average value of the ratio \( \max\{|S_1|; |S_2|; |S_3|\}/|P| \) computed for each instance is equal to 0.945.

Obviously, we are aware that incompatibility graphs associated with instances of FPP may present some particular characteristics missing from incompatibility graphs associated with instances of FSP with a random job-machine mapping. This is why we carry out computational experiments in the more general case of FSP.

4.2. The general case

100 instances of FSP have been generated for each of the following contexts characterized by the pair \((k, m)\): \((10, 50)\), \((10, 100)\), \((10, 200)\), \((10, 300)\), \((20, 50)\), \((20, 100)\), \((20, 200)\), \((30, 100)\).

An instance is defined as follows: for each job, the start time and the end time are randomly chosen between 0 and 7 hours, each job's completion time is also randomly chosen between 10 s and 100 s; each job can be completed on at most 50% of the available machines and machines on which each job can be carried out are randomly selected. An incompatibility graph \( G = [X, U] \) is associated with each instance.

![Fig. 1.](image-url)
For each instance we determine a lower bound and an upper bound for the stability number of G and we compute the following ratios:

\[ \begin{align*}
\forall i & \in \{1, \ldots, 4\}:
\beta_i &= \frac{B_i}{\min\{\alpha(G_e), \alpha(G_c)\}} \\
\text{with } \forall i & \in \{1, 2, 3\}, B_i = |S_i| \text{ and } B_4 = \max_{i \in \{1, 2, 3\}} |S_i| ,
\forall i & \in \{1, \ldots, 4\}:
\rho_i &= B_i / |P| .
\end{align*} \]

\(P\) is the partition into cliques of \(X\) in \(G\) determined with Algorithm \(A_1\).

Tables 1 and 2 present the average, the maximal and the minimal value of ratios \(\beta_i\) in each context characterized by a particular number of jobs to be completed on a particular number of machines.

It is clear from these tables that the three greedy heuristics used to determine independent sets give approximately the same results. Nevertheless, it is useful to define three independent sets in order to keep the one with the greatest cardinality. In some particular contexts, such as (20, 50), (20, 100) and (30, 100), these bounds are very efficient. It is due to the fact that in these contexts all the jobs can easily be completed. Thus, the minimum partition into cliques of \(X\) in \(G\) is exactly the same as the minimum partition into cliques of \(X\) in \(G_e\). Consequently, a maximum independent set can easily be determined with a greedy heuristic. However, in the other contexts, these bounds are not so efficient.

Table 3 presents the same results for ratio \(\rho_4\) in the contexts in which the upper bound used above was not efficient enough. With algorithm \(A_1\), \(P\) is computed in less than 4 minutes in all cases.

In all cases, \(P\) enables us to bound the stability number of \(G\) more efficiently. The worst result appears for the case in which 200 jobs must be performed on 20 machines. In these instances, neither exclusion relations nor conflict relations between execution modes are dominant. So the minimum partition into cliques of \(X\) in \(G\) is really a combination of cliques in \(G_e\) and \(G_c\). The heuristic used in Algorithm \(A_1\) does not allow us to determine such a partition. Thus in this case, it will be useful to define more suitable heuristics.

With the exception of this isolated case, the results are satisfactory. They argue in favour of this new approach which, moreover, can be applied to solve VSP in an approximate way.
5. Scheduling jobs within time windows on identical parallel machines

Our purpose now is to apply the model and algorithms previously presented to solve VSP: the problem of scheduling jobs within time windows on \( k \) identical parallel machines. This problem is NP-complete whatever the number of available machines (see [10]). Each job \( i \) is now associated with the earliest possible start time \( s_i^- \), the latest possible start time \( s_i^+ \) and a completion time \( d_i \).

When each job to be completed can be processed during a time window, the set of all jobs' execution modes is infinite. Thus, we propose to discretise this set in choosing for each job a discrete subset of execution modes, or equivalently a discrete subset of start times. In this framework an execution mode represents the completion of a job on a particular machine at a given start time. This discretisation allows us to retain a finite set \( T \) of execution modes to complete \( m \) non-preemptive jobs on \( k \) parallel machines.

On \( T \times T \), we define the exclusion and conflict relations. These relations have the same properties than those mentioned in Section 2. However, when two execution modes for the same job are performed on the same machine at different start times, this pair of execution modes can belong to both exclusion and conflict relations. Thus, incompatibility graph that represents these binary relations on \( T \times T \) is a 2-graph: two vertices can be joined in \( G \) by two edges, one belonging to the exclusion graph, the other belonging to the conflict graph.

Two cases can be distinguished: case a) in which each job cannot be carried out twice on the same machine (for all job \( i \), \( t_i^+ - t_i^- < 2d_i \)), and case b) in which it is possible to complete the same job several times consecutively on the same machine.

In case a), the incompatibility graph \( G = (X, U) \) representing the exclusion relation and the conflict relation defined on \( X \times X \) respects Properties 3.1 and 3.2 mentioned in Section 3.1.

Proof. To prove that \( G \) respects Property 3.1, it is enough to consider the 1-cliques, the 2-cliques and the 3-cliques in \( G \). Concerning the 1-cliques and the 2-cliques, the proof is obvious. But, concerning the 3-cliques, two cases must be distinguished:

(i) Let \( i, j \) and \( k \) be three vertices forming a clique in \( G \) such that \((ij) \) and \((jk) \) belong to \( \cup_e \) and \((ik) \) belongs to \( \cup_c \); as the exclusion relation is transitive, \((ik) \) belongs to \( \cup_e \). This means that \( i, j \) and \( k \) represent three execution modes for the same job on the same machine at different start times, and consequently \( \{i, j, k\} \) is a clique in \( G_e \).

(ii) Let \( i, j \) and \( k \) be three vertices forming a clique in \( G \) such that \((ij) \) and \((jk) \) belong to \( \cup_e \) and \((ik) \) belongs to \( \cup_c \); \( i, j \) and \( k \) represent three execution modes completed on the same machine, and as each job cannot be carried out twice on the same machine, \((ik) \) necessarily belongs to \( \cup_c \). So \( \{i, j, k\} \) is a clique in \( G_c \).

Since \( G_e \) is a collection of disjoint cliques, the proof for Property 3.2 is almost the same as the one previously presented.

Thus, the modelization and algorithms defined for FSP to determine a lower bound and an upper bound for the stability number of \( G \) are suitable to solve VSP in case a).

In case b), the incompatibility graph does not respect Property 3.1. It is quite simple to see from the previous proof. Indeed, if it is possible to complete the same job twice on the same machine, case (ii) of this proof is not always verified: a 3-clique \( \{i, j, k\} \) such as \((ij) \) and \((jk) \) belong to \( \cup_e \) and \((ik) \) belongs to \( \cup_c \) can exist in \( G \) without necessarily causing \((ik) \) to belong to \( \cup_c \). So, in case b), a clique in \( G \) is not only either a clique in \( G_e \) or in \( G_c \), but can be made up of both edges of \( G_e \) and \( G_c \).

In case b), algorithm \( A_1 \) is less suited because it does not consider the existence of mixed cliques (with both edges of \( G_e \) and \( G_c \)). A new heuristic must be defined to determine a partition into cliques of \( X \) in \( G \) when the number of mixed cliques is important.

For some types of low-orbit Earth observation satellites, the Feasibility Pictures Problem mentioned in Section 4.1 can be formulated as an
instance of VSP in case a) (see [9]). Moreover, some other applications in other fields can imply the resolution of instances of VSP. For example in the air traffic control field (see [4]), we have to assess the number of tasks that can be completed by a controller and an automatic system. Such a problem engenders the resolution of VSP’s instances in cases a) or b). The used of proposed algorithms in this context is discussed in [9].

6. Conclusions

The proposed model provides a new way to solve FSP and VSP. Heuristics defined in this paper are efficient in our computational experiments. But in some cases, and in particular to solve VSP when time windows in which jobs can be completed allow to perform them several times consecutively, new heuristics must be defined. It will be useful to test some known (meta)heuristics to determine a partition into cliques in incompatibility graphs.

Moreover, we would emphasize that the partition into cliques may provide additional information about each job’s feasibility (see [8]). For example, if a clique in the obtained partition P is a clique in $G_c$ whose vertices represent all the execution modes for a given job, this job can certainly be completed without preventing the completion of any of the other jobs. This type of information in certain circumstances may be very important.

References


