UNIVERSITE DES SCIENCES ET DE LA TECHNOLOGIE
HOUARI BOUMEDIENE,
USTHB - ALGER

LABORATOIRE DE RECHERCHE OPERATIONNELLE,
COMBINATOIRE, INFORMATIQUE THEORIQUE
ET METHODES STOCHASTIQUES, RECITS

Les annales RECITS

http://www.lrecits.usthb.dz/annales.htm

Vol. 2, 2015

ISSN : 2392-5426

Objectifs :

Les annales RECITS édités par le Laboratoire sont des publications scientifiques de qualité qui ont un rôle important à jouer pour la diffusion des travaux de recherche du laboratoire. Ils permettent, aux membres du laboratoire, de disposer d'un moyen de faire connaître rapidement un résultat de recherche avant sa parution dans une conférence ou une revue. Typiquement, il s'agit de la première version d'un article soumis à une revue ou à une conférence internationale et qui ne paraîtra éventuellement que beaucoup plus tard.

Comité éditorial :

Procédure de soumission :

Il n'existe pas de procédure de revue pour ces rapports de recherche. Pour publier un rapport de recherche, il faut envoyer un courrier au responsable du comité éditorial contenant le texte (fichiers latex et pdf) accompagné du formulaire de publication signé par un des membres du comité éditorial. La première page (style à télécharger) doit contenir uniquement et obligatoirement le titre, les noms, prénoms, adresses et e-mails des auteurs, les résumés et les mots clés en français et en anglais et éventuellement le financement (sponsoring, bourse de recherche, aide financière, etc.). Le tirage papier est fait en deux exemplaires dont un est remis aux auteurs. Le rapport de recherche est publié sur la page web du laboratoire et répertorié au Centre de Recherche sur l'Information Scientifique et Technique (CERIST).

Les annales RECITS du laboratoire peuvent accueillir aussi des textes de chercheurs n'appartenant pas au laboratoire et qui travaillent sur des thèmes proches que ceux développés par le laboratoire. Ces chercheurs sont priés de contacter le responsable du comité éditorial.

Le contenu des textes publiés relève de la seule responsabilité de leurs auteurs.

Contacts :

Email : lrecits@usthb.dz
Tél :
Fax :

CONTENU

Les Annales RECITS (http://www.lrecits.usthb.dz)
Vol. 2, 2015, pages 1-6

An exact method to solve the multi-objective minimum spanning tree problem

Asma BOUMESBAH * **and Mohamed El-Amine CHERGUI** †

RECITS Laboratory, Faculty of Mathematics, USTHB,
PB 32 El Alia 16111 Algiers, Algeria.

* aboumesbah@usthb.dz † mchergui@usthb.dz

Abstract: An exact method is described to generate the efficient set of the multi-objective minimum spanning tree problem. It is based on a branch and bound method and uses a branching process with respect to particular edges of the given graph, inducing a step of constructing constraints of spanning tree problem at each node of a search tree. This has the effect of partitioning the initial graph into sub-graphs, each of which corresponds to a discrete multi-objective linear program allowing to find the efficient set of spanning trees.

Keywords: Minimum spanning tree, integer linear programming, multiple objective linear optimization, combinatorial optimization.

Résumé : Dans cette étude, une méthode exacte est présentée pour le problème de l'arbre multi-objectif. Elle est basée sur un principe de séparation par rapport à des arêtes particulières du graphe, induisant une étape de construction des contraintes du problème, de proche en proche, dans une structure arborescente. Ceci a pour effet de partitionner le graphe initial en sous graphes, chacun correspondant à un programme multi-objectif linéaire discret permettant de trouver des arbres efficaces du problème.

Mots clés : Arbre, programme linéaire en variables entières, optimisation multi-objectif linéaire.

1 Introduction

The problem of multi-objective minimum spanning tree (MOST) is encountered in many practical applications such as optimization of the service's quality in a telecommunication network whose evaluation is represented by a set of criteria: minimum delay, maximum flow, minimum error rate, minimum operating cost ... etc [4]. In this study, we describe an exact method to find the efficient set of MOST in an undirected connected graph in which a cost vector of dimension p≥2 is associated with each edge. This problem is known to be NP-hard even for $p = 2$ [1, 7] and although some methods developed for $p = 2$ were theoretically generalized for $p \geq 3$ [7, 11], no implementation was made. In [12], the authors proposed an algorithm to solve the bi-objective case based on two phases. The first phase calculates two supported efficient solutions obtained by solving the single objective spanning tree problem for each criterion. To generate the non-supported solutions, a â euro œk-bestâ euro ? algorithm [5] or the branch-and-bound method are used to solve a weighted sum program. The authors of reference [2] aim to find all the non-dominated solutions of the bi-objective minimum spanning tree problem by reducing it into a single-objective problem. The method is based on the weighted sum function, which uses Kruskal's algorithm [8] to obtain the first supported spanning tree T1. To build a new spanning tree T2, the edges that are not in T1 are replaced by those which must leave T1, while preserving the non-dominance property. Still in the bi-objective case, a branch and bound approach is also given in reference [11]. This is mainly based on an estimation of all points not dominated by a given sub-problem that takes advantage of the two-dimensional nature of the objective space. In fact, in this case it is easy to list the entire extreme supported points of a sub-problem. Corley's algorithm [6], is described to generate the efficient set of MOST. However, a counter example which shows that the algorithm is able to find spanning trees that are not efficient has been given by Hamacher and Ruhe [7]. In fact, Corley suggests to construct iteratively spanning trees of the graph, using Prim's algorithm [10]. At each step of the algorithm, the set of vertices contained in the already sub-tree defines a cut in the original graph. The current sub-tree is increased by sub-trees as many as non-dominated edges existing in this cut. This paper describes an exact method for finding the complete efficient set of the MOST problem with no restriction on the criteria number p. Its principle search tree is based on a two-step procedure performed alternately: we start by a separation step with respect to edges in common with at least two cycles of the given graph, yielding the step of constructing the problem constraints in the form of linear equalities and inequalities. This ensures the non-existence of cycles. Each branch of the search tree induces a multi-objective discrete linear program modeling the problem of the corresponding MOST throughout this branch.

2 Definitions and notations

Given a connected and undirected simple graph $G = (V, E)$ of order n, where each edge $e_i \in E, i = \overline{1, m}$, is valued by a cost vector $c_{ki}, k = 1..r, r \geq 2$, and $t \subset E$ is a spanning tree of G. The cost vector of the spanning tree T is given by $C_k(T) = \sum_{e_i \in T} c_{ik}; k = 1..r$. We note $C(T) = (C_k)_{k=1..r}$. We say that the vector $C(T)$ dominates another vector $C(T')$ if $C_k(T) \leq C_k(T') \forall k = 1..r$ and $C_k(T) < C_k(T')$ for at least one index k. A spanning tree

T of G is efficient if there is no spanning tree T' of G such that $C(T)$ dominates $C(T')$. The ideal point I has coordinates I_k such that:

$$I_k = C_k(T) = \min\left\{C_k(T) \setminus T\ spanning\ tree\ of G\right\}, k = 1..r$$

The mathematical model associated with MOST is written as follows:

$$x_i = \begin{cases} 1 & if\ \ the\ \ edge\ \ e_i \in T \\ 0 & otherwise\ \ i = \overline{1,m} \end{cases}$$

$$(P)\begin{cases} MinZ_1 = \sum_{i=1}^{m} c_{i1}x_i \\ MinZ_2 = \sum_{i=2}^{m} c_{i2}x_i \\ \quad\vdots \\ MinZ_r = \sum_{i=r}^{m} c_{ir}x_i \\ \sum_{j=1}^{m} = n-1 & (1) \\ \sum_{j\in E(S)}^{n} \leq |S|-1, \forall S \subset V, S \neq \varnothing & (2) \\ x \in \{0,1\}^n \end{cases}$$

where $|E_S|$ is the number of edges of the sub-graph induced by the set $S, S \subset V$.

In the model (P), the number of constraints of type (2) is exponential and
any attempt to solve it with an exact method is already useless in the single objective case.

3 Principle of the method

We first start by recalling an important property in the graphs, namely an edge of a graph $G = (V, E)$ is either an isthmus, or it belongs to a cycle of G. Our method is based on this property in the sense that obtaining all the spanning trees of G is done by dissecting the edges of cycles of G, particularly those common to two or more cycles. We denote $e2c$ such a type of edges. Indeed, the belonging or not of such edges to a spanning tree T prevents the creation of cycles in T. The proposed approach is a separation-construction type in a structured search tree form. The separation is done on the $e2c$-edges of G and induces the construction step to describe constraints (2) of the program (P) which ensures the elimination of cycles relatively to $e2c$-edges. Each branch k of the search tree is constructed by acting alternately on both separation and construction steps, which terminates at a leaf f when there are no $e2c$-edges. Hence, at each of such leaves f, we obtain a subgraph H with two possibilities. In the first, the edges of H constitute a spanning tree reduced to the unique solution of the corresponding branch. The second possibility is that H contains only edge-disjointed cycles in which case the constraints eliminating all these cycles are added to the system of constraints previously established. The obtained set of constraints constitutes a multi-objective linear program (P_f) in binary variables with respect to the objective functions of program (P). The program (P_f) can be solved using one of the exact methods proposed in references [3, 9], generating the set MSTf of the efficient spanning trees and the set corresponding $SNDf$ of non-dominated cost-vectors, associated with the leaf f.

4 Preliminary Results

Let $G = (V, E)$ be a connected and undirected graph of order n and size m. The following propositions allow highlight particular edges of the graph G whose the belonging or not to efficient spanning trees is proved.

Proposition 1 *: Let μ a cycle of G and $e \in \mu$ an edge of G. If all costs vectors of other edges of the cycle μ dominate the one of the edge e, then this edge can not belong to any efficient spanning tree.*

Proof. Let T be a spanning tree and $e \in \mu$, μ a cycle of G such that all costs vectors of other edges of the cycle μ dominate the one of the edge e and assume that $e \in T$ and let $f \in \mu$ and $f \notin T$. Then $T' = T \cup f \setminus e$ is a spanning tree that dominates T because the cost vector of f dominates that of $e \in \mu$. ■

Proposition 2 *: Let e be an edge of G common to a set Δ of cycles of G, and assume that the cost vector of e dominates all other costs vectors of the edges of cycle Δ, then the edge e belongs to all efficient spanning trees of G.*

Proof. We note $E(\Delta)$ the set of all edges in Δ. If there exists an efficient spanning tree T which does not contain the edge e, then $T' = T \cup e \setminus f, \forall f \in E(\Delta)$, is a spanning tree whose cost vector that dominates the one of the spanning tree T because the edge e dominates the edge $f, \forall f \in E(\Delta)$. ■

Example 1 *Consider the graph $G = (V, E)$ where $V = \{1, 2, 3, 4, 5, 6\}$ and $E = \{12, 13, 15, 23, 34, 36, 45, 56\}$. Each edge is provided with three costs as shown in the following matrix of costs:*

$$C = \begin{pmatrix} 0 & 2 & 1 & 2 & 4 & 1 & 3 & 0 \\ 3 & 3 & 2 & 2 & 3 & 1 & 1 & 2 \\ 1 & 0 & 3 & 2 & 4 & 1 & 2 & 0 \end{pmatrix}$$

Reduction
Let the cycle-basis $B = \{\mu_1, \mu_2, \mu_3\}$ with $\mu_1 = \{13, 36, 65, 51\}$, $\mu_2 = \{12, 23, 31\}$ and $\mu_3 = \{43, 36, 65, 54\}$.

- *In the cycle μ_3 the cost vector of the edge (34) is dominated by the other costs vectors of all edges in μ_3. According to **Proposition 1**, this edge is removed from the graph $G = G \setminus (34)$.*
- *Determine then a new cycle-basis $B = B \setminus \mu_3$. and find the set $F = \{e_0 \in E \setminus \exists \mu_l$ and $\mu_{l'} \in B; e_0 \in \mu_l \cap \mu_{l'}\}$.*
  Let the new cycle-basis $B = \{\mu_1, \mu_2\}$ with $F = \{13\}$.

Separation and construction

The separation is done with respect to the edge (13) *creating two nodes. At the NT1-type node of the search tree, we deduce the following system:*

$$(S_1)\begin{cases} x_{13} = 1 \\ x_{13} + x_{36} + x_{65} + x_{15} \leq 3 \\ x_{13} + x_{32} + x_{21} \leq 2 \\ x_{12} + x_{13} + x_{15} + x_{23} + x_{36} + x_{45} + x_{65} = 5 \\ x_{12}, x_{13}, x_{15}, x_{23}, x_{36}, x_{45}, x_{65} \in \{0, 1\} \end{cases}$$

Evaluation

$F = \emptyset$, then the node 1 is a leaf. By using the method described in [3] for the resolution of multi-objective linear program with binary variables (P_1) whose systems of constraints is (S_1), it returns all efficient spanning trees associated with this branch of the search tree:

$T_1 = [12, 13, 45, 65, 36], Z(T_1) = (6, 10, 4)$,
$T_2 = [13, 23, 45, 65, 36], Z(T_2) = (8, 9, 5)$,
$T_3 = [12, 13, 15, 45, 36], Z(T_3) = (4, 11, 5)$.

Reduction

At the NT2-type node of the search tree, the edge (13) is deleted and the graph $G = G\backslash(13)$. The coordinates of the ideal point I corresponding to this node: $Z(I) = (4, 8, 6)$ which is not dominated by any solutions found previously in the first node, therefore the second node is not fathomed.

The new base $B = \mu_2$ was obtained with $\mu_4 = \{12, 23, 36, 65, 51\}$.

Separation and construction

The corresponding system of linear constraints is:

$$(S_2)\begin{cases} x_{13} = 0 \\ x_{12} + x_{23} + x_{36} + x_{65} + x_{51} \leq 4 \\ x_{12} + x_{13} + x_{15} + x_{23} + x_{36} + x_{45} + x_{65} = 5 \\ x_{12}, x_{13}, x_{15}, x_{23}, x_{36}, x_{45}, x_{65} \in \{0, 1\} \end{cases}$$

Evaluation

In this leaf too, it uses the method described in [3] for the resolution of multi-objective linear program with binary variables (P_2) associated with (S_2) all efficient spanning trees associated with this branch are:

$T_4 = [12, 13, 45, 65, 36], Z(T_4) = (6, 9, 6)$,
$T_5 = [12, 15, 45, 65, 36], Z(T_5) = (5, 9, 7)$,
$T_6 = [12, 23, 15, 54, 36], Z(T_6) = (4, 10, 7)$,
$T_7 = [15, 23, 45, 56, 36], Z(T_7) = (7, 8, 8)$.

Thus, the final efficient set of spanning trees of G is: $\{T_1, T_2, \cdots, T_7\}$.

5 Conclusion

The main idea of our method is based on a branching process on common edges with at least two cycles of a graph G. This generates a procedure of constructing constraints of non-existence of cycles, which become easy to enumerate after the branching process. At each leaf f of an arbitrary branch k of the search tree, a bivalent linear multi-objective program (P_f) of the minimum spanning tree problem is solved according to the constraints induced along the branch k. The obtained zero-one sparse matrix of constraints is much smaller than the initial set with an exponential number of constraints, leading to a faster

resolution. On the other hand, computing the ideal point is easy for our problem and has allowed us not to visit areas that do not contain efficient spanning trees.

References

[1] P.M. Camerini, G. Galbiati and F. Maffioli, "The complexity of weighted multi-constrained spanning tree problems", Colloquium on the Theory of Algorithms, Pecs, 1984.

[2] PC. G. da Silva and J.C. N. Clìmaco, "A note on the computation of ordered supported non-dominated solutions in the bi-criteria minimum spanning tree problems", Journal of telecommunications and information technologie pp.11–15, 2007.

[3] PM.E-A. Chergui, M. Moulaï and F.Z. Ouaïl, "Solving the Multiple Objective Integer Linear Programming Problem", Modelling, Computation and Optimization in Information Systems and Management Sciences Communications in Computer and Information Science, Volume 14, Part 1, 69–76, 2008.

[4] PJ. Clìmaco, J. Craveirinha and M. Pascoal, "Multicriteria routing models in telecommunication networks-overview and a case study", in Advances in Multiple Criteria Decision Making and Human Systems Management: Knowlodge and Wisdom, Y. Shi, D. Olson and A. Stam, Eds. Amsterdam: IOS Press, pp.17–46, 2007.

[5] PJ. Clìmaco, E. Martins, A bicriterion shortest path algorithm. European Journal of Operational Research,11:399–404,1982.

[6] PH.W. Corley, "Efficient spanning trees", J. Optim, Theory Appl. 45: 481–485, 1985.

[7] PH.W. Hamacher and G. Ruhe, "On spanning tree problems with multiple objectives", Anals of Operations Research, vol. 52, pp. 209–230, 1994.

[8] PJ.B. Kruskal, "On the Shortest Spanning Subtree of a Graph and the Travelling Salesman Problem", Pric. AMS 7, 48–50, 1956.

[9] özlen M. and Azizoğlu, M. (2009) Multi-objective integer programming : A ge- neral approach for generating all non-dominated solutions, European Jour- nal of Operational Research, Vol. 199, pp.25–35.

[10] PR. C. Prim: Shortest Connection Networks and Some Generalizations. Bell system Technical Journal 36, 1389–1401, 1957.

[11] PF. Sourd and O. Spanjaard, "A multi-objective branch-and-bound framework. Application to the bi-objective spanning tree problem. INFORMS Journal of Computing, 20(3):472–484, 2008.

[12] PS. Steiner and T. Radzik, "Computing all efficient solutions of the biobjective minimum spanning tree problem", Computers & Operations Research 35, 198–211, 2008.

Les Annales RECITS (http://www.lrecits.usthb.dz)
Vol. 02, 2015, pages 7-13

Les suites quasi Morgan-Voyce comme convolutions itérées de suites de Pell généralisées

Lyes Ait Amrane*, et Hacène Belbachir*****

* USTHB/Faculty of Mathematics, LATN Laboratory, DG-RSDT
BP 32, El Alia, 16111, Bab Ezzouar, Algiers
ALGERIA
** Ecole nationale Supérieure d'Informatique (ESI)
BP 68M Oued Smar, 16270, El Harrach, Algiers
ALGERIA
*** USTHB/Faculty of Mathematics, RECITS Laboratory, DG-RSDT
BP 32, El Alia, 16111, Bab Ezzouar, Algiers
ALGERIA

l_ait_amrane@esi.dz, lyesait@gmail.com
hbelbachir@usthb.dz, hacenebelbachir@gmail.com

Abstract: We show that the coefficients of the general term of the quasi Morgan-Voyce sequence, expressed as a polynomial in t, are a convolution of generalized Pell sequences.

Keywords: Quasi Morgan-Voyce sequences, Pell sequences and convolution.

Résumé : Nous montrons que si l'on exprime le terme général de la suite quasi Morgan-Voyce comme polynôme en t, alors ses coefficients sont un produit de convolutions de suites de Pell généralisées.

Mots clés : Suites quasi Morgan-Voyce, suites de Pell et produit de convolution.

1 Introduction

Horadam [2] a introduit la *suite quasi Morgan-Voyce* définie par

$$\begin{cases} Q_1^{(s)} = 1, \quad Q_2^{(s)} = 1 + t + s, \\ Q_n^{(s)} = (2+t)Q_{n-1}^{(s)} + Q_{n-2}^{(s)}, \quad (n \geq 3). \end{cases} \tag{1}$$

Si nous écrivons $Q_{n+1}^{(s)} = \sum_{k=0}^{n} Q^{(s)}(n,k)t^k$ pour $n \geq 0$, alors, dans un premier temps, nous montrons que les coefficients $Q^{(1)}(n,k)$ sont des produits de convolution de la suite de Pell $(P_n)_n$, définie ci-dessous, avec elle même. Ensuite, nous montrons que les coefficients $Q^{(s)}(n,k)$ sont des produits de convolutions de suites de Pell généralisés.

2 Cas $s = 1$

On commence par le cas $s = 1$. Dans ce cas, notre suite est donnée par

$$\begin{cases} Q_1^{(1)} = 1, \quad Q_2^{(1)} = 2 + t, \\ Q_n^{(1)} = (2+t)Q_{n-1}^{(1)} + Q_{n-2}^{(1)}, \quad (n \geq 3). \end{cases} \tag{2}$$

On va montrer que les coefficients $Q^{(1)}(n,k)$ sont exactement les lignes du triangle de convolution de la suite de Pell donné ci-dessous, ce triangle correspond à A054456 dans [3]. On rappelle que la suite de Pell $(P_n)_n$ est définie par

$$\begin{cases} P_1 = 1, \ P_2 = 2, \\ P_n = 2P_{n-1} + P_{n-2}, \quad (n \geq 3). \end{cases} \tag{3}$$

Le triangle de convolution des suites de Pell est défini comme suit : la première colonne est constituée des nombres de Pell, la deuxième colonne est constituée de la convolution de la colonne des nombres de Pell avec elle-même, la troisième colonne est constituée de la colonne des nombres de Pell convolée deux fois avec elle-même et ainsi de suite. Les 0 dans le Tableau 1 sont représentés par un vide.

Si on note par $D(n,k)$ les éléments de ce triangle, alors on a

$$D(n,k) = 0, \qquad (n < k), \tag{4}$$

$$D(n,n) = 1 \qquad (n \geq 0), \tag{5}$$

$$D(n,0) = P_{n+1}, \qquad (n \geq 0), \tag{6}$$

et pour tout $j \in \{0, 1, \ldots, k-1\}$

$$D(n,k) = \sum_{i=0}^{n-1} D(i,j)D(n-1-i, k-1-j), \qquad (k,n \geq 1). \tag{7}$$

$n\backslash k$	0	1	2	3	4	5	6	7	8	9	...
0	1										
1	2	1									
2	5	4	1								
3	12	14	6	1							
4	29	44	27	8	1						
5	70	131	104	44	10	1					
6	169	376	366	200	65	12	1				
7	408	1052	1212	810	340	90	14	1			
8	985	2888	3842	3032	1555	532	119	16	1		
9	2378	7813	11784	10716	6482	2709	784	152	18	1	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱

TABLE 1 – Triangle de convolution de la suite de Pell.

Lemme 1 *Soient $n \geq 1$ et $k \geq 1$ des entiers, alors on a*

$$D(n+1,k) = D(n,k-1) + 2D(n,k) + D(n-1,k). \tag{8}$$

Preuve. Soient $n \geq 1$ et $k \geq 1$ des entiers, alors

$$\begin{aligned}
D(n+1,k) &= \sum_{i=0}^{n} D(i,k-1)D(n-i,0) \\
&= D(n,k-1)D(0,0) + D(n-1,k-1)D(1,0) + \sum_{i=0}^{n-2} D(i,k-1)D(n-i,0) \\
&= D(n,k-1) + 2D(n-1,k-1) \\
&\quad + \sum_{i=0}^{n-2} D(i,k-1)\left[2D(n-1-i,0) + D(n-2-i,0)\right] \\
&= D(n,k-1) + 2D(n-1,k-1) + 2\sum_{i=0}^{n-2} D(i,k-1)D(n-1-i,0) \\
&\quad + \sum_{i=0}^{n-2} D(i,k-1)D(n-2-i,0) \\
&= D(n,k-1) + 2\sum_{i=0}^{n-1} D(i,k-1)D(n-1-i,0) \\
&\quad + \sum_{i=0}^{n-2} D(i,k-1)D(n-2-i,0) \\
&= D(n,k-1) + 2D(n,k) + D(n-1,k).
\end{aligned}$$

■

Théoreme 2 *Le terme général de la suite $\left(Q_n^{(1)}\right)_n$ est*

$$Q_{n+1}^{(1)} = \sum_{k=0}^{n} D(n,k)t^k, \qquad (n \geq 0). \tag{9}$$

Preuve. Par récurrence sur n. Pour $n = 0$ et 1, l'identité (9) est facile à vérifiée. Supposons que (9) est vraie jusqu'à l'ordre n, alors

$$\begin{aligned}
Q_{n+2}^{(1)} &= (2+t)Q_{n+1}^{(1)} + Q_n^{(1)} \\
&= (2+t)\sum_{k=0}^{n} D(n,k)t^k + \sum_{k=0}^{n-1} D(n-1,k)t^k \\
&= 2\sum_{k=0}^{n} D(n,k)t^k + \sum_{k=0}^{n} D(n,k)t^{k+1} + \sum_{k=0}^{n-1} D(n-1,k)t^k \\
&= 2\sum_{k=0}^{n} D(n,k)t^k + \sum_{k=1}^{n+1} D(n,k-1)t^k + \sum_{k=0}^{n-1} D(n-1,k)t^k \\
&= 2D(n,0) + \sum_{k=1}^{n-1} 2D(n,k)t^k + 2D(n,n)t^n + \sum_{k=1}^{n-1} D(n,k-1)t^k \\
&\quad + D(n,n-1)t^n + D(n,n)t^{n+1} + D(n-1,0) + \sum_{k=1}^{n-1} D(n-1,k)t^k \\
&= [2D(n,0) + D(n-1,0)] + \sum_{k=1}^{n-1} [D(n,k-1) + 2D(n,k) + D(n-1,k)]\, t^k \\
&\quad + [2D(n,n) + D(n,n-1)]\, t^n + D(n,n)t^{n+1},
\end{aligned}$$

d'après le Lemme 1 et les relations (4), (5), (6) et (8) on obtient

$$\begin{aligned}
Q_{n+2}^{(1)} &= D(n+1,0) + \sum_{k=1}^{n-1} D(n+1,k)t^k + D(n+1,n)t^n + D(n+1,n+1)t^{n+1} \\
&= \sum_{k=0}^{n+1} D(n+1,k)t^k.
\end{aligned}$$

■

Proposition 3 *Avec les mêmes notations que précédemment, on a*

$$D(n,k) = \sum_{j=0}^{\lfloor (n-k)/2 \rfloor} \binom{n-j}{j} \binom{n-2j}{k} 2^{n-2j-k}, \qquad 0 \le k \le n. \tag{10}$$

Preuve. D'après [1, Corollaire 1], on a

$$\begin{aligned} Q_{n+1}^{(1)} &= \sum_{j=0}^{\lfloor n/2 \rfloor} \binom{n-j}{j} (2+t)^{n-2j} \\ &= \sum_{j=0}^{\lfloor n/2 \rfloor} \left(\binom{n-j}{j} \sum_{k=0}^{n-2j} \binom{n-2j}{k} 2^{n-2j-k} t^k \right) \\ &= \sum_{\substack{j,k \\ 0 \le k \le n-2j \le n}} \binom{n-j}{j} \binom{n-2j}{k} 2^{n-2j-k} t^k \\ &= \sum_{k=0}^{n} \left(\sum_{j=0}^{\lfloor (n-k)/2 \rfloor} \binom{n-j}{j} \binom{n-2j}{k} 2^{n-2j-k} \right) t^k, \end{aligned}$$

et le résultat découle en identifiant cela avec (9). ■

3 Cas s entier quelconque

Nous traitons maintenant le cas général (s entier quelconque). Supposons donc que notre suite quasi Morgan-Voyce est donnée par (1) et soit $(G_n)_n$ la suite de Pell avec des conditions initiales généralisées, donnée par

$$\begin{cases} G_1 = 1, \quad G_2 = 1+s, \\ G_n = 2G_{n-1} + G_{n-2}, \qquad (n \ge 3). \end{cases} \tag{11}$$

On définit le triangle de convolution des suites de Pell généralisées comme suit : la première colonne est constituée des nombres de Pell avec les conditions initiales 1 et $1+s$, la deuxième colonne est constituée de la convolution de la suite $(G_n)_n$ avec la suite $(P_n)_n$, la troisième colonne est constituée de la suite $(G_n)_n$ convolée deux fois avec la suite $(P_n)_n$ et ainsi de suite. C'est-à-dire, chaque colonne est la convolution de la colonne précédente avec la suite $(P_n)_n$. On va montrer que les coefficients $Q^{(s)}(n,k)$ sont exactement les lignes du triangle de convolution des suites de Pell généralisées donné ci-dessous.

$n\backslash k$	0	1	2	3	4	5	6	...
0	1							
1	1+s	1						
2	3+2s	3+s	1					
3	7+5s	10+4s	5+s	1				
4	17+12s	30+14s	21+6s	7+s	1			
5	41+29s	87+44s	77+27s	36+8s	9+s	1		
6	99+70s	245+131s	262+104s	156+44s	55+10s	11+s	1	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱

TABLE 2 – Triangle de convolution des suites de Pell généralisées.

Si on note par $B(n,k)$ les éléments de ce triangle, alors on a

$$B(n,k) = 0, \qquad (n < k), \tag{12}$$

$$B(n,n) = 1 \qquad (n \geq 0), \tag{13}$$

$$B(n,0) = G_{n+1}, \qquad (n \geq 0), \tag{14}$$

et pour tout $j \in \{0,1,\ldots,k-1\}$

$$B(n,k) = \sum_{i=0}^{n-1} D(i,j)B(n-1-i,k-1-j), \qquad (k,n \geq 1). \tag{15}$$

Lemme 4 *Soient $n \geq 1$ et $k \geq 1$ des entiers, alors on a*

$$B(n+1,k) = B(n,k-1) + 2B(n,k) + B(n-1,k). \tag{16}$$

Preuve. Soient $n \geq 1$ et $k \geq 1$ des entiers, alors

$$\begin{aligned}
D(n+1,k) &= \sum_{i=0}^{n} D(i,0)B(n-i,k-1) \\
&= D(0,0)B(n,k-1) + D(1,0)B(n-1,k-1) + \sum_{i=2}^{n} D(i,0)B(n-i,k-1) \\
&= B(n,k-1) + 2B(n-1,k-1) \\
&\quad + \sum_{i=2}^{n} \left[2D(i-1,0) + D(i-2,0)\right] B(n-i,k-1) \\
&= B(n,k-1) + 2B(n-1,k-1) + 2\sum_{i=2}^{n} D(i-1,0)B(n-i,k-1) \\
&\quad + \sum_{i=2}^{n} D(i-2,0)B(n-i,k-1) \\
&= B(n,k-1) + 2\sum_{i=1}^{n} D(i-1,0)B(n-i,k-1) \\
&\quad + \sum_{i=2}^{n} D(i-2,0)B(n-i,k-1) \\
&= B(n,k-1) + 2\sum_{i=0}^{n-1} D(i,0)B(n-1-i,k-1) \\
&\quad + \sum_{i=0}^{n-2} D(i,0)B(n-2-i,k-1) \\
&= B(n,k-1) + 2B(n,k) + B(n-1,k).
\end{aligned}$$

■

Théoreme 5 *Le terme général de la suite $\left(Q_n^{(s)}\right)_n$ est*

$$Q_{n+1}^{(s)} = \sum_{k=0}^{n} B(n,k)t^k, \qquad (n \geq 0). \tag{17}$$

Preuve. Par récurrence sur n. Pour $n = 0$ et 1, l'identité (17) est facile à vérifiée. Supposons que (17) est vraie jusqu'à l'ordre n, alors

$$\begin{aligned}
Q_{n+2}^{(s)} &= (2+t)Q_{n+1}^{(s)} + Q_n^{(s)} \\
&= (2+t)\sum_{k=0}^{n} B(n,k)t^k + \sum_{k=0}^{n-1} B(n-1,k)t^k \\
&= 2\sum_{k=0}^{n} B(n,k)t^k + \sum_{k=0}^{n} B(n,k)t^{k+1} + \sum_{k=0}^{n-1} B(n-1,k)t^k \\
&= 2\sum_{k=0}^{n} B(n,k)t^k + \sum_{k=1}^{n+1} B(n,k-1)t^k + \sum_{k=0}^{n-1} B(n-1,k)t^k \\
&= 2B(n,0) + \sum_{k=1}^{n-1} 2B(n,k)t^k + 2B(n,n)t^n + \sum_{k=1}^{n-1} B(n,k-1)t^k \\
&\quad + B(n,n-1)t^n + B(n,n)t^{n+1} + B(n-1,0) + \sum_{k=1}^{n-1} B(n-1,k)t^k \\
&= [2B(n,0) + B(n-1,0)] + \sum_{k=1}^{n-1} [B(n,k-1) + 2B(n,k) + B(n-1,k)]\, t^k \\
&\quad + [2B(n,n) + B(n,n-1)]\, t^n + B(n,n)t^{n+1},
\end{aligned}$$

d'après le Lemme 4 et les relations (12), (13), (14) et (15) on obtient

$$\begin{aligned}
Q_{n+2}^{(S)} &= B(n+1,0) + \sum_{k=1}^{n-1} B(n+1,k)t^k + B(n+1,n)t^n + B(n+1,n+1)t^{n+1} \\
&= \sum_{k=0}^{n+1} B(n+1,k)t^k.
\end{aligned}$$

■

Références

[1] H. Belbachir et F. Bencherif. *Linear recurrence sequences and powers of a square matrix*, Integers **6** (2006), A12, 17 pp.

[2] A. F. Horadam. *Quasi Morgan-Voyce polynomials and Pell convolutions*, Applications of Fibonacci numbers, Vol. 8, (Rochester, NY, 1998), 179–193, Kluwer Acad. Publ., Dordrecht, 1999.

[3] N. J. A. Sloane. *The On-Line Encyclopedia of Integer Sequences* Published electronically at http ://www.oeis.org, 2014.

Les Annales RECITS (http://www.lrecits.usthb.dz)
Vol. 02, 2015, pages 15-26

Scheduling with agreements: new results

Mohamed Bendraouche[1,2] and Mourad Boudhar[2]

[1] Faculty of Sciences, Saad Dahleb University, Route de Soumaa, BP 270, Blida, Algeria
[2] RECITS Laboratory, Faculty of Mathematics, USTHB University, BP 32 El-Alia, BEZ, Algiers, Algeria

mbendraouche@yahoo.fr , mboudhar@usthb.dz

Abstract: In this paper, the problem of scheduling with agreements (SWA) is considered. This consists in scheduling a set of jobs non-preemptively on identical machines subject to constraints that only some specific jobs can be scheduled concurrently on different machines. These constraints are given by an agreement graph and the aim is to minimize the makespan. In the case of two machines we extend two NP-hardness results of SWA with processing times at most 3 that hold for bipartite agreement graphs to any class of agreement graphs with a specific structure. Complexity results of SWA are established in the case of split and complement of bipartite graphs. Finally we present some approximation results for SWA.

Keywords: Scheduling, agreement graph, makespan, complexity, approximation.

Résumé : Nous considérons le problème d'ordonnancement avec concordance. Il consiste à ordonnancer un ensemble de tâches sans interruption sur des machines identiques sous la contrainte que quelques tâches spécifiques peuvent être ordonnancées simultanément sur des machines différentes. Ces contraintes sont représentées par un graphe. Dans le cas de deux machines, deux résultats de NP-complexité sont étendus. De nouveaux résultats, dans les cas d'un graphe scindé et du complémentaire d'un graphe biparti, sont établis. Des résultats d'approximation sont aussi présentés.

Mots clés : Ordonnancement, graphe de concordance, makespan, complexité, approximation.

1 Introduction

We recall the problem of scheduling with agreements (SWA in short) [2]: the input consists of a set $V = \{J_1, J_2, .., J_n\}$ of n jobs with processing and release times p_i and r_i respectively $(i = 1, 2, ..., n)$, and a set of m identical machines. Each machine can process at most one job at a time and each job can be processed on only one machine. The preemption of the jobs is not allowed. We assume that there is a graph $G = (V, E)$ over the jobs, called the agreement graph. Each edge in E models a pair of agreeing jobs that can be scheduled concurrently (on different machines). The agreement constraints are such that only the agreeing jobs can be scheduled concurrently. A schedule is an assignment of jobs to the machines which specifies for each job the time interval and the machine on which this job is to be processed. A feasible schedule is a non-preemptive one which respects the agreement constraints. The aim is to find a feasible schedule that minimizes the makespan.

The SWA problem can be found in the resource-constrained scheduling problems when the resources are non-sharable. A motivation of this problem is the school exam planning: at school, there are n exams $E_1, .., E_n$ 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 class-rooms whose number is m. Also we suppose that the capacity of a class-room is big enough so that any exam can be taken in any class-room. We seek a schedule which minimizes the length of the examination period. This problem can be modelised 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 class-rooms as being the machines. The agreement graph $G = (V, E)$ is such that $V = \{J_1, J_2, ..., J_n\}$ 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 $G = (V, E)$ with processing times in {1,2}.

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 [1], by considering the complement of the agreement graph, called the conflict graph. Many results concerning this problem can be found in the literature (see for example [2, 4, 9, 11, 13, 14]).

Scheduling with agreement graph or scheduling with agreements (SWA) is equivalent to scheduling with conflict graph (the complement of the agreement graph). The latter is known as scheduling with conflicts which was first studied by Irani and Leung [12] and then by Chrobak et al. [7]. Even et al. [8] 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. [8] 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 [2] 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 [3], the authors have thoroughly closed the complexity status of SWA on two machines with two fixed processing times.

The remainder of this paper consists of five sections and is organized as follows. Section 2 presents the generalization of two NP-hardness results of SWA in the case of two machines, processing times at most 3 for bipartite agreement graphs to any class of agreement graphs with a specific structure. In Section 3, a polynomial result in the case of split agreement graphs is established. Section 4 is devoted to complexity results related to complement of bipartite agreement graphs. In Section 5, we establish some approximation results. Concluding remarks constitute the last section.

2 NP-hardness generalizations for two machines

Although for a given agreement graph the SWA problem with three values of processing times is a subproblem of the two values-case, it is still interesting to know the complexity of the former for different classes of agreement graphs. For this purpose we establish the following results.

2.1 Case $p_i \in \{1, 2, 3\}$

As it has previously been mentioned, the SWA problem with two machines and $p_i \in \{1, 2, 3\}$ is strongly NP-hard for arbitrary bipartite agreement graphs. We generalize this result to any class of arbitrary agreement graphs with a specific structure, as stated in the following theorem.

Theorem 1 *The SWA problem with two machines and $p_i \in \{1, 2, 3\}$ is strongly NP-hard for any class of agreement graphs $G = (A \cup B, E)$ where A is an independent set with $|A| = 2a$ and B is a set of cardinality $2a - b$ inducing a subgraph with a specific structure ϕ. The parameters a and b are two arbitrary integers satisfying $b \leq a$.*

Proof. Consider a class C_ϕ of agreement graphs of the form cited, associated with a specific structure ϕ. For example ϕ can either be a discrete, complete, bipartite, or split graph-like structure. We use a similar reduction from the 3-Dimensional Matching problem(3-DM) as it has been done in [2] for the case of arbitrary bipartite graphs $G = (S_1, S_2; E)$. The agreement graph of the scheduling instance is similar: $G = (S \cup W, E)$ where $S = V_Y \cup\ V_Z \cup V_D \cup V_R$ and $W = V_M \cup V_Q$. With respect to the bipartite version the sets W and S play the role of S_1 and S_2 respectively. The edge-set E is constructed as follows. The edges of E between S and W are constructed similarly as those between S_1 and S_2 in the bipartite case. S is an independent set in G. The only difference is that we impose the subgraph induced by W to have the structure ϕ. Clearly we have $|S| = 2|M|$ and $|W| = 2|M| - q$. Note that $|S|$ and $|W|$ are of the form $2a$ and $2a - b$ respectively $(a = |M|, b = q)$ such that $b \leq a$, and this makes the agreement graph G belong to the class C_ϕ.

Next we show that 3-DM has a matching M' if and only if there is a feasible schedule σ with makespan $C_{\max}(\sigma) \leq 4|M| - 2q$.

For the necessary part, the construction of a feasible schedule σ satisfying $C_{\max}(\sigma) \leq 4|M| - 2q$, is the same as in the bipartite case [2].

Conversely, suppose there is a feasible schedule σ with $C_{\max}(\sigma) \leq 4|M| - 2q$. Since the total processing times equals $8|M| - 4q$, then all the jobs are scheduled on both machines without idle times and $C_{\max}(\sigma) = 4|M| - 2q$.

We claim that for every set V_i $(i = 1, ..., q)$, there are $(|M_i| - 1)$ quadruplets of the form $\{R_{i,k}, Q_{i,k}, D_{i,k}$, one job of $V_{M_i}\}$ $(k \in \{1, 2, ..., |M_i| - 1\})$ which must have been scheduled either as shown in Figure 1 or in its symmetric configuration.

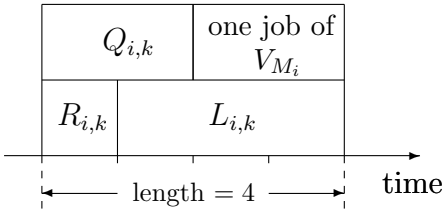

Figure 1: Schedule of the jobs of V_i

To prove this, let $R_{i,k}$ be a job of V_{R_i}, therefore the corresponding job $Q_{i,k}$ of V_{Q_i} must have been scheduled in parallel with $R_{i,k}$ as shown in Figure 2 or in its symmetric configuration. Consider the case of Figure 2, the symmetric case can be done similarly.

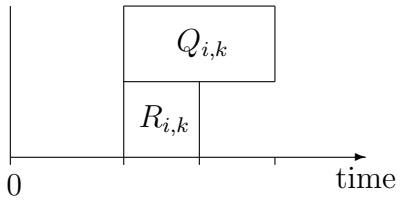

Figure 2: Gantt diagram of case 1

By construction of the agreement graph only four cases may have happened:

case 1: $Q_{i,k}$ has been scheduled in parallel with some job $Q_{j,k'}$ of some set V_{Q_j}. Here two subcases may have happened:
case 1.1: The job $R_{j,k'}$ of V_{R_j} which is in correspondence with the job $Q_{j,k'}$ has been scheduled in parallel with $Q_{j,k'}$, as shown in Figure 3.
The time left is equal to $4|M| - 2q - 3$. On the other hand, the remaining independent jobs not represented in the Gantt diagram of this figure are: $(|M| - q - 2)$ jobs of V_R and all the jobs of V_D, V_Y and V_Z. The minimum time that has been taken by all of these jobs is: $|M| - q - 2 + 3(|M| - q) + 2q = 4|M| - 2q - 2 > 4|M| - 2q - 3$. This implies that the remaining independent jobs just cited could not have been fitted in the time left, therefore this subcase hasn't happened.

case 1.2: The job $D_{j,k'}$ of V_{D_j} which is in correspondence with the job $Q_{j,k'}$ has been scheduled in parallel with $Q_{j,k'}$ and thus the job $D_{j,k'}$ must have been scheduled in parallel

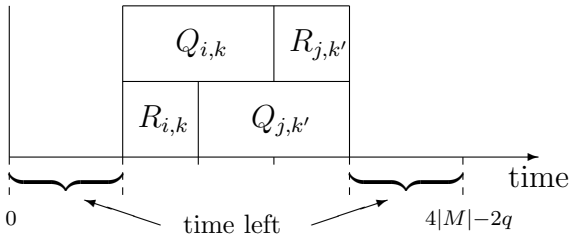

Figure 3: Gantt diagram of case 1.1

with some job of V_{M_j}, see Figure 4.
The remaining independent jobs not represented in the Gantt diagram of this figure are: $(|M| - q - 1)$ jobs of V_R, $(|M| - q - 1)$ jobs of V_D and the $2q$ jobs of V_Y and V_Z. These require a total time of at least $|M| - q - 1 + 3(|M| - q - 1) + 2q = 4|M| - 2q - 4 > 4|M| - 2q - 5$ (that is the time left), contradiction, so this subcase hasn't occurred either.

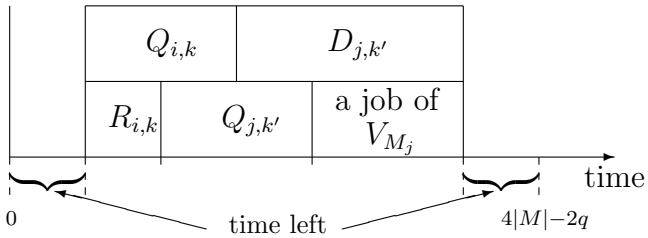

Figure 4: Gantt diagram of case 1.2

case 2: $Q_{i,k}$ has been scheduled in parallel with some job of V_M. The maximum time left is equal to $4|M| - 2q - 2$ (see Figure 5). The remaining independent jobs not represented in this figure are: $(|M| - q - 1)$ jobs of V_R and all the jobs of V_D, V_Y and V_Z. The time that has been taken by all of these jobs is: $|M| - q - 1 + 3(|M| - q) + 2q = 4|M| - 2q - 1 > 4|M| - 2q - 2$. Therefore the independent jobs just cited could not have been scheduled in the maximum time left and thus this case hasn't happened. We deduce that the only possibility that has happened is the following case.

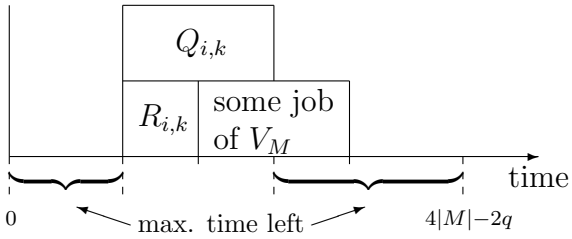

Figure 5: Gantt diagram of case 2

case 3: The job $D_{i,k}$ of V_{D_i} corresponding to the job $Q_{i,k}$ has been scheduled in parallel with $Q_{i,k}$ and some job of V_{M_i} must have been scheduled in parallel with $D_{i,k}$, see Figure 1. The proof of our claim is terminated. With the same argument as in [2], one puts in evidence the required matching. ■

Remark 1 *In the case of two machines, it is worth noting that one can derive many new results by considering different structures ϕ, for instance if ϕ is a discrete graph-like structure, then C_ϕ is a subclass of the class of bipartite agreement graphs and therefore we re-obtain the old NP-hardness result cited in reference [2], for the class of bipartite agreement graphs. If ϕ is a complete graph-like structure, this similarly implies that SWA is strongly NP-hard when $p_i \in \{1, 2, 3\}$ for the class of split graphs. Since the latter is a subclass of chordal graphs, we also have an analogous result for chordal graphs. In case ϕ is a bipartite graph-like structure, we deduce an analogous NP-hardness result for the class of 3-partite agreement graphs.*

2.2 Case $p_i \in \{1, 2\},\ r_i \in \{0, r\}$ (r arbitrary)

We recall that in reference [2], it has been proved that when $m = 2$, $p_i \in \{1, 2\}$ and $r_i \in \{0, r\}$ (r arbitrary), the SWA problem is strongly NP-hard for arbitrary bipartite agreement graphs.
An analogous NP-hardness generalization as in theorem 1, can also been obtained in the case $m = 2$, $p_i \in \{1, 2\}$ and $r_i \in \{0, r\}$ (r arbitrary), bipartite agreement graphs, and this is stated in the following theorem.

Theorem 2 *The SWA problem with two machines, $p_i \in \{1, 2\}$ and $r_i \in \{0, r\}$ (r arbitrary) is strongly NP-hard for any class of agreement graphs $G = (A \cup B, E)$ where A is an independent set with $|A| = 2a$ and B is a set of cardinality $2a - b$ inducing a subgraph with a specific structure. The parameters a and b are two arbitrary integers satisfying $b \leq a$.*

Proof.

Here again we consider the similar reduction from 3-DM, used in Theorem 2 of reference [2], for bipartite agreement graphs. The corresponding scheduling instance is similar, except that the agreement graph is $G = (S \cup W, E)$ where $S = V_Y \cup V_Z \cup V_D$ and $W = V_M$. The edge-set E is constructed in such a way that S is an independent set and that the subgraph induced by W possesses the structure ϕ.

Next we show that 3-DM has a matching M' if and only if there is a feasible schedule σ with makespan $C_{\max}(\sigma) \leq 2|M|$.

The necessary condition is the same as in the bipartite case.

Conversely, in a similar way we show that $|M| - q$ jobs of W must have been scheduled in the time interval $[2q,\ 2|M|]$, opposite to those of V_D, see Figure 5 of reference [2]. As for the $3q$ remaining jobs, the possibility that two jobs of W have been scheduled in parallel cannot happen since the jobs of the set $V_Y \cup V_Z$ could not have been scheduled. Consequently, this gives the required matching. ■

The same previous remark can be drawn with respect to this theorem, in particular we re-obtain the old analogous NP-hardness result of SWA, reference [2], with two machines, $p_i \in \{1, 2\}$, $r_i \in \{0, r\}$ (r arbitrary) and bipartite agreement graphs.

3 Case of Split graphs with unit processing times: a polynomial result

By Remark 1, one can deduce that the SWA problem is strongly NP-hard for arbitrary split agreement. Arbitrary split graphs are denoted $G = (K, S; E)$, where K and S are the clique and the independent set of G respectively. Next we prove that it becomes polynomial when the jobs of the clique K have unit processing times and those of the independent set S are arbitrary.

Proposition 3 *The SWA problem for arbitrary split agreement graphs $G = (K; S, E)$, can polynomially be solved in $O(n^3)$ when the processing times of the jobs of K are equal to one and those of S are arbitrary.*

This problem can be reduced to the maximum flow problem, in the same way as it has been done in [2], in the case of two machines and bipartite graphs $G = (S_1, S_2)$ where the jobs of S_1 have unit processing times. The same Algorithm 1 can be adjusted for the problem at hand by replacing S_1 by the clique K and S_2 by the stable set S, except for some changes in such a way that the main steps of the new Algorithm 1 will take the form:

Algorithm 1
Begin

1: Construct the network R similarly, except that an arc J_i from S to p has capacity $p_i(m-1)$.
2: Determine a maximum feasible flow f^* in R.
3: In Schedule the jobs of S successively in the time interval $[0, \sum_{i=1}^{|S|} p_i]$, on $P1$.
4: In the interval $[0, \sum_{i=1}^{|S|} p_i]$, schedule the entering jobs of K with unit flow value into blocks successively on the $m-1$ machines, such that each block is scheduled with its corresponding job of S.
5: Schedule the remaining jobs of K successively and alternately on the m machines, in the time interval $[\sum_{i=1}^{|S|} p_i, \sum_{i=1}^{|S|} p_i + \lceil \frac{|K| - f^*(u_r)}{m} \rceil]$.

end

Note that step 4 can be implemented in $O(n)$, thus the complexity of this algorithm is $O(n^3)$.

Remark 2 *This theorem generalizes a result of Bodlaender and Jansen (1995).*

4 Case of complement of bipartite agreement graphs, $m = n$, $p_i = 1$

In this section, we consider the SWA problem with unit processing times when the agreement graph is an arbitrary complement of a bipartite graph which is denoted by $G = (K_1, K_2; E)$, where K_1, K_2 are two cliques of the graph G. We also suppose that $m = n$.

4.1 Case $r_i \in \{0, 1, 2\}$

The condition $m = n$ means that there are no processor constraints, then the problem at hand is equivalent to the minimum coloring problem for arbitrary bipartite graphs which is known to be polynomial.

Next we prove that if further we add the release time condition $r_i \in \{0, 1, 2\}$, we get a strongly NP-hard problem.

Theorem 4 *The SWA problem with arbitrary complement of bipartite agreement graphs, unit processing times and $r_i \in \{0, 1, 2\}$ is strongly NP-hard even if $m = n$.*

Proof. Consider the 1-Precoloring Extension problem for bipartite graphs (1-PrExt for short): **Instance:** a bipartite graph $G = (S_1, S_2; E)$ and 3 vertices $v_1, v_2, v_3 \in S_1$. **Question:** is there a 3-coloring of G such that v_i receives color i, for all $i (i = 1, 2, 3)$?
This problem is NP-complete [5]. We prove that the latter can polynomially be reduced to the following problem (D'):
instance: let $m = n$ and let $G' = (K_1; K_2 \cup \{u_1, u_2, u_3\}, E')$ be composed of $\overline{G} = (K_1, K_2; \overline{E})$ so that K_1 and K_2 are two cliques of G' corresponding to S_1 and S_2 respectively, and three dummy jobs u_1, u_2, u_3 such that $K_2 \cup \{u_1, u_2, u_3\}$ is also a clique in G'. Other edges of E' are defined as follows: for each i $(i = 1, 2, 3)$, u_i is adjacent to v_i and to all jobs of $K_1 \backslash \{v_1, v_2, v_3\}$. Thus K_1 and $K_2 \cup \{u_1, u_2, u_3\}$ are two cliques of G' making it the complement of a bipartite graph see Figure 6 (a). All the processing times of the jobs of G' are equal to 1. $r_{v_i} = r_{u_i} = i - 1$ for all i $(i = 1, 2, 3)$, the release times for the rest of the jobs are zero.
Question: does there exist a feasible schedule σ with $C_{\max}(\sigma) \leq 3$?

Suppose there is a 3-coloring (C_1, C_2, C_3) of G such that $v_i \in C_i$ for all i $(i = 1, 2, 3)$. We construct the schedule σ as follows: the jobs of C_i and the job u_i are scheduled at time $t_i = i - 1$ for each i $(i = 1, 2, 3)$. The schedule σ is represented in Figure 6 (b). One can verify that σ is a feasible schedule of the problem D' satisfying $C_{\max}(\sigma) \leq 3$.

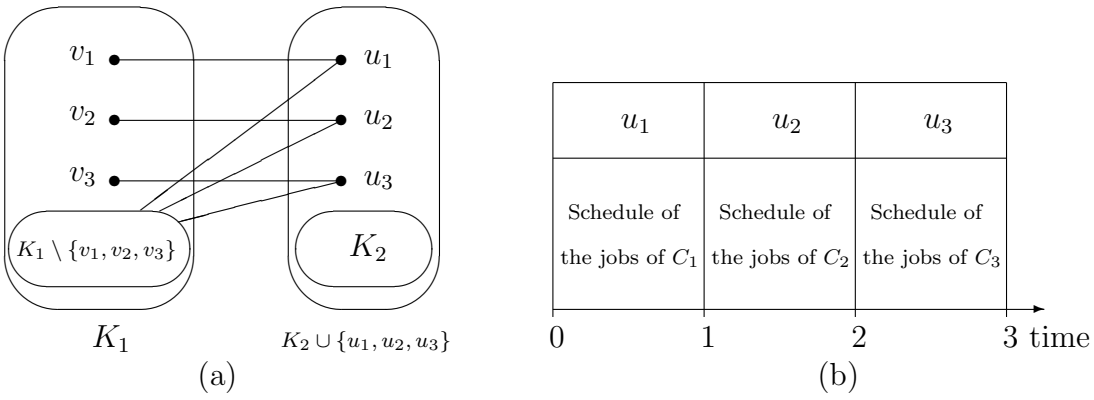

Figure 6: (a) The agreement graph G' (b) The Gantt diagram representing σ

Conversely, suppose there exists a feasible schedule σ of D' with $C_{\max}(\sigma) \leq 3$. With unit processing times, one can restrict itself to the case where jobs start at integer dates. This can be shown by rearranging jobs from left to right if they have non-integer starting dates. The jobs u_3 and v_3 having release times 2, they must have been scheduled in the same time interval $[2, 3]$. By construction, the jobs u_i and v_i must have been scheduled in $[i-1, i]$ for each i $(i = 1, 2)$. Then for each i $(i = 1, 2, 3)$ let C_i= {jobs (different from u_i) scheduled at time $i-1$}. $r_{u_3} = r_{v_3} = 2$ implies that u_3, v_3 must have been scheduled in $[2, 3]$. $r_{u_2} = r_{v_2} = 1$, this means that u_2, v_2 have been scheduled in $[1, 3]$. On the other hand u_2 is not adjacent to v_3 in G' therefore u_2 must have been scheduled in $[1, 2]$. Similarly v_2 is not adjacent to u_3 in G' therefore v_2 must have been scheduled in $[1, 2]$. With the same argument we prove that u_1, v_1 have been scheduled in $[0, 1]$. The sets (C_1, C_2, C_3) correspond to cliques in G', therefore they form a 3-coloring of G such that $v_i \in C_i$ for all i $(i = 1, 2, 3)$. It can be verified that this reduction is polynomial and that $(D') \in NP$, completing the proof of the above theorem. ■

4.2 Unit processing times and arbitrary release times with the same parity

Theorem 4 implies that the SWA problem is strongly NP-hard in the case of complement of bipartite graphs with $m = n$, $p_i = 1$ and arbitrary release times. We prove that we get a polynomial result if one restricts itself to odd release times, as stated in the next theorem.

Note that the condition $m = n$ means that there are extra machines and therefore one needs to concentrate only on the starting times of the jobs. For any job J_i let t_i represent its starting time. We start by supposing that the release times are arbitrary and odd. Suppose $K_1 = \{J_1, J_2\ , .., J_s\ \}$ and $K_2 = \{J_{s+1}, J_{s+2}\ , .., J_n\}$ and consider the following greedy algorithm.

Algorithm 2
Begin
1: **for** $i := 1$ to s **do**

2: $t_i := r_i$
3: **end for**
4: **for** $i := s+1$ to n **do**
5: **if** (all the jobs of K_1 scheduled at time r_i agree with J_i) **then**
6: $t_i := r_i$
7: **else**
8: $t_i := r_i + 1$
9: **end if**
10: **end for**

end

Theorem 5 *Algorithm 2 solves the SWA problem in the case of arbitrary complement of bipartite agreement graphs, arbitrary odd release times, unit processing times and $m = n$, polynomially in $O(n^2)$.*

Proof. Let τ be the schedule obtained by Algorithm 2 and let $C^*_{\max}$ be the optimal makespan. One can verify that τ is feasible and let us prove that it is optimal. We set $L = \max_{1\leq i\leq n}\{t_i\}$, thus $C_{\max}(\tau) = L+1$. Let J_k be a job satisfying $L = t_k$. Two possibilities may happen:
case 1: t_k is even: then by construction of the algorithm $J_k \in K_2$. On the other hand J_k must have been scheduled at step 8 of the algorithm and we have $t_k = r_k + 1$. Also, there must exist a job $J_i \in K_1$ scheduled at time t_i such that $t_i = r_k$, conflicting with J_k. Since $J_i \in K_1$ then $t_i = r_i$ and hence $r_i = r_k$. Since the jobs J_i and J_k are conflicting jobs with equal release times, then $C^*_{\max} \geq r_k + 2 = t_k + 1 = L + 1 = C_{\max}(\tau)$.
case 2: t_k is odd: so J_k has not been scheduled at step 8 of the algorithm and hence $t_k = r_k$. This implies that $C^*_{\max} \geq r_k + 1 = L + 1 = C_{\max}(\tau)$ and thus the schedule τ is optimal.
Time complexity of Algorithm 2: the for-loop through steps 1-3 can be implemented in $O(n)$ time at worse, the for-loop through steps 4-10 requires at most $O(n^2)$ iterations. Then the time complexity of Algorithm 2 equals $O(n^2)$. ■

By using a similar argument, one can prove the following result:

Corollary 6 *The SWA problem for arbitrary complement of bipartite agreement graphs with arbitrary even release times, unit processing times and $m = n$, can polynomially be solved in $O(n^2)$ time.*

4.3 Identical processing times with two distinct release times

We suppose that the processing times of all jobs are identical and equal to p with two distinct release times 0 and r. Let V_0 and V_r denote the set of jobs with release times 0 and r respectively. We recall that the agreement graph G is the complement of a bipartite graph and so it is not complete. Consider the following greedy algorithm GA.

Algorithm GA
Begin

Case 1: V_r is not a clique.
- Process the jobs of K_1 at time r.
- Process the jobs of K_2 at time $r + p$.

Case 2: Both V_r and V_0 are cliques.
- Process the jobs of V_0 at time 0.
- Process the jobs of V_r at time $\max\{p, r\}$.

Case 3: V_r is a clique and V_0 is not.

3.a: $r \geq 2p$.
- Process the jobs of $V_0 \cap K_1$ at time 0.
- Process the jobs of $V_0 \cap K_2$ at time p.
- Process the jobs of V_r at time r.

3.b: $r < 2p$.

3.b.1: If V_0 can be partitioned into 2 cliques A and B such that $B \cup V_r$ is a clique.
- Process the jobs of A at time 0.
- Process the jobs of $B \cup V_r$ at time $\max\{p, r\}$.

3.b.2: If not, there are two subcases:

If $r < p$ then
- Process the jobs of K_1 at time r.
- Process the jobs of K_2 at time $r + p$.

If $r \geq p$ then
- Process the jobs of $V_0 \cap K_1$ at time 0.
- Process the jobs of $V_0 \cap K_2$ at time p.
- Process the jobs of V_r at time $2p$.

end

In this part, we shall prove that the problem at hand is polynomial, as stated in Theorem 9. For this we start by proving the following two lemmas, needed in the proof of this theorem.

Lemma 7 *The question "can V_0 be partitioned into 2 cliques A and B such that $B \cup V_r$ is a clique of G?" can be checked in polynomial time.*

Proof. Note that, as the graph G is the complement of a bipartite graph, then V_0 can always be partitioned into two cliques A and B.
Let G_{V_0} denote the subgraph of G induced by V_0. The partitioning of V_0 into two cliques A and B, can be formulated by the equations : $x_i + x_j = 1$ for all non-agreeing pairs of jobs $J_i, J_j \in V_0$, $x_i \in \{0, 1\}$. Since A and B play a symmetric role, we may assume that the jobs J_i of A correspond to $x_i = 1$, and those of B correspond to $x_i = 0$. Thus, we have to solve the system of linear equations $MX = \vec{1}$, where $\vec{1} = (1, \ldots, 1)^t$, $X = (x_1, \ldots, x_{|V_0|})^t$ and M is the transpose of the vertex-edge incidence matrix of the complement graph of G_{V_0}. Since the latter is bipartite, then M is totally unimodular. The fact that "$B \cup V_r$ is a clique of G " is equivalent to saying that any job J_i non-adjacent with some job of V_r cannot be in B, and so it is necessarily in A. Therefore, for the condition "$B \cup V_r$ is a

clique of G ", we simply set $x_i = 1$ if the job J_i does not agree with some job of V_r. The system of linear equations $MX = \vec{1}$ for some fixed values of $X \in \{0,1\}^{|V_0|}$ and M totally unimodular is solved in polynomial time by the linear programming. ■

Lemma 8 *If V_r is a clique, V_0 is not a clique of G and V_0 cannot be partitioned into two cliques A and B such that $B \cup V_r$ is a clique of G, we have:*
(1) If $r < p$ then $C^*_{max} \geq r + 2p$.
(2) If $p \leq r < 2p$ then $C^*_{max} \geq 3p$.

Proof. We start by proving (1). To the contrary, suppose $C^*_{max} < r + 2p$, then there would exist a feasible schedule σ with $C_{max}(\sigma) < r + 2p$. Let $A = \{J_i \in V : t_i < r\}$, where t_i is the starting time of the job J_i with respect to the schedule σ. Let us show that $A \neq \phi$. Suppose $A = \phi$, then $t_i \geq r$ for all jobs J_i of V, in particular for those of V_0. As V_0 is not a clique, then there exist two non-agreeing jobs in V_0, thus $C_{max}(\sigma) \geq r + 2p$, contradiction and so $A \neq \phi$. Moreover, we claim that $A \neq V_0$ since otherwise for all jobs J_i of $V_0 : t_i < r_i$. As processing times of the jobs is p and $r < p$ then the jobs of V_0 will be pairwise concurrently scheduled in σ. This would imply that V_0 is a clique, contradiction with the hypothesis and this terminates our claim. In addition, since all the jobs of V_r have to start not before the time r, then A is a non-empty proper subset of V_0. Let $X = V \setminus A$. The set X contains the set V_r since any job J_i of V_r has a starting time $t_i \geq r$ and so it is in X. X is a clique of G, since otherwise there would be two non-agreeing jobs in X leading to $C_{max}(\sigma) \geq r + 2p$, contradiction. As $A \neq V_0$, then X can be written $X = B \cup V_r$ where $B = V_0 \setminus A$, and subsequently the sets A and B would form a partition of V_0 such that $B \cup V_r$ is a clique of G, contradicting the hypothesis. The statement (1) is proved. As far as statement (2) is concerned, one can adopt the same argument used in proving statement (1), by setting $A = \{J_i \in V : t_i < p\}$ and proceed similarly until arriving at the same contradiction with the hypothesis. This ends the proof of Lemma 8. ■

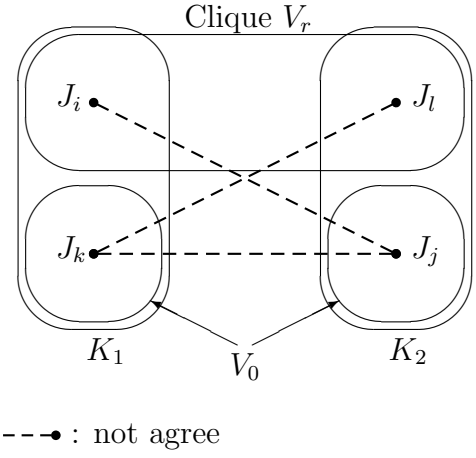

Figure 7: Illustration of Lemma 2: a situation where V_r is a clique, V_0 is not a clique and V_0 cannot be partitioned into 2 cliques A and B such that $B \cup V_r$ is a clique.

Figure 7 illustrates a possibility where V_0 is not a clique and V_r is a clique. Let us explain why V_0 cannot be partitioned into two cliques A and B such that $B \cup V_r$ is a clique. Suppose it is not the case, that is V_0 can be partitioned into two cliques A and B such that $B \cup V_r$ is a clique. Consider the pair of jobs (J_k, J_j). Since these two jobs are not adjacent then either $J_k \in B$ or $J_j \in B$ but not both. Now, in view of this figure, neither J_k nor J_j is adjacent to V_r and hence $V_r \cup B$ cannot be a clique, contradiction.

Theorem 9 *The algorithm GA cited above solves the SWA problem in the case of complement of bipartite agreement graphs, two distinct release times (*0 *and* r*), identical processing times (*$p_i = p$*) and* $m = n$*, in polynomial time.*

Proof. We denote by C^*_{max} the makespan of an optimal solution of the problem at hand. Since we have supposed that agreement graph $G = (K_1; K_2, E)$ is not complete, so there are two non-agreeing jobs, one from K_1 and the other from K_2 which implies that $C^*_{max} \geq 2p$. On the other hand $C^*_{max} \geq r + p$ ($V_r \neq \phi$ since by hypothesis 0 and r are distinct), then we obtain the inequality $C^*_{max} \geq \max\{2p, r + p\}$, which can be written

$$C^*_{max} \geq \max\{r, p\} + p \tag{1}$$

Case 1: V_r is not a clique. In this case, there are two non-agreeing jobs in V_r and thus $C^*_{max} \geq r + 2p$. therefore Algorithm GA returns an optimal solution.

Case 2: Both V_r and V_0 are cliques. The solution obtained by Algorithm GA has a makespan equals $\max\{r, p\} + p$, by inequality 1 it is optimal.

Case 3: V_r is a clique and V_0 is not. We distinguish the two sub-cases (3.a) and (3.b).

Case 3.a: $r \geq 2p$. Since $\max\{r, p\} = r$, then from inequality 1 the solution obtained by Algorithm GA is optimal with $C_{max} = r + p$

Case 3.b: $r < 2p$. Here as well, we discuss the two possibles cases (3.b.1) and (3.b.2).

Case 3.b.1: V_0 can be partitioned into two cliques A and B such that $B \cup V_r$ is a clique of G. One can verify that the schedule obtained by Algorithm GA is feasible with $C_{max} = \max\{r, p\} + p$ and hence it is optimal by inequality 1.

Case 3.b.2: V_0 cannot be partitioned into two cliques A and B such that $B \cup V_r$ is a clique in G. If $r < p$ then $C_{max} = r + 2p$ and hence, by Lemma 8, the obtained solution is optimal. A similar argument can be done when $r \geq p$ with $C_{max} = 3p$.

For all the cases cited above, the algorithm GA gives the optimal solution, and this terminates the proof of theorem 9. ■

5 Approximation results

The first result is a direct consequence of Theorem 3.

Corollary 10 *It is NP-hard to approximate with factor better than* $4/3-\epsilon$ *for any* $\epsilon > 0$*, the SWA problem with arbitrary complement of bipartite agreement graphs, unit processing times and* $r_i \in \{0, 1, 2\}$ *even if* $m = n$*.*

Proof. In the reduction used in the proof of Theorem 4, it has been proved that there is 3-coloring (C_1, C_2, C_3) of G such that $v_i \in C_i$, for all $i (i = 1, 2, 3)$ if and only if there is a feasible schedule σ for D' with $C_{\max}(\sigma) \leq 3$.
This reduction is in fact a gap reduction since one can show that if the problem 1-PrExt is ρ-approximable with $\rho < 4/3$ then it will be polynomial, and this implies $P = NP$. ■

The next result is a positive one, concerning complement of bipartite agreement graphs.

Theorem 11 *The SWA problem with arbitrary complement of bipartite agreement graphs,* $m = n$*, identical processing times* $(p_i = p)$ *and arbitrary release times has a polynomial time approximation algorithm A with performance guarantee* $|A(I) - OPT(I)| \leq p$*.*

Proof. We have $OPT(I) \geq \max\limits_{1 \leq i \leq n} \{r_i\} + p$ for all instances I. If we consider the algorithm A which schedules the jobs of one clique at time $\max\{r_i\}$ and the jobs of the other clique at time $\max\{r_i\} + p$, the makespan obtained is $A(I) \leq \max\{r_i\} + 2p$ for any instance I. Then $A(I) \leq OPT(I) + p$, which implies that $|A(I) - OPT(I)| \leq p$. ■

Note that this algorithm gives a 2-approximation for the problem at hand. It is also worth noting that in the case of unit processing times, there is an approximation which guarantees $|A(I) - OPT(I)| \leq 1$.

In reference [8], the authors have proved that for $m = 2$ there is no polynomial time approximation algorithm A for SWA with performance guarantee $|A(I) - OPT(I)| \leq K$ for any fixed constant K. Next we generalize this result for any NP-hard subproblem of SWA.

Proposition 12 *If* $P \neq NP$ *then there is no polynomial time approximation algorithms A for any NP-hard subproblem of SWA which can guarantee* $|A(I) - OPT(I)| \leq K$ *for any fixed constant* K*.*

Proof. We proceed by contradiction using a scaling argument. Assume such an algorithm A exists. Let I be an instance of SWA and I' is similar to I except that the processing times are scaled up by a factor of $K+1$. One can verify that to each feasible schedule σ for

the instance I there corresponds a feasible schedule σ' for the instance I' and vice-versa such that $C_{\max}(\sigma') = (K+1)C_{\max}(\sigma)$. We now run algorithm A on both instances I and I' and let σ_A the schedule obtained by A. Clearly $|A(I') - OPT(I')| \leq K$ which implies that σ_A) is an optimal schedule for SWA, contradicting $P \neq NP$. ■

6 Conclusion

In this paper, the problem of scheduling with agreements (SWA) is addressed. We have generalized two old NP-hardness results for SWA in the case of two machines, processing times at most 3 that hold for bipartite agreement graphs to any class of agreement graphs with a specific structure. A polynomial result for the case of split agreement graphs has been presented, followed by some complexity results in the case of complement of bipartite agreement graphs. Finally we have established some approximation results for SWA.

References

[1] B.S. Baker, E.G. Coffman, Mutual Exclusion Scheduling, Theoretical Computer Science. 162 (1996) 225–243.

[2] M. Bendraouche, M. Boudhar, Scheduling jobs on identical machines with agreement graph, Computers and Operations Research. 39 (2012) 382–390.

[3] M. Bendraouche, M. Boudhar, A. Oulamara, Scheduling: Agreement graph vs resource constraints. European Journal of Operational Research. 240 (2015) 355–360.

[4] H. L. Bodlaender, K. Jansen, Restrictions of graph partition problems part I, Theoretical Computer Science. 148 (1995) 93–109.

[5] H. L. Bodlaender, K. Jansen, G.J. Woeginger, Scheduling with incompatible jobs, Discrete Applied Mathematics. 55 (1995) 219–232.

[6] J. Cheriyan, S.N. Maheshwari, Analysis of preflow push algorithms for maximum network flow, SIAM Journal on Computing. 18 (1989) 1057–1086.

[7] M. Chrobak, J. Csirik,C. Imreh, J. Noga, J. Sgall, G.J. Woeginger, The buffer minimization problem for multiprocessor scheduling with conflicts, In Proc. 28th International Colloquium on Automata, Languages, and Programming. (2001) 862–874.

[8] G. Even, M.M. Halldorson, L. Kaplan, D. Ron, Scheduling with conflicts: online and offline algorithms, Journal of scheduling. 12 (2009) 199–224.

[9] F. Gardi, Mutual exclusion scheduling with interval graphs or related classes Part I, Discrete Appl Math. 157 (2009) 19–35.

[10] M.R. Garey, D.S. Johnson, Computers and Intractability, New York: Freeman, 1979.

[11] P. Hansen, A. Hertz, J. Kuplinski, Bounded vertex colorings of graphs, Discrete Math. 111 (1993) 305–312.

[12] S. Irani, V. Leung, Scheduling with conflicts, and applications to traffic signal control, Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms. (1996) 85–94.

[13] K. Jansen, The mutual exclusion scheduling problem for permutation and comparability graphs, Inform and Comput. 180(2) (2003), 71–81.

[14] J. Jarvis, B. Zhou, Bounded vertex coloring of trees, Discrete Math. 232 (2001) 145–151.

Les Annales RECITS (http://www.lrecits.usthb.dz)
Vol. 02, 2015, pages 27-39

Métaheuristiques pour un problème d'ordonnancement sous contraintes de préparation

Wafaa LABBI et Mourad BOUDHAR

Laboratoire RECITS, Faculté de Mathématiques, USTHB
BP 32 El-Alia, BEZ, Alger, Algerie

fawalab@yahoo.fr, mboudhar@yahoo.fr

Abstract: In this paper, we study the scheduling problem under preparation time constraints. This consists in scheduling a set of independent jobs on a set of identical parallel machines. We assume that there are k types of renewable resources needed for the preparation of the jobs. Each job has an execution time and requires prior to its execution a preparation time completed by a subset of resources. The objective is to find a schedule that minimizes the makespan. We have proposed metaheuristics to solve this problem and conducted with numerical experiments.

Keywords: Scheduling, parallel machines, makespan, heuristics, metaheuristics, resources, preparation times.

Résumé : Nous considérons le problème d'ordonnancement sous contraintes de préparation. Il consiste à ordonnancer un ensemble de tâches indépendantes sur un ensemble de machines parallèles identiques. Nous supposons qu'il existe k types de ressources renouvelables nécessaires à la préparation des tâches. Chaque tâche possède pour son traitement un temps d'exécution qui doit être précédée par une phase de préparation réalisée par un sous ensemble de ressources. L'objectif est de minimiser le makespen. Nous avons proposé des métaheuristiques pour résoudre ce dernier suivies d'une étude expérimentale.

Mots clés : Ordonnancement, machines parallèles, makespan, heuristiques, métaheuristiques, ressources, temps de préparation.

1 Introduction

Le problème que nous abordons dans ce papier est un problème d'ordonnancement sur machines identiques dont le but est de minimiser la durée totale (C_{max}). Nous nous intéressons particulièrement au cas où une tâche nécessite avant son exécution un temps de préparation réalisé par un sous ensemble de ressources renouvelables. Ce n'est qu'à partir des années 1960 que les chercheurs ont commencé à s'intéresser à ce type de problèmes. Les principaux résultats ont été résumés dans [4, 22, 5].

Le problème d'ordonnancement sous contraintes de préparation peut être décrit comme suit : Un ensemble $T = \{T_1, T_2, ..., T_n\}$ de n tâches indépendantes doit être exécuté sur un ensemble de m machines parallèles identiques. Outre que les machines, nous supposons qu'il existe un ensemble de k types de ressources renouvelables $R = \{R_1, R_2, ..., R_k\}$ disponible pour la phase de préparation où chacun est disponible en une unité. Le traitement de chaque tâche T_i nécessite un temps de préparation s_i, un temps d'exécution p_i et un sous-ensemble de ressources. Durant la phase de préparation, la machine n'est pas disponible pour une autre tâche et l'exécution d'une tâche doit être effectuée immédiatement après sa préparation. L'interruption de la préparation et de l'exécution des tâches n'est pas autorisée.

Ce problème peut trouver plusieurs applications en industrie, par exemple, il peut être issu du processus de fabrication de pneumatique. La fabrication d'un pneu passe par plusieurs phases et chaque phase du processus de fabrication requérant une grande précision, et d'importants contrôles. Nous nous intéressons de prés dans ce travail à l'étape de cuisson qui nécessite un ensemble de ressources (tel que la main-d'oeuvre, les outils) qualifié pour commencer la période de préparation de la charge de moule et le positionnement des pneus sur les machines, le pneu est au curry après la phase de préparation. Dans cette situation, le temps de traitement d'une tâche sur une machine est ainsi décomposé en deux parties : le temps de préparation et le temps d'exécution. La période de préparation doit précéder la période d'exécution pour chaque tâche et exige un certain type de ressources. Ce type de problème peut être rencontré aussi en informatique dans un environnement multiprocesseurs ou dans un réseau d'ordinateurs en présence de serveurs. Le serveur qui est une ressource se charge de l'allocation des processus aux ordinateurs ce qui correspond à la phase de préparation et le traitement des processus sur les processeurs correspond à la phase d'exécution.

Les problèmes d'ordonnancement sous contraintes de ressources pour l'affectation des tâches ont reçu peu d'attention dans la littérature, Abdelkhodaee et Wirth [1] ont étudié le problème d'ordonnancement sur deux machines parallèles identiques avec ressource supplémentaire qui est un serveur. Chaque tâche nécessite deux opérations, la première est l'opération de setup traitée par le serveur, alors que la deuxième opération est exécutée automatiquement par l'une des machines parallèles (sans serveur). L'objectif est de minimiser le makespan. Les auteurs ont prouvé que ce problème est NP-difficile au sens fort. Ils ont proposé une formulation mathématique en nombres entiers, ont présenté deux sous problèmes polynomiaux, et ont aussi proposé deux heuristiques avec des expérimentations numériques pour le problème général. L'opération de setup correspond à la phase de préparation dans notre problème, les auteurs ont relaxé la contrainte que la phase d'exécution

d'une tâche doit être effectuée immédiatement après la phase de préparation. Abdekhodaee et al. [2] ont discuté deux cas particuliers où les temps d'exécution sont identiques et les temps de setup sont aussi identiques. Ils ont montré que ces deux problème sont NP-difficile au sens faible et ont proposé des bornes inférieures ainsi que des heuristiques constructives. Suite aux travaux cités en [3], Gan et al. [14] ont développé deux formulations de programmation linéaire mixte en nombres entiers et deux variantes of a branch-and-price scheme. Koulamas [18] a traité le problème d'ordonnancement à deux machines parallèles Semi-automatiques pour minimiser le temps mort résultant de l'indisponibilité du robot avec la condition que ces deux machines partagent le même serveur (robot) pour la préparation des tâches, il a démontré que ce dernier est NP-difficile au sens fort, aussi il a développé une procédure de réduction pour le transformer en un problème de taille plus petite. Kravchenko et Werner [20] ont considéré le problème d'ordonnancement de m machines parallèles en présence d'un seul serveur dont l'objectif est de minimiser le makespan, ils ont présenté un algorithme pseudo-polynomial pour le cas de deux machines où les temps de préparation sont unitaires et ont prouvé que le problème avec un nombre arbitraire de machines et les temps de préparation sont égaux à 1 est NP-difficile, ils ont proposé des heuristiques pour sa résolution. Dans [23] les mêmes auteurs ont étudié le problème d'ordonnancement où avant le début de l'exécution d'une tâche, un temps de setup est nécessaire et doit être effectué par un ensemble de serveurs. Ils ont proposé un algorithme pseudo-polynomial pour le problème avec des temps de setup unitaires, m machines et $m-1$ serveurs et ont montré que le problème avec un nombre fixe de machines et de serveurs pour minimiser le retard maximum est NP-difficile au sens faible. Aussi, ils ont généralisé des algorithmes pour les problèmes d'ordonnancement à machines parallèles avec des temps de traitement constants au problème correspondant avec serveurs lorsque $s_i = s$ et ils ont donné l'analyse du plus mauvais cas des deux heuristiques de liste. Hasani et al. [17] ont considéré le problème d'ordonnancement sur deux machines identiques avec un seul serveur pour minimiser le makespan. Avant le traitement, chaque tâche doit être chargée sur une machine et prend un temps donné de setup qui doit être effectué par le serveur. Ils ont proposé une formulation de programmation linéaire mixte en nombres entiers en utilisant une simple idée qui consiste à une décomposition possible de l'ordonnancement en un ensemble de blocs et ils ont comparé la performance de ce modèle avec des heuristiques proposées dans [14]. Les expérimentations numériques ont montré que ce modèle performe mieux que les heuristiques.

Lorsque les ressources sont nécessaires durant l'exécution des tâches, plusieures recherches sont considérées dans la littérature, en particulier pour problèmes d'ordonnancement avec machines parallèles, [15, 9, 7]. Garey and Johnson [15] ont montré que le problème $P2|res\cdot\cdot\cdot, p_i = p|C_{max}$ est résolu polynomialement en $O(n^{5/2})$, alors que le problème $P2|res1\cdot\cdot, p_i = 1|C_{max}$ est résolu en $O(nlogn)$. Blazewicz et Ecker [10] ont montré que le problème $P|res\, sor, p_i = 1|C_{max}$ (où le nombre de types de ressources, les limites de ressources et les besoins en ressources sont fixés par les entiers positifs s, o, r, respectivement) est résolu en $O(n)$, même pour un nombre arbitraire de ressources, où les temps de traitement des tâches appartiennent à un nombre fixe de classes L, ou les tâches de même classe ont le même traitement et les besoins en ressources. Blazewicz et al. [11] ont montré que le problème $Pm|res\, sor|C_{max}$ est résolu en $O(n^{L(m+1)})$, lorsque le nombre de machines m est fixé. Blazewicz et al. [8] ont montré que le problème $P2|res\cdot 11, p_i = 1, r_j|C_{max}$ est NP-difficile au sens fort et que ce dernier est un cas particulier du problème $P2, |res^t\cdot 11, s_i = 1, p_i = p, r_i|C_{max}$ en posant $p = 0$ et en remplaçant $s_i = 1$ par $p_i = 1$, donc le problème

$P2, |res^t \cdot 11, s_i = 1, p_i = p, r_i|C_{max}$ est NP-difficile au sens fort. Dans la section 2, nous présentons les approches métaheuristques proposées pour résoudre le problème général. La section 3 porte sur l'évaluation numériques de la performance des méthodes proposés et une conclusion clôt ce tapuscrit.

2 Approche de résolution

Les métaheuristiques sont des heuristiques stochastiques itératives qui progressent vers un optimum local en se comportant comme des algorithmes de recherche, ces méthodes sont généralement hybridées avec une recherche locale. Elles sont souvent inspirées par des analogies avec des phénomènes de la nature, la physique (Recuit simulé), la biologie (algorithmes évolutionnaires) et l'éthologie (colonies de fourmis). Parmi ces méthodes, nous allons adopter deux d'entre elles : le recuit simulé et l'algorithme génétique.

2.1 Recuit simulé

Le recuit simulé est une métaheuristique inspirée d'un processus utilisé en métallurgie. Ce processus alterne des cycles de refroidissement lent et de réchauffage (recuit) qui tendent à minimiser l'énergie du matériau. Elle est aujourd'hui utilisée en optimisation pour trouver les extrémaux d'une fonction. Elle a été mise au point par trois chercheurs de la société IBM, S. Kirkpatrick, C.D. Gelatt et M.P. Vecchi en 1983 [19], et indépendamment par V. Cerny en 1985 [12]. Le recuit simulé s'appuie sur l'algorithme de Metropolis, qui permet de décrire le comportement d'un système en équilibre thermodynamique à une certaine température T, partant d'une configuration donnée (solution initiale). Par analogie avec le processus physique, la fonction objectif à minimiser deviendra l'énergie E du système. Les étapes du recuit simulé adaptées à notre problème sont les suivantes :

– Solution initiale : La solution initiale que nous avons considéré est représentée par la liste de priorité de la meilleure heuristique (voir [21]).
– Voisinage : Le mécanisme proposé pour générer des voisins d'une solution est le suivant : sans perte de généralité, supposons que le $C_{max} = maximum(C_j)$ $(j = 1..., m)$, est sur la machine M_1 et le $minimum(C_j)$ est sur la machine M_2, nous calculons l'écart $\Delta C = C_1 - C_2$. S'il existe une tâche T_i qui est traitée sur la machine M_1 avec un temps de traitement inférieur à ΔC, on la déplace à la dernière position sur la machine M_2 si elle respecte les contraintes de ressources. Sinon, on permute entre deux tâches quelconques dans la liste de priorité décrivant la solution en cours, on note le voisinage d'une liste l l'ensemble des listes $N(l)$.

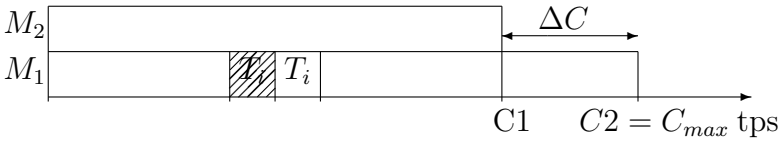

FIGURE 1 – Voisinage.

– Fonction d'évaluation : L'objectif du notre problème est de minimiser la date de fin de traitement de l'ensemble des tâches qui correspond à la minimisation de l'énergie du système. A toute solution du problème, nous appliquons les sélections Sérielle et Parallèle et nous choisissons la meilleure des deux.

Le schéma général de la méthode est donné par l'algorithme suivant :

Algorithme de recuit simulé pour le problème $Pm|res^t \cdot 11|C_{max}$

1. Initialiser l la liste de départ.
2. Initialiser une température maximale T et une température minimale $Tmin$.
3. **Tantque** $(T > Tmin)$ **faire**
4. - Choisir un voisin $l' \in N(l)$
5. - Calculer ΔC : $\Delta C = C(l') - C(l)$
6. **Si** $\Delta C < 0$ **then** $l' = l$
7. **Sinon**
8. - Générer aléatoirement une probabilité r
9. **Si** $r \leq \exp \frac{-\Delta C}{T}$ **then** $l' = l$
10. **Finsi**
11. **Finsi**
12. $T = T * (1 - alpha)$
13. **Fin**

2.2 Algorithme génétique

Les algorithmes génétiques font partie de la famille des algorithmes évolutionnaires. Ils s'inspirent de l'évolution biologique des espèces et basées sur une imitation des phénomènes d'adaptation des êtres vivants. Avec ce type de méthodes, il ne s'agit pas de trouver une solution analytique exacte mais de trouver une bonne solution satisfaisante dans un temps de calcul raisonnable. La première description du processus des algorithmes génétiques a été donnée par Holland en 1975 [13], puis Goldberg [16] les a utilisés pour résoudre des problèmes concrets d'optimisation.

Le but de ces algorithmes génétiques est d'optimiser une fonction prédéfinie, appelée fonction objectif, ou fitness ; ils travaillent sur un ensemble de solutions candidates, appelé *population* d'individus ou chromosomes. Ces derniers sont constitués d'un ensemble d'éléments, appelés *gènes*, qui peuvent prendre plusieurs valeurs, appelées *allèles*. Un chromosome est une représentation ou un codage d'une solution du problème donné. Une première population est choisie soit aléatoirement, soit par des heuristiques ou par des méthodes spécifiques au problème, soit encore par mélange de solutions aléatoires et heuristiques. Cette population doit être suffisamment diversifiée pour que l'algorithme ne reste pas bloqué dans un optimum local. C'est ce qui se produit lorsque trop d'individus sont semblables. Les algorithmes génétiques génèrent de nouveaux individus, de telle sorte qu'ils soient plus performants que leurs prédécesseurs. Le processus d'amélioration des individus s'effectue par utilisation d'opérateurs génétiques, qui sont la sélection, le croisement et la mutation. Les étapes de l'algorithme génétique adaptées à notre problème sont les suivantes :

- Codage : Le codage que nous avons retenu est de longueur égale au nombre total de tâches à réaliser. L'individu est alors représenté par une séquence de tâches. La Figure **??** présente le codage d'une solution quelconque. La tâche numéro 5 sera lancé en production la première, suivi de la tâche numéro 7... etc. et enfin la tâche numéro 6, placé dans le dernier chromosome du code, sera lancée la dernière.

- Population initiale : la population initiale que nous avons considéré est composée de séquences de tâches utilisant les règles de priorités définies dans [21] et de séquences de tâches générées aléatoirement. Le problème principal dans cette étape est le choix de la taille de la population. Si la taille de la population est trop grande, le temps de calcul augmente et demande un espace mémoire important. Par contre, une population de taille très petite, la solution obtenue n'est pas satisfaisante. Il faut donc trouver le bon compromis. Pour cela, nous avons testé le paramètre T_{int} qui représente la taille de notre population avec les valeurs $\{10, 20, 50, 100, 200\}$ et d'après les résultats obtenus, la bonne valeur est $T_{int} = 50$.
- La procédure de sélection : la sélection permet d'identifier les individus susceptibles d'être croisés dans une population. La procédure de sélection que nous avons utilisée est la sélection aléatoire, cette sélection se fait aléatoirement, uniformément et sans intervention de la valeur de la fonction objectif, donc chaque individu a une probabilité uniforme $\frac{1}{T_{int}}$ d'être sélectionné.
- Le croisement : le croisement permet d'enrichir la population en manipulant les composantes des chromosomes. Un croisement est envisagé avec deux parents et génère un ou deux enfants. Les individus survivants à la phase de sélection vont subir le croisement avec une probabilité P_{crois}, dans notre algorithme génétique, nous utilisons le croisement en 2-points. Ce dernier s'effectuera de la manière suivante :

  - Choisir deux individus de la population actuelle comme parents pour générer deux enfants.
  - Générer aléatoirement deux positions $k_1, k_2 \in [1, ..., n]$ avec $k_1 < k_2$.
  - Générer aléatoirement une probabilité de croisement α.
  - Si $\alpha \leq P_{crois}$: ce croisement conserve dans l'enfant i la zone interne du parent j (zone comprise entre k_1 et k_2)et pour l'enfant j la zone interne du parent i. Ensuite, compléter les cases vides restantes de l'enfant i par les éléments du parent i et les cases vides restantes de l'enfant j par les éléments du parent j qui ne provoquent pas de doublon.
  - Si $\alpha > P_{crois}$: copier le parent i à l'enfant i et le parent j à l'enfant j.
- La mutation : les individus issus du croisement vont ensuite subir un processus de mutation avec une probabilité P_{mut}. La mutation nous garantit que l'algorithme génétique sera susceptible d'atteindre la plupart des points du domaine réalisable. L'opérateur de mutation utilisé dans notre cas est l'opérateur d'échange réciproque qui permet de sélectionner au hasard deux gènes et de les échanger. Si β, généré aléatoirement, appartient à l'intervalle $[0, P_{mut}]$ nous appliquons l'opérateur de mutation sur l'enfant i pour avoir un enfant muté $imut$, et si $\beta > P_{mut}$ nous appliquons l'opérateur de mutation sur l'enfant j pour avoir un enfant muté $jmut$.
- Fonction d'évaluation : pour chaque chromosome, qui est une permutation de n tâches, nous appliquons les sélections Sérielle et Parallèle, ensuite nous choisissons la meilleure des deux solutions trouvées.
- Insertion : les individus qui sont générés aléatoirement et les enfants mutés sont triés

selon leur fonction d'évaluation dans un ordre croissant. Seule la moitié supérieure de la population, correspondant aux meilleurs individus, est sélectionnée. Il est à noter que la taille de la population reste fixe égale à T_{int} de génération en génération.
– Critère d'arrêt : l'algorithme génétique s'arrête après un nombre fixé de génération noté $Itermax$.

Finalement, les étapes de notre algorithme sont données par l'algorithme suivant :

Algorithm génétique pour le problème $Pm|res^t \cdot 11|C_{max}$

1. Initialiser : T_{int}, $Itermax$, P_{mut}, P_{croi}, δ.
2. Générer la population initiale de taille T_{int}.
3. **Répéter**
4. i=0
5. **Tant que** $\delta < \frac{T_{int}}{2}$ **faire**
6. Sélectionner aléatoirement deux parents de la population.
7. Croisement des deux parents pour obtenir deux enfants par une probabilité P_{croi}.
8. Muter les deux enfants par une probabilité P_{mut} pour obtenir un enfant muté.
9. $\delta = \delta + 1$.
10. **Fin Tant que.**
11. Evaluer tous les chromosomes par la fonction d'évaluation.
12. Ranger les parents et les enfants dans l'ordre croissant selon leur fonction d'évaluation.
13. Supprimer les T_{int} chromosomes faibles et enregistrer les T_{int} meilleures chromosomes selon leur fonction d'évaluation.
14. i=i+1.
15. **Jusqu'à** $i = Itermax$.

2.3 Algorithme génétique hybride

Les algorithmes génétiques possèdent une connaissance globale à travers leurs populations et explorent l'espace de recherche tandis que le recuit simulé détient une connaissance locale et exploite le voisinage. Dans cette partie, nous proposons un algorithme génétique hybride séquentielle. Notre approche se fait en trois étapes.

1. La première est une application directe de l'algorithme génétique et à la fin de cette dernière, une population d'individus qui satisfait notre critère est obtenue.
2. Dans une deuxième étape, on a recours au recuit simulé. Dans ce cas, le meilleur individu de la population finale obtenu dans la première étape est choisi comme solution initiale.
3. A la fin de la deuxième étape, on obtient une solution qui va être améliorée dans la troisième étape par l'application de l'un des algorithmes de liste basés sur les rangements des tâches définis ci-dessous.
   – CJ : A l'étape i, si la tâche $T_{j'}$ qui est à la position $i+1$ est compatible avec la tâche T_j qui est à la position i on la laisse à cette position, sinon on cherche une tâche compatible avec cette dernière et on la permute avec la tâche $T_{j'}$.

– CJI : On prend l'inverse du rangement CJ.
– ICJ : A l'étape i, si la tâche $T_{j'}$ qui est à la position $i+1$ est incompatible avec la tâche T_j qui est à la position i on la laisse à cette position, sinon on cherche une tâche incompatible avec cette dernière et on la permute avec la tâche $J_{j'}$.
– $ICJI$: On prend l'inverse du rangement ICJ.
– ER : On sélectionne deux tâches au hasard et on les permute.

Pour le choix des meilleures listes, nous avons mené une étude comparative entre eux en leurs appliquant les sélections Sérielle et Parallèle. Nous avons généré des instances aléatoires du problème pour les valeurs de m et n appartenant aux ensembles $\{2, 5, 10\}$, $\{50, 100, 500, 1000\}$, respectivement. Pour chaque valeur de n, nous avons testé 100 instances. Les temps de préparation et d'exécution des tâches sont tirés suivant la loi uniforme dans les intervalles [1, 10], [10, 50] et [50, 100]. Nous avons comparé ces heuristiques les unes par rapport aux autres en calculant le nombre de fois où une heuristique fournit de meilleures solutions. Les résultats des expérimentations numériques sont donnés dans la Table 1.

n	$\|R\|$	$s_i, p_i \in [1,10]$					$s_i, p_i \in [10,50]$					$s_i, p_i \in [50,100]$				
		CJ	**CJI**	**ICJ**	**ICJI**	**ER**	**CJ**	**CJI**	**ICJ**	**ICJI**	**ER**	**CJ**	**CJI**	**ICJ**	**ICJI**	**ER**
m=2																
50	1	12	59	34	0	20	13	46	30	0	17	7	53	37	0	4
	2	3	50	52	0	19	5	48	43	0	10	4	39	45	0	12
	5	2	58	24	11	18	4	58	17	7	14	15	44	20	10	12
	7	12	61	13	10	16	10	56	9	17	12	18	33	17	18	16
100	1	14	50	46	0	22	9	46	39	0	15	8	46	42	0	8
	2	1	49	51	0	9	2	54	40	0	9	2	56	40	0	5
	5	0	89	10	0	5	0	81	15	0	5	7	60	20	2	12
	7	0	88	7	5	5	3	72	15	7	5	22	40	14	12	13
500	1	2	50	52	0	24	4	41	42	0	18	2	47	46	0	6
	2	1	47	46	0	22	0	50	47	0	9	0	55	43	0	2
	5	0	99	4	0	0	0	97	3	0	0	0	94	6	0	0
	7	0	100	0	0	0	0	99	1	0	0	0	96	3	0	1
1000	1	7	46	47	0	28	3	47	40	0	15	1	38	53	0	8
	2	0	56	46	0	10	0	50	45	0	7	0	50	49	0	2
	5	0	95	9	0	0	0	99	1	0	0	0	99	1	0	0
	7	0	100	0	0	0	0	100	0	0	0	0	100	0	0	0
m=5																
50	1	21	44	34	0	17	28	31	20	0	21	23	32	29	0	16
	2	19	43	28	2	26	23	39	21	1	24	14	52	21	2	14
	5	4	68	17	13	12	7	62	16	7	15	4	71	15	9	8
	7	3	57	25	15	22	9	57	17	16	11	4	64	16	10	13
100	1	36	25	21	0	22	31	29	21	0	22	18	28	27	0	28
	2	16	52	20	0	22	18	45	21	0	19	7	65	14	0	15
	5	0	80	11	4	9	2	80	9	3	6	0	78	10	8	5
	7	2	85	8	4	5	1	82	4	5	10	1	86	7	4	5
500	1	20	31	22	0	30	17	37	36	0	11	25	26	25	0	25
	2	14	58	20	0	17	8	67	12	0	13	11	66	14	0	9
	5	0	97	2	0	1	0	99	1	0	0	0	98	0	0	2
	7	0	100	0	0	0	0	99	0	0	1	0	100	0	0	0
1000	1	19	41	24	0	20	21	31	26	0	22	22	25	27	0	26
	2	7	61	18	0	18	3	66	14	0	17	7	65	14	0	14
	5	0	98	2	0	0	0	98	1	0	1	0	99	1	0	0
	7	0	99	1	0	0	0	100	0	0	0	0	100	0	0	0
m=10																
50	1	38	45	29	0	30	27	36	23	0	21	29	40	24	0	23
	2	27	48	35	7	35	27	39	34	5	25	30	48	29	6	28
	5	7	51	25	21	19	10	42	25	20	20	6	63	20	6	13
	7	13	56	23	19	23	5	63	16	11	18	8	55	18	15	13
100	1	27	41	25	0	23	27	33	16	0	28	27	31	28	0	17
	2	23	45	30	0	22	21	38	21	0	25	20	39	21	2	30
	5	3	65	18	11	13	5	58	18	11	13	4	67	9	11	12
	7	1	78	11	10	10	1	78	9	9	9	0	78	10	6	8
500	1	33	30	25	0	28	28	27	24	0	23	18	22	27	0	33
	2	16	54	21	0	23	19	42	22	0	17	16	43	22	0	19
	5	0	90	8	1	1	0	76	11	4	10	3	66	18	4	9
	7	0	99	0	0	1	0	97	1	0	2	0	95	1	0	4
1000	1	29	27	20	0	31	27	24	24	0	27	21	32	25	0	22
	2	20	50	18	0	19	17	46	26	0	17	14	47	19	0	20
	5	0	96	3	0	1	0	71	14	0	15	3	62	15	5	15
	7	0	100	0	0	0	0	96	1	0	3	0	96	3	0	1

TABLE 1 – Comparaison entre les algorithmes de liste.

A partir de la Table 1, nous constatons que l'heuristique basée sur la liste CJI donne de meilleurs résultats pour toutes les instances testées. Nous remarquons qu'elle est meilleure à 100% pour la plupart des instances à 500 et 1000 tâches avec 5 et 7 types de ressources. L'heuristique basée sur la liste ICJ fournit des résultats satisfaisants pour les instances à 50 et 100 tâches avec 1 et 2 types de ressources.

3 Expérimentations numériques

Dans la section précédente, nous avons discuté des approches métaheuristiques que nous avons adapté pour la résolution du problème $Pm|res^t \cdot 11|C_{max}$. Les métaheuristiques disposent de mécanismes particuliers permettant d'éviter d'être rapidement piégées par les minimas locaux. Ces méthodes se montrent donc le plus souvent plus puissantes que les heuristiques. Cependant, le bon réglage des différents paramètres est l'étape cruciale pour que la métaheuristique retourne de bons résultats. Pour le recuit simulé, nous avons fixé ses paramètres de la manière suivante : Température initiale $T = 100$ (nous avons testé le paramètre T avec plusieurs valeurs 10, 50, 100 et 500, et d'après les résultats obtenus, la bonne valeur est $T = 100$ car nous n'avons remarqué aucune amélioration de la solution pour les autres valeurs). Température minimale $T_{min} = 10^{-4}$, donc l'algorithme s'arrête quand $T < 10^{-4}$ et pour diminuer la température nous avons choisi $\alpha = 0,3$. Le nombre de paliers effectués vaut donc 40. Pour les paramètres de l'algorithme génétique et après plusieurs expérimentations nous avons pu les fixer comme suit : Une taille de la population égale à 50, une probabilité de croisement $P_{crois} = 0.8$, une probabilité de mutation $P_{mut} = 0.1$ et en fin l'algorithme s'arrête après 100 itérations.

L'objectif de cette section est de comparer les performances de l'algorithme génétique (AG), l'algorithme génétique hybride (AGH) et le recuit simulé (RS). Nous avons généré des problèmes tests possédant différentes caractéristiques, dans un premier temps nous avons fixé le nombre de machines à $m = 2$ et 5, et pour chaque nombre de machines nous avons fait varier le nombre de tâches ($n = 50, 100, 500$) et pour chaque nombre de tâches nous avons fait varier le nombre de types de ressources ($k = 1, 2, 5, 7$). Les temps de préparation et d'exécution sont générés selon une loi uniforme dans les intervalles suivants : [1, 10],[10, 50] et [50, 100]. Les résultats de ces méthodes ont été comparés avec la borne inférieure LB. Nous avons calculé le nombre de fois où une métaheuristique fournit de meilleures solutions par rapport aux autres et le nombre de fois où la solution trouvée par une métaheuristique est égale à la borne inférieure. Aussi, nous avons calculé les déviations moyenne et maximale. Les Tables 2 et 3 illustrent ces résultats.

Sur l'ensemble des tests effectués, l'algorithme génétique se montre efficace dans la résolution de la majorité des problèmes. Il est capable de résoudre à l'optimalité tous les problèmes à 50, 100 et 500 tâches avec 1, 2 et 5 types de ressources quand les temps de préparation et d'exécution prennent leurs valeurs dans l'intervalle [1, 10]. Nous remarquons aussi que pour les instances à $n = 100$ et 500 tâches, l'algorithme génétique et l'algorithme génétique hybride sont proches en terme de performance. Toutefois, l'algorithme génétique hybride devient meilleur pour les instances à 5 et 7 types de ressources quand les temps de préparation et d'exécution prennent leurs valeurs dans les intervalles [10, 50] et [50, 100]. Par ailleurs, le recuit simulé est le moins performant des trois.

n	$\|R\|$		$s_i, p_i \in [1,10]$			$s_i, p_i \in [10,50]$			$s_i, p_i \in [50,100]$		
			AG	**AGH**	**RS**	**AG**	**AGH**	**RS**	**AG**	**AGH**	**RS**
50	1	#Best	100	86	79	100	79	14	98	92	5
		#LB	100	86	79	100	79	14	95	90	5
		$\%Dev_{max}$	0	1,06	6,36	0	0,59	5,31	0,4	0,32	4,0
		$\%Dev_{moy}$	0	0,06	0,29	0	0,03	0,77	0,008	0,01	1,28
	2	#Best	100	74	54	96	88	4	94	83	1
		#LB	100	74	54	96	88	4	88	78	1
		$\%Dev_{max}$	0	4,16	14,23	0,13	0,32	7,58	0,71	0,6	4,39
		$\%Dev_{moy}$	0	0,21	0,93	0,003	0,01	1,56	0,02	0,03	1,82
	5	#Best	97	53	2	81	71	0	61	72	0
		#LB	95	52	2	50	51	0	36	37	0
		$\%Dev_{max}$	0,37	5,0	15,38	1,67	2,16	9,58	0,9	1,05	8,32
		$\%Dev_{moy}$	0,01	0,74	4,33	0,14	0,21	4,41	0,17	0,14	3,47
	7	#Best	90	62	0	54	59	0	62	44	0
		#LB	52	38	0	13	9	0	1	2	0
		$\%Dev_{max}$	2,43	10,76	18,05	2,15	1,91	12,05	1,63	1,45	8,73
		$\%Dev_{moy}$	0,28	1,07	6,44	0,52	0,49	6,57	0,5	0,56	5,21
100	1	#Best	100	84	84	100	74	16	90	97	1
		#LB	100	84	84	100	74	16	90	97	1
		$\%Dev_{max}$	0	0,35	0,36	0	0,13	4,46	0,17	0,09	1,98
		$\%Dev_{moy}$	0	0,03	0,14	0	0,01	0,48	0,003	0,003	0,7
	2	#Best	100	67	49	99	84	5	87	92	0
		#LB	100	67	49	99	84	5	81	86	0
		$\%Dev_{max}$	0	0,53	6,66	0,03	0,41	4,61	0,22	0,15	2,67
		$\%Dev_{moy}$	0	0,07	0,64	$3 * 10^{-6}$	0,02	0,97	0,01	0,01	1,05
	5	#Best	88	64	0	79	81	0	55	70	0
		#LB	83	58	0	64	63	0	23	29	0
		$\%Dev_{max}$	0,37	3,11	8,08	1,41	0,72	5,64	0,76	0,66	5,29
		$\%Dev_{moy}$	0,04	0,44	3,12	0,09	0,08	2,67	0,14	0,13	2,55
	7	#Best	80	60	0	55	59	0	51	53	0
		#LB	41	29	0	11	8	0	0	0	0
		$\%Dev_{max}$	1,88	5,28	9,57	2,43	2,9	9,39	1,47	1,34	6,43
		$\%Dev_{moy}$	0,35	0,62	5,11	0,51	0,51	4,67	0,45	0,44	3,93
500	1	#Best	100	90	87	100	85	12	90	100	3
		#LB	100	90	87	100	85	12	90	100	3
		$\%Dev_{max}$	0	0,03	0,58	0	0,04	1,51	0,03	0	0,75
		$\%Dev_{moy}$	0	0,003	0,01	0	0,002	0,16	0,001	0	0,16
	2	#Best	100	75	62	100	86	1	100	96	0
		#LB	100	75	62	100	86	1	99	96	0
		$\%Dev_{max}$	0	0,07	1,92	0	0,09	1,73	0,005	0,04	0,86
		$\%Dev_{moy}$	0	0,009	0,15	0	0,003	0,25	$5*10^{-7}$	$6*10^{-6}$	0,25
	5	#Best	100	80	0	97	98	0	64	79	0
		#LB	100	80	0	97	98	0	44	51	0
		$\%Dev_{max}$	0	0,33	4,03	0,1	0,01	2,63	0,19	0,14	1,55
		$\%Dev_{moy}$	0	0,03	1,22	0,001	$2*10^{-6}$	0,68	0,031	0,026	0,94
	7	#Best	84	89	0	65	70	0	53	55	0
		#LB	71	71	0	39	39	0	7	8	0
		$\%Dev_{max}$	0,25	0,25	5,08	0,58	0,8	2,98	0,48	0,41	2,79
		$\%Dev_{moy}$	0,02	0,02	1,46	0,08	0,08	1,73	0,16	0,16	1,78

TABLE 2 – Etude comparative entre les métaheuristiques pour $m = 2$.

Plus le nombre de machines augmente, plus l'écart moyen entre les méthodes et la borne inférieure augmente. Pour $m = 5$ machines, l'algorithme génétique apparaît relativement plus performant dans le cas où on a 1 et 2 types de ressources. La déviation moyenne est inférieure à $0,6\%$ pour $k = 1$ et inférieure à 4% pour $k = 2$. En effet, pour $k = 5, 7$, l'algorithme génétique hybride est en mesure de fournir des solutions meilleures que l'algorithme génétique. Par exemple, pour $n = 500$ et $k = 5$ il donne une solution réalisable dans 74 cas et une déviation moyenne inférieure à $3,3\%$ pour le premier intervalle et il trouve une solution réalisable dans 60 cas pour $k = 7$ et une déviation moyenne ne dépassant pas $3,7\%$.

4 Conclusion

Dans ce papier, nous avons considéré le problème avec un nombre arbitraire de types de ressources tel que chaque tâche nécessite soit 0 soit 1 unité de chaque type pour la phase de préparation. Ce papier a été dédié à la résolution de ce problème et ceci par des méthodes approchées basées sur des métaheuristiques. Nous avons aussi mené différents tests expérimentaux sur des instances générées aléatoirement pour évaluer la performance des métaheuristiques.

n	$\lvert R\rvert$		$s_i, p_i \in [1,10]$			$s_i, p_i \in [10,50]$			$s_i, p_i \in [50,100]$		
			AG	**AGH**	**RS**	**AG**	**AGH**	**RS**	**AG**	**AGH**	**RS**
50	1	#Best	97	10	0	97	4	0	100	0	0
		#LB	75	4	0	51	0	0	62	0	0
		$\%Dev_{max}$	3,63	15,78	30,55	4,34	18,91	28,54	4,98	20,41	33,51
		$\%Dev_{moy}$	0,39	6,96	9,53	0,32	9,62	10,71	0,35	10,52	14,67
	2	#Best	90	54	0	81	36	0	91	12	0
		#LB	69	37	0	23	3	0	26	0	0
		$\%Dev_{max}$	10,31	21,42	30,83	3,37	12,85	22,06	7	13,3	29,66
		$\%Dev_{moy}$	0,76	2,43	10,6	0,69	2,8	9,23	0,81	4,34	9,68
	5	#Best	96	62	0	93	97	14	87	95	0
		#LB	45	29	0	42	43	0	23	23	0
		$\%Dev_{max}$	16,1	19,08	34,35	16,71	16,56	28,63	12,03	12,03	31,74
		$\%Dev_{moy}$	2,3	3,85	12,86	1,32	1,33	10,94	2,43	2,41	12,99
	7	#Best	98	57	0	79	91	0	88	95	0
		#LB	49	28	0	29	27	0	23	24	0
		$\%Dev_{max}$	8,72	12,79	29,86	10,71	10,96	25,45	8,8	8,7	30,03
		$\%Dev_{moy}$	1,22	2,52	9,29	1,17	1,16	9,36	1,39	1,4	9,78
100	1	#Best	98	3	0	100	1	0	100	0	0
		#LB	74	0	0	48	0	0	41	0	0
		$\%Dev_{max}$	5,02	15,0	18,72	4,41	20,45	26,79	2,62	18,17	28,66
		$\%Dev_{moy}$	0,4	8,55	9,26	0,04	11,48	11,62	0,12	12,0	14,63
	2	#Best	78	55	0	68	34	0	77	25	0
		#LB	25	10	0	8	0	0	8	0	0
		$\%Dev_{max}$	9,15	8,05	23,25	9,54	10,69	21,76	7,67	10,79	22,88
		$\%Dev_{moy}$	0,8	1,58	9,3	1,83	3,18	9,32	1,86	4,49	10,8
	5	#Best	85	86	0	70	84	0	68	85	0
		#LB	29	31	0	13	16	0	17	17	0
		$\%Dev_{max}$	11,9	11,5	36,8	14,29	14,22	24,08	11,32	11,22	25,49
		$\%Dev_{moy}$	2,75	2,33	12,9	2,35	2,32	13,2	2,02	1,9	13,19
	7	#Best	68	94	0	76	89	0	85	98	0
		#LB	13	14	0	10	11	0	6	7	0
		$\%Dev_{max}$	18,41	10,63	30,32	10,54	10,17	23,7	8,54	8,51	24,65
		$\%Dev_{moy}$	2,85	2,46	12,48	1,77	1,77	12,35	1,87	1,86	12,45
500	1	#Best	100	0	0	100	0	0	100	0	0
		#LB	18	0	0	6	0	0	0	0	0
		$\%Dev_{max}$	3,15	14,18	13,76	5,6	18,55	17,27	1,57	17,09	21,0
		$\%Dev_{moy}$	0,59	10,7	10,2	0,36	12,9	12,21	0,27	14,23	16,13
	2	#Best	43	60	0	50	50	0	70	30	0
		#LB	0	0	0	0	0	0	0	0	0
		$\%Dev_{max}$	4,52	5,19	10,39	8,39	8,33	14,59	6,82	7,47	13,56
		$\%Dev_{moy}$	3,15	3,08	7,8	4,74	4,73	9,01	4,32	5,21	11,02
	5	#Best	45	74	0	50	75	0	50	53	0
		#LB	0	0	0	0	0	0	0	0	0
		$\%Dev_{max}$	8,02	7,33	21,42	8,23	8,68	20,9	4,67	4,65	19,6
		$\%Dev_{moy}$	3,34	3,21	13,29	2,42	2,33	14,36	2,006	1,9	14,15
	7	#Best	45	60	0	34	72	0	40	74	0
		#LB	0	0	0	0	0	0	0	0	0
		$\%Dev_{max}$	8,23	8,1	21,88	8,47	7,7	22,31	7,03	6,3	23,2
		$\%Dev_{moy}$	3,74	3,61	15,63	3,13	3,0	11,53	2,73	2,52	15,9

TABLE 3 – Etude comparative entre les métaheuristiques, $m = 5$.

Références

[1] Abdelkhodaee A.H., Wirth A. Scheduling parallel machines with a single server : some solvable cases and heuristics, Computers & Operations Research, 29(3), 295–315, 2002.

[2] Abdelkhodaee A.H., Wirth A., Gan H.S. Equal processing and equal setup time cases of Scheduling parallel machines with a single server. Computers & Operations Research, 31, 1867–1889, 2004.

[3] Abdelkhodaee A.H., Wirth A., Gan H.S. Scheduling two parallel machines with a single server : the general case. Compputers & Operations Research, 33, 994–1009, 2006.

[4] Allahverdi A., Gupta J.N.D., Aldowaisan T. A review of scheduling research involving setup considerations. OMEGA The International Journal of Management Sciences, 27, 219–239, 1999.

[5] Allahverdi A., Ng C.T., Cheng T.C.E., Kovalyov M.Y. A survey of scheduling problems with setuo times or costs. European Journal of Operational Research, 187, 985–1032, 2008.

[6] Blazewicz J. Selected topics in scheduling theory. Annals of Discrete Mathematics, vol. 31, 1-61, 1987.

[7] Blazewicz J., Cellary W., Slowinski R., Weglarz J. Scheduling under resource constraints : deterministic models. Annals of Operations Research, vol. 7, 1986.

[8] Blazewicz J., Ecker K.H., Pesch E.,Schmidt G. and Weglarz J. Scheduling Computer and Manufacturing Processes. Springer-Verlag, Berlin, 2001.

[9] Blazewicz J., Lenstra J.K., Rinnooy Kan A.H.G. Scheduling subject to resource constraints : classification and complexity. Discrete Applied Mathematics, vol. 5, 11-24, 1983.

[10] Blazewicz J., Ecker K. A linear time algorithm for restricted bin packing and scheduling problems. Oper. Res. Lett., 2, 80, 1983.

[11] Blazewicz J., Kubiak W., Szwarcfiter J. Scheduling independent fixed-type tasks. In R. Slowinski and J. Weglarz (eds). Advances in Project Scheduling, Elsevier, Amsterdam, 225, 1989.

[12] Cerny, V. Thermodynamical approach to the traveling salesman problem : an efficient simulation algorithm. J. of Optimization Theory and Applications, tome 45, n°1, P. 41-51, 1985.

[13] Holland J.H. Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, MI, 1975.

[14] Gan HS., Wirth A., Abdelkhodaee A. A branche-and-price algorithm for the general case of scheduling parallel machines with a single server. Computer and Operations Research, 39, 2242-2247, 2012.

[15] Garey M.R., Johnson D.S. Complexity results for multiprocessor scheduling under resource constraints. SIAM J. Comput. 4, 397, 1974.

[16] Goldberg, D. Genetic algorithm in search, optimisation and machine learning. Addison Wesley. 36, 1989.

[17] Hasani K., Kravchenko S.A, Werner F.Block models for scheduling jobs on two parallel machines with a single server. Computer and Operations Research, 41, 94–97, 2014.

[18] Koulamas, CP. Scheduling two parallel semiautomatic machines to minimize machine interference. Computers and Operations Research, 23(10), 945–956, 1996.

[19] Kirkpatrick S., Gelatt C., Vecchi M. Optimization by simulated annealing. Science, tome 220, n°4598, P. 671–680, 1983.

[20] Kravchenko S.A., Werner F. Parallel machine scheduling problem with a single server. Mathematical and Computer Modelling, 26(12), 1–11, 1997.

[21] Labbi, W., Boudhar, M., Oulamara, A. Scheduling two identical parallel machines with preparation constraints. International Journal of Production Research, DOI : 10.1080/00207543.2014.978032. Accepted. To appear.

[22] Yang W.H., Liao C.J. Survey of scheduling research involving setup times. International Journal of Systems Science, 30, 143–155, 1999.

[23] Werner F., Kravchenko S.A. Scheduling with Multiple Servers. Automation and Remote Control, vol. 71(10), 2109–2121, 2010.

Présentation :

RECITS est un laboratoire de recherche opérationnelle, de combinatoire, d'informatique théorique et de méthodes stochastiques agréé par l'arrêté ministériel n° 242 du 03 avril 2013. Ses domaines de compétence en recherche appliquée concerne principalement l'aide à la précise de décision par une méthodologie scientifique aboutissant à la conception de méthodes et d'algorithmes performants pour la résolution de problèmes fortement combinatoires. Les domaines d'application de nos activités de recherche sont orientés vers des entreprises et/ou des organismes publics ou privés. Nous citons à titre d'exemples : le secteur de la sécurité informatique, le secteur de la production industrielle, les hôpitaux, les banques, les compagnies d'assurance, la formation professionnelle, etc.

Le Laboratoire RECITS possède une compétence dans des domaines comme l'ordonnancement, le transport, le codage, la théorie des graphes, la combinatoire, la prévision et la prospective, les enquêtes et sondages, l'analyse des données, l'optimisation, la théorie de la décision, la gestion du risque, etc. Au-delà de ces domaines, ses compétences touchent à l'étude et à la résolution de problèmes fortement combinatoires, de par sa maîtrise des outils comme la recherche opérationnelle, les logiciels de résolution et de programmation avancée, les algorithmes évolutionnistes, etc.

Le laboratoire dispose d'équipes de recherche très actives dont la complémentarité des actions nous fournit tous les atouts pour la production de logiciels performants d'aide à la décision, utilisant des méthodes originales préalablement étudiées de manière théorique. Il est structuré en cinq équipes de recherche :

CATI Combinatoire, Arithmétique et Informatique Théorique

RO-TOP Recherche Opérationnelle pour le Transport et l'Ordonnancement en Productique

CEAPA Combinatoire Enumérative, Analytique, Probabiliste et Applications

STEP Séries Temporelles, Econométrie et Probabilités

O2M Optimisation à Objectifs Multiples

Contacts :

Web : **http://www.lrecits.usthb.dz**

Email : **lrecits@usthb.dz**

Tél :

Fax :