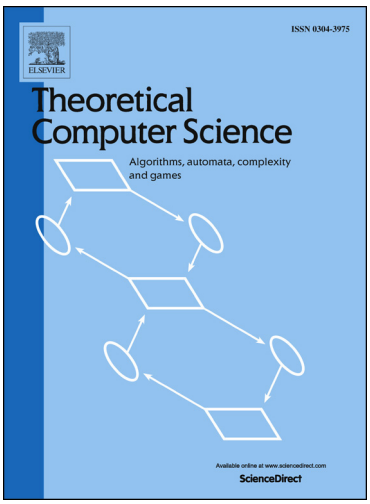
Accepted Manuscript



New results in two identical machines scheduling with agreement graphs

Amine Mohabeddine, Mourad Boudhar

PII: S0304-3975(19)30079-9
DOI: https://doi.org/10.1016/j.tcs.2019.01.038
Reference: TCS 11894

To appear in: *Theoretical Computer Science*

Received date: 3 September 2018
Revised date: 8 January 2019
Accepted date: 28 January 2019

Please cite this article in press as: A. Mohabeddine, M. Boudhar, New results in two identical machines scheduling with agreement graphs, *Theoret. Comput. Sci.* (2019), https://doi.org/10.1016/j.tcs.2019.01.038

Highlights

- The Scheduling problem with agreement graph on two identical machines is NP-hard for trees.
- The Scheduling problem with agreement graph on two identical machines is solvable in polynomial time for caterpillars (and also path and star graphs).
- The Scheduling problem with agreement graph on two identical machines is solvable in polynomial time for cycles.

New results in two identical machines scheduling with agreement graphs

Amine Mohabeddine, Mourad Boudhar*

RECITS Laboratory, Faculty of Mathematics, USTHB University, BP 32 El-Alia, Bab-Ezzouar, Algiers, Algeria

Abstract

Scheduling problem with agreement graph on two identical machines is addressed. The problem consists in scheduling a set of non-preemptive jobs on two identical machines in a minimum time. Agreement constraints are imposed over the jobs. They express that only specific jobs can be scheduled concurrently on different machines. These constraints are represented by an agreement graph. This problem can be seen as a problem of partitioning a vertex-weighted graph into cliques. The problem is NP-hard for an arbitrary agreement graph, but for some particular graphs the problem is still open. For this reason, we study the complexity of the problem for some specific graphs. In particular, we prove the NP-hardness of the problem if the agreement graph is a tree. We also propose polynomial time algorithms to solve the problem for the cases of caterpillars and cycles.

Keywords: Scheduling; Identical machines; makespan; Agreement graph; Conflict graph; Complexity.

1. Introduction

In this paper, we consider the problem of scheduling with agreements (SWA in short) on two identical machines. The problem consists in scheduling a set J of n non-preemptive jobs on two identical machines. Each job $j \in J$ has a processing time p_j. The schedule is built according to agreement constraints between jobs modeled by an agreement graph $G = (J, E)$. Two jobs are connected in the agreement graph G if and only if these jobs can be scheduled concurrently on different machines. The makespan (C_{max}) is the completion time of the last completed job. The objective of this problem is to minimize the makespan.

This problem was initially introduced as a problem of scheduling with conflicts (SWC in short) by Even et al. [1]. Instead of an agreement graph, the

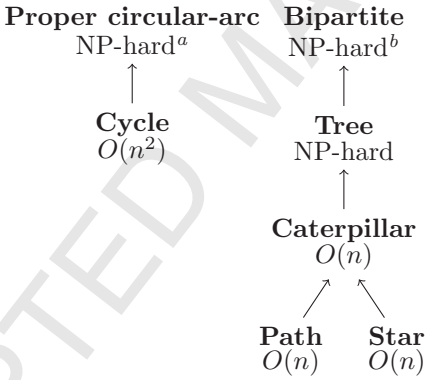
*Corresponding author

Email addresses: a.mohabeddine@yahoo.fr (Amine Mohabeddine), mboudhar@yahoo.fr (Mourad Boudhar)

authors considered a conflict graph (the compliment of the agreement graph) where each edge connects a pair of conflicting jobs that cannot be scheduled concurrently on different machines. It is clear that the SWA problem and SWC problem are polynomially equivalent; for this reason, we are also interested in the SWC problem particularly in the literature review section.

The SWA problem appears generally in the literature as a problem of scheduling with non-sharable resource constraints. A pair of non-agreeing jobs (conflicting jobs) represents jobs sharing the same resource [2]. In the literature, one can find other applications, Bendraouche and Boudhar [3] presented an application for the management of a workshop derived from the resource constrained problem. Baker and Coffman [4] proposed an application in balancing the load in a parallel computation. Halldórsson et al. [5] mentioned more applications in traffic intersection control, frequency assignment in cellular networks, and session management in local area network.

The cartography given in Figure 1 summarizes the results presented in this paper and situates them in relation to some other known cases of the problem on two machines.



[a]Garey and Johnson [6] have proved the NPhardness of a problem equivalent to the SWA for the case of two machines and complete agreement graphs which is a subproblem of the SWA problem with proper circular-arc graphs.

[b]Even et al. [1] and Bendraouche and Boudhar [3] proved its NP-hardness.

Figure 1: Complexity hierarchy of the studied cases.

The present paper is organized as follows. In Section 1, we have introduced the SWA problem by giving its definition and some applications. In Section 2, we present an overview of the previous work. In Section 3, we prove the NP-hardness of the SWA problem if the agreement graph is a tree. In Sections 4 and 5, we present two polynomial time algorithms which optimally solve the problem when the agreement graph is either a caterpillar or a cycle. The conclusion constitutes the final Section.

2. Literature review

As mentioned above, the SWC problem was firstly introduced by Even et al. [1]. The authors have proposed a polynomial time algorithm to solve the SWC problem on two identical machines for processing times $p_j \in \{1, 2\}$. They also proposed a $\frac{4}{3}$-approximation algorithm to solve the SWA problem on two identical machines for processing times $p_j \in \{1, 2, 3\}$. The authors proved that the SWA problem on two machines for a bipartite graph and processing times $p_j \in \{1, 2, 3, 4\}$ is APX-hard. The second part of their work was dedicated to the online version of the problem.

Bendraouche and Boudhar [3] have proved the NP-hardness of the SWA problem on two machines for a bipartite graph and processing times $p_j \in \{1, 2, 3\}$. They also proved the NP-hardness of the problem on two machines for a bipartite graph, processing times $p_j \in \{1, 2\}$, and release dates $r_j \in \{0, r\}$. On the other hand, they proposed a polynomial time algorithm to solve the problem on two identical machines for $\overline{G}$ being a bipartite graph and for which the jobs of one part of the bipartite graph have unit processing times. Some list scheduling algorithms for the approached resolution of the SWA problem on m identical machines was proposed by the authors as well.

For two machines with processing times $p_j = a$, Garey and Johnson [7] proved that SWA can be solved in $O(n^{2.5})$. Bendraouche et al. [8] extended the polynomial status of SWA for $p_j \in \{1, 2\}$ to $p_j \in \{a, 2a\}$. In the same paper, these authors proved the NP-hardness of this problem on two machines, bipartite agreement graphs and two different processing times: $p_j \in \{a, 2a + b\}$ $(b \neq 0)$. By this result, Bendraouche et al. [8] thoroughly closed the complexity status of SWA for two machines with at most two processing time values. In the second part of their study, the authors have proved the equivalence between the SWA problem and the resource constrained scheduling problem [9] where the availability of each resource is equal to one and the requirement of each job for any resource is at most one.

In addition, Bendraouche and Boudhar [10] have proved the NP-hardness of the SWA problem on two machines for agreement graphs of the form $G = (A \cup B, E)$ where sets A and B have a particular structure. They have also proposed some polynomial cases for specific split graphs and complements of bipartite graphs.

Baker and Coffman [4] have called Mutual Exclusion Scheduling (MES) the SWC problem in which each job requires one unit of processing time. This problem corresponds exactly to a minimum coloring of the conflict graph G such that each color appears at most m times. The MES problem is NP-hard since the problem of finding a minimum coloring of a graph is NP-hard. However, there are many special graph classes for which the MES problem is solvable in polynomial time. For more results about the MES problem, the interested readers are referred to the papers [3, 11].

Previous results dealing with the complexity of the MES problem for m machines are summarized as follows. The MES problem is polynomial for: bipartite graphs and co-graphs with a fixed number of machines, interval graphs

for $m \geq 4$ (see Bodlaender and Jansen [12]), permutation graphs for $m \geq 6$ (see Jansen [13]) and bounded tolerance graphs for $m \geq 3$ (see Gardi [11]). The MES problem has been proved to be NP-hard on: split graphs for $m \geq 3$ (see Gardi [11]), bipartite graphs, and forests for $m \geq 3$ (see Bodlaender and Jansen [12]), collection of disjoint cliques for $m \geq 3$ (see De Werra [14]), co-graphs (see Bodlaender and Jansen [12]), and proper circular arc graphs for $m \geq 3$ (see Gardi [11]).

Tellache and Boudhar [15] have developed mathematical models and a branch and bound algorithm for the two-machine flow shop problem with unit-time operations and conflict graph. They also studied in [16] the problem of scheduling with conflict graph but in an open shop system: they discussed the complexity of different versions of the problem and they presented lower bounds and heuristic algorithms. In [17], Tellache and Boudhar have proved that the flow shop problem is NP-hard for several conflict graphs. Then, they presented polynomial-time solvable cases. On the other hand, they proposed heuristics and lower bounds alongside with an experimental study.

3. Trees

As mentioned in the introduction, we will prove the NP-hardness of the SWA problem on two machines if the agreement graph is a tree. To prove its NP-hardness, we will polynomially reduce the partition problem to the decision problem derived from the SWA problem on two machines denoted DSWA.

Definition of the DSWA problem. We consider the decision problem associated with the SWA on two identical machines denoted DSWA problem. The DSWA problem is defined as follows. Instance: a set of jobs J, processing times p_j, $\forall j \in J$, an agreement graph $G = (J, E)$, and a positive integer $K < \sum_{j \in J} p_j$. Question: is there a feasible schedule σ of the SWA problem with agreement graph G on two identical machines such that $C_{max}(\sigma) \leq K$?

Partition problem. Instance: a finite set A of n elements, a size $s(a_i)$ for each element $a_i \in A$ and a positive integer $B = \frac{\sum_{a_i \in A} s(a_i)}{2}$. Question: is there a subset $A' \subseteq A$ such that $\sum_{a_i \in A'} s(a_i) = B$? This problem is NP-complete [6].

Theorem 1. *The SWA problem on two machines where the agreement graph is a tree is NP-hard.*

Proof. We consider the partition problem defined above. We prove that the partition problem can polynomially be reduced to the DSWA problem if the agreement graph is a tree.

Given an instance of the partition problem, our corresponding instance of the DSWA problem is defined as follows. We consider a set J of $2n+5$ jobs and a tree $T = (J, E)$. J is composed of two subsets X and Y of n jobs each and five jobs u, v_1, v_2, w_1 and w_2. The processing times of these jobs are given in Table 1.

Table 1: Processing times of the jobs of the DSWA problem.

Subsets	-	-	-	X	Y
Jobs	u	v_i	w_i	x_i	y_i
Proces. times	$2B+2$	2	1	$2s(a_i)$	$s(a_i)$
i	-	$i \in \{1,2\}$	$i \in \{1,2\}$	$i = 1,\ldots,n$	$i = 1,\ldots,n$

The jobs that are in correspondence with the elements of A are the jobs of both sets X and Y, where each job x_i and each job y_i are in correspondence with the element a_i $(i = 1,\ldots,n)$. One can view the jobs of X as the "actual" corresponding jobs and the jobs of Y as "dummy" corresponding jobs. The tree T is shown in Figure 2 and it is described as follows: job u is connected to all the jobs of X and to jobs v_1 and v_2. Every job x_i is connected to its corresponding job y_i, for $i = 1,\ldots,n$. Similarly, each job v_i is connected to its corresponding job w_i, for $i \in \{1,2\}$.

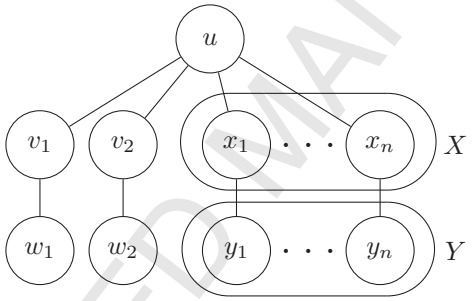


Figure 2: Agreement graph (tree) T.

To prove that the DSWA problem is NP-complete, we have to prove that there exists a subset $A' \subseteq A$ such that $\sum_{a_i \in A'} s(a_i) = B$ if and only if there exists a feasible schedule σ of the SWA problem on two machines with the agreement graph T such that $C_{max}(\sigma) \leq 5B + 4$.

First, suppose that $A' \subseteq A$ is a subset of A such that $\sum_{a_i \in A'} s(a_i) = B$. The schedule σ is shown in Figure 3 and is described as follows: job v_i $(i \in \{1,2\})$ is scheduled concurrently with jobs u and w_i. The jobs of $X' \subseteq X$, that correspond to the elements of A', are scheduled between jobs v_1 and v_2 opposite to job u. The remaining jobs of X denoted X'' are scheduled opposite to their neighbors $Y'' \subseteq Y$ and the remaining jobs of Y denoted Y' are scheduled opposite to an idle time. Of course, the jobs of Y'' have to be separated by idle times, but to facilitate the readability of figures they are represented as a single bloc of jobs, which has no incidence on the proof. The schedule σ has the following makespan:

$$C_{max}(\sigma) = p_{v_1} + \sum_{j \in X'} p_j + p_{v_2} + \sum_{j \in X''} p_j + \sum_{j \in Y'} p_j = 2+2B+2+2B+B = 5B+4$$

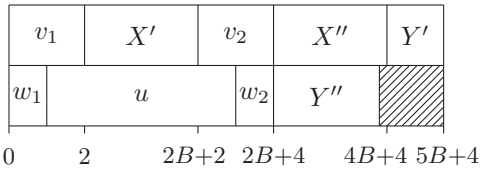


Figure 3: Schedule σ (the jobs of Y'' are separated by idle times).

Conversely, suppose there exists a feasible schedule σ of the SWA problem on two machines with the agreement graph T such that $C_{max}(\sigma) \leq 5B+4$. We denote by $X' \subseteq X$ the subset of jobs that are scheduled opposite to the job u, the set of the remaining jobs of X is denoted X''. The subset of jobs of Y, that are adjacent to the jobs of X', is denoted Y', and the set of the remaining jobs of Y is denoted Y''.

Now, suppose that job w_i is entirely scheduled in parallel with v_i, and v_i is scheduled, as much as possible, in parallel with u ($i \in \{1,2\}$). So, the makespan of the schedule σ depends on the sum of the processing times of the jobs of X'. Thus, we have three possible cases:

1. $\sum_{j\in X'} p_j = 2B+\epsilon$ ($\epsilon > 0$): the schedule σ is described in Figure 4 and has the following makespan:

$$\begin{aligned} C_{max}(\sigma) &= p_{v_1} + \sum_{j\in X'} p_j + p_{v_2} + \sum_{j\in X''} p_j + \sum_{j\in Y'} p_j \\ &= 2 + (2B+\epsilon) + 2 + (2B-\epsilon) + (B+\frac{\epsilon}{2}) = 5B+4+\frac{\epsilon}{2} \end{aligned}$$

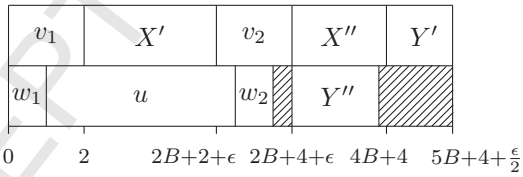


Figure 4: Schedule σ in the first case (the jobs of Y'' are separated by idle times).

2. $\sum_{j\in X'} p_j = 2B-\epsilon$ ($\epsilon > 0$): the schedule σ is described in Figure 5 and has the following makespan:

$$\begin{aligned} C_{max}(\sigma) &= p_{w_1} + p_u + p_{w_2} + \sum_{j\in X''} p_j + \sum_{j\in Y'} p_j \\ &= 1 + (2B+2) + 1 + (2B+\epsilon) + (B-\frac{\epsilon}{2}) = 5B+4+\frac{\epsilon}{2} \end{aligned}$$

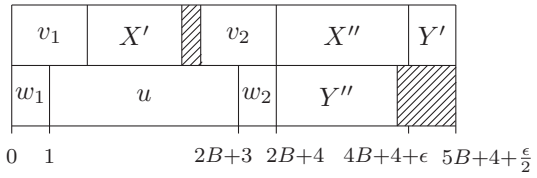


Figure 5: Schedule σ in the second case (the jobs of Y'' are separated by idle times).

3. $\sum_{j\in X'} p_j = 2B$: the schedule σ is described in Figure 3 and has the following makespan:

$$\begin{aligned} C_{max}(\sigma) &= p_{v_1} + \sum_{j\in X'} p_j + p_{v_2} + \sum_{j\in X''} p_j + \sum_{j\in Y'} p_j \\ &= 2 + 2B + 2 + 2B + B = 5B + 4 \end{aligned}$$

Now, we discuss the cases where the assumptions regarding w_i and v_i ($i \in \{1, 2\}$) do not hold. If w_i is not scheduled opposite to v_i, the time interval opposite to w_i will be vacant and no other job than cover this time interval, and it is the same for the job v_i. In fact, if v_i is not scheduled opposite to w_i and u, then it will create a time interval opposite to v_i that could not be filled with any job. So, the makespan of σ will not decrease if one of these configuration occurs. For the first two cases, we have $C_{max}(\sigma) > 5B + 4$, so whatever the position of w_i and v_i are, the makespan of σ remains greater than $5B+4$. Thus, $\sum_{j\in X'} p_j = 2B$ (case 3) is the only possible case that satisfy $C_{max}(\sigma) \leq 5B+4$.

Hence, the set A' composed of the elements that are in correspondence with the jobs of X' is a subset of A such that $\sum_{a_i\in A'} s(a_i) = B$. □

4. Caterpillars

A caterpillar is a tree in which all the vertices are within distance 1 of a central path. We present in this section a polynomial time algorithm to optimally solve the problem where the agreement graph is a caterpillar.

4.1. Notations

In this section, we need the following notations:

- Let $G = (J, E)$ be an agreement graph along with a weighted function p over J, representing the processing times of J.
- The maximum weighted independent set (MWIS) of the agreement graph G is denoted $I_p(G)$.
- $\overline{I_p}(G)$ denotes the weight of a maximum independent set in G. ($\overline{I_p}(G)$ is clearly a lower bound on the optimal makespan)
- $N(j)$ is the set of neighbors of the job $j \in J$. This notation can be generalized for a subset $J' \subseteq J$ by $N(J')$.

- $Lv(j)$ is the set of leaves connected to job j in G.
- t_j is the starting time of a job $j \in J$, and $t(N(j))$ is defined as follows $t(N(j)) = min_{k \in N(j)}\{t_k\}$.

4.2. Caterpillar scheduling algorithm

Given a set of jobs J, and a caterpillar $CAT = (J, E)$, an optimal schedule σ of the SWA problem with agreement graph CAT on two identical machines is constructed as described in the *caterpillar scheduling algorithm* given below.

The algorithm requires a maximum weighted independent set computation (which is polynomially solvable for trees) as its core - this already tells us how to split the jobs between the two machines. Apart from this, it just suffices to define the starting times of the jobs and show that the result is optimal since the weight of a maximum weighted independent set in CAT is a lower bound on the makespan.

Algorithm 1 Caterpillar scheduling Algorithm

Input: $CAT = (J, E)$, processing times p_j, $\forall j \in J$;
Output: An optimal schedule σ;

1 Determine a MWIS S^* in the graph CAT (see paragraph 4.2.1);
2 Remove the edges connecting each pair of jobs of $J \setminus S^*$ (remove all edges that are not incident to S^*);
3 We obtain k connected caterpillars $CAT_i = (J_i, E_i)$ $(i = 1, \ldots, k$[c]$)$;
4 **For** $i = 1$ **to** k **do** {Construction of the schedule σ_i for the graph CAT_i}
  5 $S_i^* = J_i \cap S^*$;
  6 Arrange the jobs of J_i in a list L_i (see paragraph 4.2.2);
  7 According to the order induced by L_i, schedule the jobs of S_i^* successively on the first machine;
  8 Schedule the jobs of $J_i \setminus S_i^*$ in the second machine as follows:
  $t(\alpha_0$[d]$) = 0$;
  if α and β are in $J_i \setminus S_i^*$, and are consecutive in this order with respect to L_i **then** $t_\beta = max\{t_\alpha + p_\alpha, t(N(\beta))\}$;
endfor
9 Construct the optimal schedule σ by concatenating all the schedules σ_i, $i = 1, \ldots, k$;

4.2.1. Computation of S^*

To compute S^*, we use the dynamic program presented by Chen et al. [18] which computes a MWIS for the graph CAT in $O(n)$. This method has been designed to compute the MWIS for trees, which is still valid for the case of caterpillars.

[c] k is the number of connected graphs resulting from the second step.
[d] α_0 is the first job of $J_i \setminus S_i^*$ in the list L_i.

4.2.2. Arranging of jobs

To arrange the jobs, we take the jobs sequence of the central path (as ordered in the central path), which is naturally defined in caterpillar graphs, and add each job of this sequence to the list followed by its leaves (in any order).

4.3. Example

Consider a set $J = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o\}$ of 15 jobs and a caterpillar $CAT = (J, E)$. The processing times of the jobs are given in Table 2 and the agreement graph is described in Figure 6. We will solve the SWA problem using the *caterpillar scheduling algorithm*:

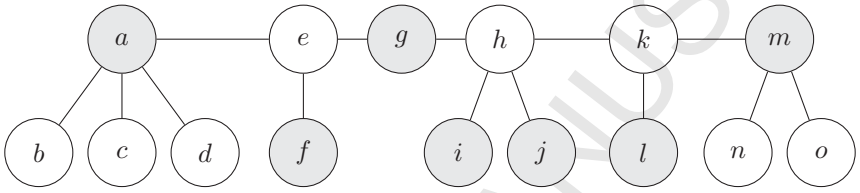


Figure 6: Agreement graph CAT (nodes filled with gray represent jobs of S^*).

Table 2: Processing times of the jobs.

Jobs	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
Processing times	5	1	2	1	3	4	2	4	3	2	1	5	4	2	1

1. $S^* = \{a, f, g, i, j, l, m\}$.
2. The only edge connecting the jobs of $J \setminus S^*$ is hk which we remove.
3. The two resulting caterpillars are $CAT_1 = (J_1, E_1)$ and $CAT_2 = (J_2, E_2)$.
4. The construction of the schedule σ_1 corresponding to the agreement graph CAT_1:
   (a) $L_1 = (a, b, c, d, e, f, g, h, i, j)$ (the jobs sequence of the considered central path is (a, e, g, h)).
   (b) $S_1^* = \{a, f, g, i, j\}$.
   (c) Jobs a, f, g, i and j are successively scheduled in this order on the first machine at respective starting times $t_a = 0$, $t_f = 5$, $t_g = 9$, $t_i = 11$, and $t_j = 14$.
   (d) Jobs b, c, d, e and h are scheduled in this order on the second machine at respective starting times $t_b = 0$, $t_c = 1$, $t_d = 3$, $t_e = 4$ and $t_h = 9$.
   (e) The obtained schedule σ_1 is shown in Figure 7 with $C_{max}(\sigma_1) = 16$.
5. The construction of the schedule σ_2 corresponding to the agreement graph CAT_2:
   (a) $L_2 = (k, l, m, n, o)$ (the jobs sequence of the considered central path is (k, m)).
   (b) $S_2^* = \{l, m\}$.

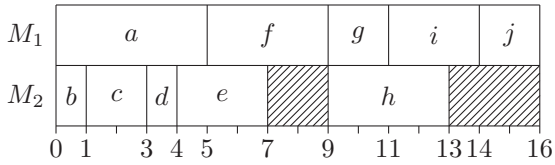


Figure 7: Schedule σ_1.

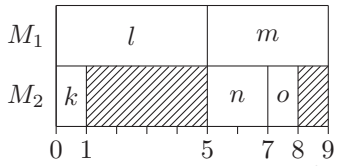


Figure 8: Schedule σ_2.

(c) Jobs l and m are successively scheduled in this order on the first machine at respective starting times $t_l = 0$ and $t_m = 5$.
(d) Jobs k, n and o are scheduled in this order on the second machine at respective starting times $t_k = 0$, $t_n = 5$ and $t_o = 7$.
(e) The obtained schedule σ_2 is shown in Figure 8 with $C_{max}(\sigma_2) = 9$.

6. Concatenate the schedules σ_1 and σ_2 to obtain the optimal schedule σ shown in Figure 9 with $C_{max}(\sigma) = 25$.

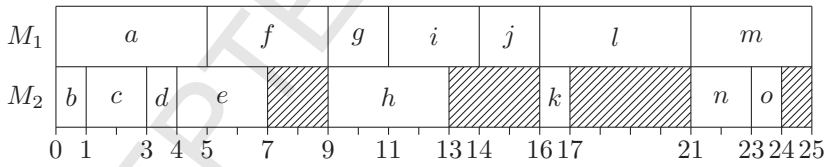


Figure 9: Optimal schedule σ.

4.4. Polynomiality and optimality proofs of the algorithm

In order to prove that the *caterpillar scheduling algorithm* computes an optimal schedule in polynomial time, we present the following results. For a given agreement graph CAT_i and schedules σ_i $(i = 1, \ldots, k)$, we have:

Claim 1. *S_i^* is a MWIS for the agreement graph CAT_i.*

Proof. Suppose that there exists a MWIS $I_p(CAT_i)$ such that $\overline{I_p}(CAT_i) > \sum_{j \in S_i^*} p_j$, we also consider a set $S' = (S^* \setminus S_i^*) \cup I_p(CAT_i)$. The sets $S^* \setminus S_i^*$ and $I_p(CAT_i)$ do not have any connection in CAT. In fact, $(S^* \setminus S_i^*) \cap J_i = \emptyset$,

so the only edges that could connect $S^* \setminus S_i^*$ and $I_p(CAT_i)$ are the removed edges from the second step of the *caterpillar scheduling algorithm*. Notice that these edges are not incident to the jobs of S^*. So, S' is an independent set in CAT. S' also has a weight greater than S^*, in fact, since $\overline{I_p}(CAT_i) > \sum_{j \in S_i^*} p_j$ we have:

$$\sum_{j \in S'} p_j = \overline{I_p}(CAT) - \sum_{j \in S_i^*} p_j + \overline{I_p}(CAT_i) > \overline{I_p}(CAT)$$

The weight of the independent set S' is greater than the weight of MWIS, which is a contradiction. □

Claim 2. $\forall W \subseteq J_i \setminus S_i^*, \sum_{j \in W} p_j \leq \sum_{j \in N(W)} p_j$.

Proof. Suppose that there exists a subset $W \subseteq J_i \setminus S_i^*$ such that $\sum_{j \in W} p_j > \sum_{j \in N(W)} p_j$. We also consider a set $S' = (S_i^* \setminus N(W)) \cup W$. We have $N(W) \subseteq S_i^*$ (see step 2 of the *caterpillar scheduling algorithm*). So, S' is an independent set in CAT_i and the weight of this independent set is also greater than the weight of S_i^*. In fact since $\sum_{j \in W} p_j > \sum_{j \in N(W)} p_j$ we have:

$$\sum_{j \in S'} p_j = \sum_{j \in S_i^*} p_j - \sum_{j \in N(W)} p_j + \sum_{j \in W} p_j > \sum_{j \in S_i^*} p_j$$

and this contradicts Claim 1. □

Claim 3. *$\forall \beta \in J_i \setminus S_i^*$, the neighbors of β are scheduled successively on the first machine in the time interval $I_\beta = [t(N(\beta)), t(N(\beta)) + \sum_{j \in N(\beta)} p_j]$.*

Proof. A job $\beta \in J_i \setminus S_i^*$ either be a leaf or not. If β is a leaf: it has a unique neighbor α, so α is scheduled in the time interval $[t_\alpha, t_\alpha + p_\alpha] = I_\beta$.

If β is a non-leaf job, then β could have multiple leaves and a maximum of two non-leaf neighbors α and γ. The jobs enter the list L_i in this order: job α (we could take γ first but without loss of generality we take α) and its leaves then job β and its leaves, and finally the job γ as well as its leaves, as shown in Figure 10. All the neighbors of β belong to S_i^*, therefore, these jobs are scheduled successively on the first machine in the order induced by L_i. Thus, the jobs of $N(\beta)$ are all scheduled successively in the time interval $[t(\alpha), t(\alpha) + p_\alpha + \sum_{j \in Lv(\beta)} p_j + p_\gamma] = I_\beta$. □

Claim 4. *$\forall \alpha, \beta \in J_i \setminus S_i^*$ if α and β are scheduled successively on the second machine, then α and β have a common neighbor.*

Proof. Without loss of generality, we suppose that α and β are scheduled in this order. All the possible cases for α and β are given as follows:

1. α is a non-leaf job: let γ be a non-leaf neighbor of α.
   a. β is the first scheduled leaf of $Lv(\gamma)$ (see Figure 11a).

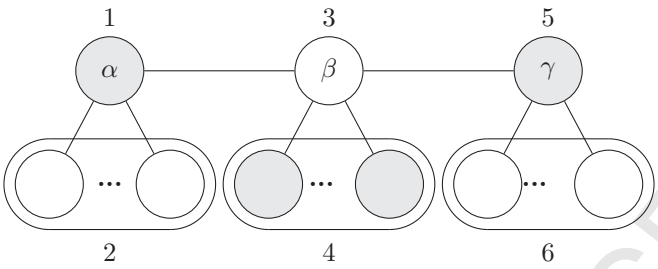


Figure 10: An illustration of the neighbors of β and their leaves as well as the order in which the jobs enter the list.

   b. If $L(\gamma) = \emptyset$, then job β is another non-leaf neighbor of γ (other than α) (see Figure 11b).

2. α is a leaf: let γ be a non-leaf neighbor of α.
   a. β is a leaf of γ that is scheduled right after α (see Figure 11c);
   b. If α is the last job scheduled among the jobs of $Lv(\gamma)$, then β is the next non-leaf neighbor of γ (see Figure 11d).

For all the possible cases, α and β have γ as a common neighbor. □

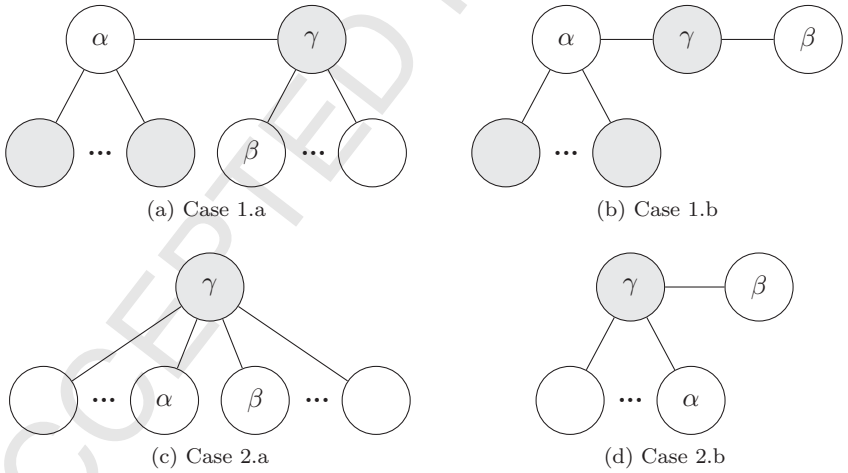


Figure 11: All the possible cases for two successive jobs α and β on the second machine (the nodes filled with gray represent the jobs of S_i^*).

Claim 5. *If $W = \{\alpha, \ldots, \beta\}$ is a set of consecutive jobs of $J_i \setminus S_i^*$ then the jobs of $N(W)$ are scheduled on the first machine in the time interval $I_W = [t(N(\alpha)), t(N(\beta)) + \sum_{j \in N(\beta)} p_j]$.*

Proof. Consequence of Claim 3 and Claim 4. □

Lemma 1. *Each job $\beta \in J_i \setminus S_i^*$ is scheduled in the time interval $I_\beta = [t(N(\beta)), t(N(\beta)) + \sum_{j \in N(\beta)} p_j]$.*

PROOF. We suppose that some job $\beta \in J_i \setminus S_i^*$ is not scheduled in the time interval I_β. β is not scheduled in this interval implies that: either $t_\beta < t(N(\beta))$ or $t_\beta + p_\beta > t(N(\beta)) + \sum_{j \in N(\beta)} p_j$. Let β_0 be the job, right before β on the second machine.

1. $t_\beta < t(N(\beta))$: job β has a starting time $t_\beta = max\{t_{\beta_0} + p_{\beta_0}, t(N(\beta))\}$, so, we cannot have $t_\beta < t(N(\beta))$.
2. $t_\beta + p_\beta > t(N(\beta)) + \sum_{j \in N(\beta)} p_j$: since $t_\beta = max\{t_{\beta_0} + p_{\beta_0}, t(N(\beta))\}$ we distinguish two cases:
   (a) $t_\beta = t_{\beta_0} + p_{\beta_0}$: let α be the last job scheduled on the second machine such that α precedes β and $t_\alpha = t(N(\alpha))$. We also consider $W = \{\alpha, \ldots, \beta\}$ the set of consecutive jobs scheduled on the second machine from α to β. There is no idle times between the jobs of W, so the jobs of W are scheduled in the time intervals $[t_\alpha, t_\beta + p_\beta]$. The jobs of $N(W)$ are scheduled in the time interval $[t(N(\alpha), t(N(\beta)) + \sum_{j \in N(\beta)} p_j]$ (Claim 5) as shown in Figure 12. Since $t_\alpha = t(N(\alpha))$ and $t_\beta + p_\beta > t(N(\beta)) + \sum_{j \in N(\beta)} p_j$, we have $\sum_{j \in W} p_j > \sum_{j \in N(W)} p_j$ which is a contradiction according to Claim 2.
   (b) $t(\beta) = t(N(\beta))$: this case is reported to the case 2.a where $\alpha = \beta$ and $W = \{\beta\}$.

□

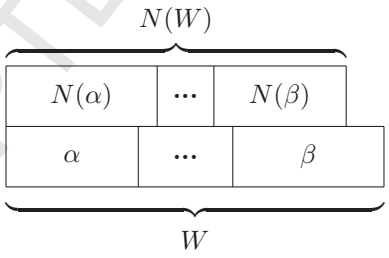


Figure 12: Scheduling of the jobs of both W and $N(W)$.

Theorem 2. *The caterpillar scheduling algorithm computes an optimal schedule σ of the SWA problem on two identical machines with the caterpillar CAT in $O(n)$ time, with $C_{max}(\sigma) = \overline{I_p}(CAT)$.*

PROOF. The determination of a MWIS for caterpillars (step1) is done in $O(n)$ (see [18]) and the determination of the connected components of a caterpillar graph (step 3) is done in $O(n)$ (see Hopcroft and Tarjan [19]). Then, the complexity of the *caterpillar scheduling algorithm* is trivially $O(n)$.

For $i = 1, \ldots, k$, the schedule σ_i is feasible and $C_{max}(\sigma_i) \leq \overline{I_p}(CAT_i)$ (direct consequence of Lemma 1). Since the schedule σ is the concatenation of the schedules σ_i, $i = 1, \ldots, k$, we have:

$$C_{max}(\sigma) = \sum_{i=1}^{k} \sigma_i \leq \sum_{i=1}^{k} \sum_{j \in S_i^*} p_j \leq \sum_{j \in S^*} p_j = \overline{I_p}(CAT)$$

$\overline{I_p}(CAT)$ is a lower bound on the optimal makespan. Therefore, the *caterpillar scheduling algorithm* computes an optimal schedule σ of the SWA problem on two machines with the caterpillar CAT as an agreement graph in $O(n)$ such that $C_{max}(\sigma) = \overline{I_p}(CAT)$. □

Corollary 1. *The SWA problem on two identical machines where the agreement graph is either a path or a star graph can optimally be solved in linear time using the caterpillar scheduling algorithm.*

Proof. It is a consequence of Theorem 2. In fact, star graphs and paths are subclasses of caterpillars. □

5. Cycles

Given a set of jobs J and a cycle $C = (J, E)$, an optimal schedule of the SWA problem with the agreement graph C on two identical machines is constructed as described in the *cycle scheduling algorithm* given below.

Algorithm 2 Cycle scheduling algorithm

Input: $C = (J, E)$, processing times p_j, $\forall j \in J$;
Output: An optimal schedule σ;
Begin

1 Foreach $e \in E$ **do**
  2 Compute an optimal schedule of the SWA problem for the path $P = (J, E - e)$ using the *scheduling caterpillar algorithm*;
enddo
3 σ is the schedule with the best makespan;
end.

Claim 6. *In the schedule σ, there exists at least one job $j \in J$ such that j is not scheduled opposite to its two neighbors.*

Proof. Suppose that each job $j \in J$ is scheduled opposite to its two neighbors. This implies that the jobs are scheduled in a circular manner which is impossible. □

Corollary 2. *The cycle scheduling algorithm computes an optimal schedule of the SWA problem with a cycle agreement graph on two identical machines in* $O(n^2)$.

Proof. To build a feasible schedule, we cannot use all the agreements represented by a cycle (according to Claim 6). So, solving the SWA problem when the agreement graph is a cycle is equivalent to solve the problem by removing the "unused" edge from the cycle. Removing all the edges one at a time[e] and solve the SWA problem for each of the resulting paths, provides an optimal schedule σ. The *cycle scheduling algorithm* uses n times the *caterpillar scheduling algorithm*, so, the complexity of the *cycle scheduling algorithm* is $O(n^2)$.

6. Conclusion

In the present paper, we have considered the problem of scheduling with agreements. We have proved the NP-hardness of the problem for the case of trees. Polynomial time algorithms have been proposed to optimally solve the problem when the agreement graph is either a caterpillar or a cycle.

For future work, it would be interesting to design and implement exact and/or approximation methods to solve the NP-hardness cases of the problem including the one presented in this paper.

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

[2] M. R. Garey, R. L. Graham, Bounds for multiprocessor scheduling with resource constraints, SIAM Journal on Computing 4 (1975) 187–200.

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

[4] B. S. Baker, E. G. Coffman, Mutual exclusion scheduling, Theoretical Computer Science 162 (1996) 225–243.

[5] M. M. Halldórsson, G. Kortsarz, A. Proskurowski, R. Salman, H. Shachnai, J. A. Telle, Multi-coloring trees, in: Proceedings of the Fifth International Computing and Combinatorics Conference (COCOON), Tokyo, Japan, Lecture Notes in Computer Science, vol. 1627, Springer, Berlin, 1999, pp. 271–280.

[6] M. R. Garey, D. S. Johnson, Computers and intractability: A Guide to the Theory of NP-Completeness, New York: Freeman, 1979.

[7] M. R. Garey, D. S. Johnson, Complexity results for multiprocessor scheduling under resource constraints, SIAM Journal on Computing 4 (1975) 397–411.

[e] Enumerate all the possible cases.

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

[9] J. Blazewicz, J. K. Lenstra, A. R. Kan, Scheduling subject to resource constraints: classification and complexity, Discrete Applied Mathematics 5 (1983) 11–24.

[10] M. Bendraouche, M. Boudhar, Scheduling with agreements: new results, International Journal of Production Research 54 (2016) 3508–3522.

[11] F. Gardi, Mutual exclusion scheduling with interval graphs or related classes, part i, Discrete Applied Mathematics 157 (2009) 19–35.

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

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

[14] D. De Werra, Restricted coloring models for timetabling, Discrete Mathematics 165 (1997) 161–170.

[15] N. E. H. Tellache, M. Boudhar, Two-machine flow shop problem with unit-time operations and conflict graph, International Journal of Production Research 55 (2017) 1664–1679.

[16] N. E. H. Tellache, M. Boudhar, Open shop scheduling problems with conflict graphs, Discrete Applied Mathematics 227 (2017) 103–120.

[17] N. E. H. Tellache, M. Boudhar, Flow shop scheduling problem with conflict graphs, Annals of Operations Research 261 (2018) 339–363.

[18] G. Chen, M. Kuo, J. Sheu, An optimal time algorithm for finding a maximum weight independent set in a tree, BIT Numerical Mathematics 28 (1988) 353–356.

[19] J. Hopcroft, R. Tarjan, Algorithm 447: efficient algorithms for graph manipulation, Communications of the ACM 16 (1973) 372–378.