Scheduling problem subject to compatibility constraints

Mohamed Bendraouche[1] and Mourad Boudhar[2]

[1] Faculty of Sciences, Saad Dahleb University,
Route de Soumaa, BP 270, Blida, Algeria
mbendraouche@yahoo.fr
[2] Faculty of Mathematics, USTHB University,
BP 32 El-Alia, Bab-Ezzouar, Algiers, Algeria
mboudhar@yahoo.fr

Abstract. The problem studied in this paper consists in finding the minimum makespan in a problem of scheduling jobs on identical parallel processors subject to compatibility constraints that some jobs cannot be scheduled simultaneously in any time interval. These constraints are modeled by a graph in which compatible jobs are represented by adjacent vertices. We study the complexity of this problem for bipartite graphs and their complements. We propose polynomial heuristics which are experimentally evaluated and compared.

Keywords. scheduling, compatibility graph, complexity, heuristics.

1 Introduction

In this work we study the problem of scheduling n independent jobs $J_1, J_2, ..., J_n$ non-preemptively on m identical parallel processors. Each job J_i has a processing time p_i and a release time r_i. Two jobs are compatible if they can be scheduled simultaneously. We suppose that there exist compatibility constraints between the jobs such that non-compatible jobs cannot be scheduled simultaneously in any time interval. These constraints are represented by a graph $G = (V, E)$ where V is the jobs set and $\{J_i, J_j\} \in E$ if and only if J_i and J_j are compatible. The graph G is called the compatibility graph. The aim is to minimize the makespan.subject to the compatibility constraints. Following the three parameters notation introduced in [1] our problem is denoted by $P/G = (V, E), r_i/C_{\max}$ and is sometimes referred to as the general problem. When m is fixed the problem is denoted by $Pm/G = (V, E), r_i/C_{\max}$.
This problem arises in the resource-constrained scheduling when the resources are non-sharable. Applications of this problem include the one of Baker and Coffman [2] presented for balancing the load in a parallel computation, others are mentioned in traffic intersection control, frequency assignment in cellular networks and session management in local area networks (see Halldorsson et al.[3]). Bodlaender and Jansen [4] have described an application derived from a problem of assigning operations to processors, where the operations are given

in a flow graph. Our interest to this problem initially comes from the following problem. **The Workshop Resource-Constrained** problem (W.R.C in short): there are n tasks $J_1, J_2, .., J_n$ to be executed by m workers in a workshop. Each task J_i requires a time p_i for its treatment and a subset of resources $R_i \subseteq R$ where R is the set of the available resources. The objective is to execute these tasks in a minimum time. If we regard the workers as processors and associate the graph $G = (V, E)$ in which the tasks correspond to the jobs set and that two jobs are compatible if they use no resources in common, one can verify that the W.R.C problem can be modeled as the problem $P/G = (V, E)/C_{\max}$.

Example 1. $n = 5, m = 2$, $R = \{res_1, res_2\}$. The resource requirements and the processing times are given in Table 1, the compatibility graph is represented on Fig. 1 (a). Fig. 1 (b) is the Gantt diagram representing a feasible schedule for this example, which is also optimal.

Table 1. Processing times and the resources requirements

J_i	J_1	J_2	J_3	J_4	J_5
p_i	1	2	1	3	1
R_i	$\{res_1, res_2\}$	$\{res_1\}$	$\{res_2\}$	$\{res_1\}$	$\emptyset$

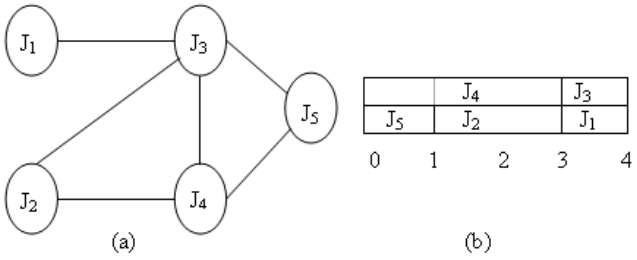

Fig. 1. (a)The compatibility graph of Example1 (b) The Gantt diagram of Example1

2 Related work and previous results

Among the related problems are:
1-The Mutual Exclusion Scheduling problem, M.E.S in short [2]: n unit-processing times jobs have to be scheduled on m processors in a minimum time subject to constraints represented by a conflict graph G such that adjacent jobs in G must be scheduled in disjoint time intervals.

2-Scheduling With Conflicts problem, S.W.C in short: Recently in [5] Guy Even, Magnus M. Halldorsson, Lotem Kaplan and Dana Ron have considered the problem of Scheduling With Conflicts (S.W.C) which consists in finding a minimum makespan on identical machines where conflicting jobs cannot be scheduled concurrently. The relationship between these problems and ours is as follows: The S.W.C problem with an arbitrary conflict graph $G = (V, E)$ is equivalent to the problem $Pm/\overline{G} = (V, \overline{E})/C_{\max}$ for which the compatibility graph is $\overline{G}$, the complement of G.
The M.E.S problem with fixed m is a special case of the S.W.C problem in which the processing times of the jobs are equal to 1. The M.E.S problem with conflict graph $G = (V, E)$ is equivalent to the problem $P/\overline{G} = (V, \overline{E}), p_i = 1/C_{\max}$ where the compatibility graph is $\overline{G}$, the complement of G. As far as the previous results are concerned, the problem $P2/G = (V, E), p_i = 1/C_{\max}$ can be reduced the maximum matching problem in G. The authors of [5] have established that the S.W.C problem can polynomially be solved when $m = 2$ and $p_i \in \{1, 2\}$, we deduce that the problem $P2/G = (V, E), p_i \in \{1, 2\}/C_{\max}$ is polynomial. By a reduction from Partition ([6]) the problem $P2/G = (V, E)/C_{\max}$ is NP-hard even if the compatibility graph is complete. The problem $Pm/G = (V, E), p_i = 1/C_{\max}$ is NP-hard for any $m \geq 3$ due to a result of B.S. Baker and E.G. Coffman [2]. The problem $Pm/G = (V, E), p_i \in \{1, 2, 3, 4\}/C_{\max}$ is APX-hard [5].

3 Bipartite graphs

In this section, we suppose that $m = 2$ and that the compatibility graph G is bipartite which is denoted by $G = (S_1 \cup S_2, E)$.

3.1 Case when $p_i \in \{1, 2\}, r_i \in \{0, r\}$

Theorem 1. *The problem $P2/G = (S_1 \cup S_2, E), p_i \in \{1, 2\}, r_i \in \{0, r\}/C_{\max}$ is NP-hard.*

Proof. Let Dbipartite be the decision problem associated with the above problem. We make a reduction from the 3- Dimentional Matching problem(3-DM). Let an arbitrary instance of 3-DM be given. We construct an instance of Dbipartite as follows: the jobs set is $V = V_M \cup V_Y \cup V_Z \cup V_D$ such that the jobs of V_M are in correspondence with the elements of M. Thus to any triplet $(x, y, z) \in M$ corresponds a job $J(x, y, z)$ in V_M. The elements of V_Y and V_Z are in correspondence with the elements of the sets Y and Z respectively. Thus each element $y \in Y$ corresponds to a job J_y of V_Y and each element $z \in Z$ corresponds to a job J_z of V_Z. Note that $|V_M| = |M|$, $|V_Y| = |V_Y| = q$. Suppose that $X = \{x_1, x_2, ..., x_q\}$. For each i ($i = 1, 2, ..., q$) let $M_i = \{(x, y, z) \in M : x = x_i\}$ so that $\bigcup_{i=1}^{q} M_i = M$ and let V_{M_i} be the subset of V_M corresponding to M_i. Note that $|V_{M_i}| = |M_i|$. To each job set V_{M_i} corresponds a set V_{D_i} of dummy jobs such that $|V_{D_i}| = |M_i| - 1$

$(i = 1, 2, ..., q)$. Let $V_D = \bigcup_{i=1}^{q} V_{D_i}$. One can see that $|V_D| = |M| - q$. The compatibility graph G is defined in such a way that each of the sets V_M, V_Y, V_Z, V_D form an independent set of vertices in G.

Every job $J(x, y, z)$ of V_M is compatible with both of the jobs J_y and J_z belonging to V_Y and V_Z respectively. For each $i = 1, 2, ..., q$, the jobs of V_{M_i} are joined to all of the jobs of V_{D_i}. Let G_M, G_Y, G_Z and G_D denote the subgraphs of G induced by V_M, V_Y, V_Z and V_D respectively. The processing times of the jobs of G_M and G_D are equal to 2 and those of G_Y and G_Z are equal to 1. The release times of the jobs of G are zero except those of G_D which are equal to $2q$, see $Fig.$ 2.

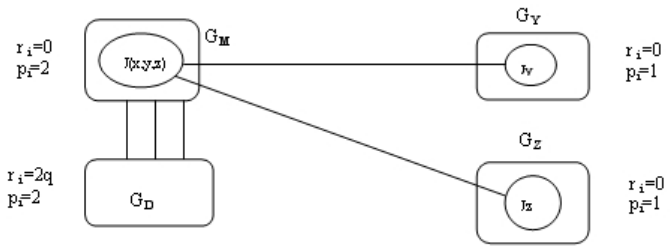

Fig. 2. The compatibility graph $G = (V, E)$

The graph G is bipartite since it can be written $G = (S_1 \cup S_2, E)$ where $S_1 = V_M$ and $S_2 = V_Y \cup V_Z \cup V_D$ and that §1 and S_2 are independent sets of vertices in G. Suppose that 3-DM has a solution. Consider the schedule σ defined as follows: let $\{J_1, J_2, .., J_q\}$ be the set of jobs of V_M corresponding to M' such that $J_i \in V_{M_i}$ $(i = 1, 2, ..., q)$. We schedule these jobs as well as their corresponding jobs of the graphs G_Y and G_Z in the time interval $[0, 2q]$ as indicated in Fig. 3.

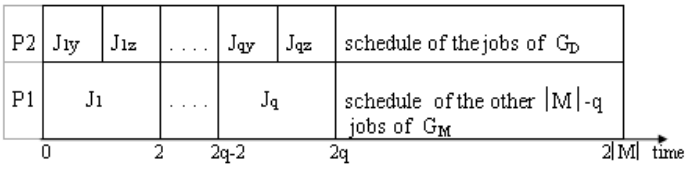

Fig. 3. Passage from a feasible solution of 3-DM to the one of Dbipartite

We schedule the remaining $|M| - q$ jobs of G_M in the interval $[2q, 2|M|]$ on the processor $P1$ as follows: we schedule these jobs into groups: the jobs of $V_{M_1} \backslash \{J_1\}$ first then the ones of $V_{M_2} \backslash \{J_2\}$ and so on till the jobs of $V_{M_q} - \{J_q\}$. We then schedule the jobs of G_D on $P2$, within the same interval in such a way that the jobs of $V_{M_i} - \{J_i\}$ are opposite to their corresponding jobs of V_{D_i} $(i = 1, 2, ..., q)$, see Fig. 3. The schedule that has just been obtained, say σ is therefore a feasible schedule for the problem Dbipartite with a makespan $C_{\max}(\sigma) \leq 2|M|$. Conversely, assume that there is a feasible schedule σ for the problem Dbipartite with a makespan $C_{\max}(\sigma) \leq 2|M|$. The jobs of G_D form an independent set of vertices of order $|M| - q$ in the compatibility graph G. Having each a processing time equal to 2 and a release time equal to $2q$, they must have been executed in the time interval $[2q, 2|M|]$. Without loss of generality we may suppose that they have been executed on $P2$. The remaining jobs include the ones of the graph G_M and those of the graphs G_Y and G_Z and their number is equal to $|M| + 2q$.

The remaining space equals $4|M| - (2|M| - 2q) = 2|M| + 2q$. Thus this space must entirely have been occupied by the $|M| + 2q$ remaining jobs of the graphs G_M, G_Y and G_Z. By construction of the graph G, q jobs of G_M must have been processed in the time interval $[0, 2q]$ along with their corresponding jobs of G_Y and G_Z respectively as indicated in Fig. 3. This corresponds to a set $M' \subseteq M$ with q elements, producing a solution of 3-DM. It is straightforward to see that the transformation used is polynomial and that DBipartite $\in$ NP, which completes the proof of Theorem 1. □

3.2 Unit processing times for S_1 and arbitrary for S_2

Next we consider the case when the processing times of the jobs of S_1 are equal to unity and the ones of S_2 are arbitrary. This problem is denoted by $P2/G = (S_1 \cup S_2, E)$, $p_{S_1} = 1/C_{\max}$. Suppose that $S_1 = \{T_1, T_2, ..., T_{|S_1|}\}$ and $S_2 = \{J_1, J_2, ..., J_{|S_2|}\}$.

Algorithm 1
Begin

1: Construct the network R=(V,U,c) as follows :
$V = \{s, p\} \cup S_1 \cup S_2$, $U = U_1 \cup U_2 \cup U_3 \cup \{u_r\}$ where $u_r = (p, s)$ is a return arc, $U_1 = \{(s, T_i), T_i \in S_1\}$, $U_2 = \{(T_i, J_j) \in S_1 \times S_2 : \{T_i, J_j\} \in E\}, U_3 = \{(J_j, p), J_j \in S_2\}$

2: Construct the arc capacity function c as follows:
if $(s, T_i) \in U_1 \Longrightarrow c(s, T_i) = 1$, if $(T_i, J_j) \in U_2 \Longrightarrow c(T_i, J_j) = 1$,
$c(u_r) = +\infty$, if $(J_j, p) \in U_3 \Longrightarrow c(J_j, p) = p_j$.

3: Determine a maximum feasible flow f^* in the network R.

4: Schedule the jobs of S_2 successively in the time interval $[0, \sum_{i=1}^{|S_2|} p_i]$ on $P1$.

5: Schedule the jobs T_i of S_1 such that $f^*(T_i, J_1) = 1$ opposite to the job J_1 in the time interval $[0, p_1]$, and the jobs T_i of S_1 such that $f^*(T_i, J_2) = 1$ opposite to the job J_2 in the time interval $[p_1, p_1 + p_2]$ and so on.

6: Schedule the remaining jobs of S_1 successively in the time interval $[\sum_{i=1}^{|S_2|} p_i, \sum_{i=1}^{|S_2|} p_i + |S_1| - f^*(u_r)\ ]$.

end

Remark 1. 1- If u is an arc of R, and f is a flow in R then $f(u)$ represents the value of the flow f on the arc u in R.
2- In step 3 of the above Algorithm we use J. Cherian and S.N. Maheshwari 'Algorithm [7] for obtaining a maximum flow.

Theorem 2. *Algorithm 1 solves the problem* $P2/G = (S_1 \cup S_2, E), p_{S_1} = 1/C_{\max}$ *polynomially in* $O(n^3)$.

Proof. One can easily see that Algorithm 1 returns a feasible schedule σ^* for the problem $P2/G = (S_1 \cup S_2, E), p_{S_1} = 1/C_{\max}$ with a makespan $C_{\max}(\sigma^*) = \sum_{i=1}^{|S_2|} p_i + |S_1| - f^*(u_r)$. Let us now show that for every feasible schedule σ we have: $C_{\max}(\sigma) \geq C_{\max}(\sigma^*)$. For let σ be an arbitrary feasible schedule. We shift the jobs of S_2 to the leftmost side of the Gantt diagram representing σ, as well as the jobs of S_1 that are scheduled opposite to the jobs of S_2 so that all these jobs would be scheduled in the time interval $[0, \sum_{i=1}^{|S_2|} p_i]$. By shifting and removing all the idle times in the Gantt diagram and by scheduling successively the remaining jobs of S_1 after the instant $\sum_{i=1}^{|S_2|} p_i$, we get a feasible schedule σ_1 and a feasible flow f_1 corresponding to σ_1 such that $C_{\max}(\sigma) \geq C_{\max}(\sigma_1) = \sum_{i=1}^{|S_2|} p_i + |S_1| - f_1(u_r)$. But since f^* is a maximum flow in R then $f_1(u_r) \leq f^*(u_r)$ and thus for every feasible schedule σ we have: $C_{\max}(\sigma) \geq \sum_{i=1}^{|S_2|} p_i + |S_1| - f^*(u_r)$. Since $\sum_{i=1}^{|S_2|} p_i + |S_1| - f^*(u_r) = C_{\max}(\sigma^*)$ (proved previously) we deduce that the schedule σ^* obtained by Algorithm 1 is optimal . The construction of the network R can be achieved in at most $O(|S_1|\,|S_2|) = O(n^2)$ time. By using J. Cherian and S.N. Maheshwari' Algorithm, step2 can be performed in $O(n^2\sqrt{|U|})$. As $|U| = O(|S_1|\,|S_2|) = O(n^2)$, step 2 can be achieved in $O(n^3)$. We deduce that the time complexity of Algorithm 1 equals $O(n^3)$. □

4 Complement of bipartite graphs

Now we consider the case when the compatibility graph G is the complement of a bipartite graph noted $G = (K_1, K_2; E)$ where K_1, K_2 are the cliques of G forming a partition of V.

H.L. Bodlaender, K. Jansen [4] have established that the problem $P/G = (K_1, K_2, E), p_i = 1/C_{\max}$ is NP-hard. We prove that the problem becomes polynomial if we restrict to odd r_i s. Consider the following greedy algorithm in which

for any job J_i, t_i represents its starting time and suppose $K_1 = \{J_1, J_2\ , .., J_s\ \}$ and $K_2 = \{J_{s+1}, J_{s+2}\ , .., J_n\}$.

Algorithm 2
Begin

1: **for** $i := 1$ to s **do**
2: $t_i := r_i$
3: **end for**
4: **for** $k := s + 1$ to n **do**
5: **if** (all the jobs of K_1 scheduled at time $t = r_k$ are compatible with J_k) **then**
6: $t_k := r_k$
7: **end if**
8: $t_k := r_k + 1$
9: **end for**

end

Theorem 3. *The problem $P/G = (K_1, K_2; E), p_i = 1, r_i$ odd $/C_{\max}$ with $m \geq n$ can polynomially be solved in $O(n^2)$ time.*

Proof. Let τ be the schedule obtained by Algorithm 2. First we prove that τ is an optimal feasible schedule. It is clear that τ is feasible. Let $L = max_{1 \leq i \leq n}\{t_i\}$, thus $C_{\max}(\tau) = L+1$. Two possibilities may happen: case1: L is even. Let J_k be a job satisfying $L = t_k$. Since t_k is even, 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 $t_i = r_k$ which is is not compatible 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 not compatible, with the condition $r_i = r_k$ it follows that $C^*_{\max} \geq r_k + 2 = t_k + 1 = L + 1 = C_{\max}(\tau)$.
case2: L is odd . Let J_p be a job satisfying $L = t_p$. t_p is odd , so J_p has not been scheduled at step 8 of the algorithm and hence $t_p = r_p$. This implies that $C^*_{\max} \geq r_p + 1 = L + 1 = C_{\max}(\tau)$. We deduce that the schedule τ is optimal.
Time complexity : the for-loop through steps 1-3 can be implemented in $O(n)$ time at worse, the for-loop through steps 4-9 requires at most $O(n^2)$ iterations. Then the time complexity equals $O(n^2)$. □

5 Heuristics for the problem $P/G = (V, E)/C_{\max}$

In this section, we propose some heuristics for the problem $P/G = (V, E)/C_{\max}$. The approach used is based on the list scheduling one used by R.L. Graham [8] for the problem $P//C_{\max}$ with some modifications.

5.1 Lower bounds on the optimal makespan

Let $Opt(G)$ denote the optimal makespan of the problem $P\,/G = (V, E)/\,C_{\max}$. A trivial lower bound on $Opt(G)$ is $LB_0 = Max\{\lceil(\sum_{j=1}^{n} p_j)/m\rceil, Max_{1 \leq j \leq n}\{p_j\}\}$.

On the other hand in [9] S. Sakai, M. Togasaki, K. Yamazaki have studied the Maximum Weighted Independent Set problem and have elaborated three greedy algorithms for this problem namely GWMIN, GWMAX and GWMIN2. Since non-adjacent jobs must be scheduled in disjoint time intervals for any feasible schedule, we deduce three lower bounds on $Opt(G)$ based on the three algorithms previously cited, say LB_1, LB_2 and LB_3 respectively. From the implementation we have observed that the second lower bound is weaker and therefore the overall lower bound on $Opt(G)$ is given by: $LB = Max\{LB_0, LB_1, LB_3\}$.

5.2 Definition of nine job lists

First we define the compatibility number (C.N for short) of a job to be the number of jobs which are compatible with it. Next we define nine job lists. If the jobs are sorted in order of increasing compatibility numbers, the list obtained is called $List1$. If the jobs are arranged in a decreasing order of their compatibility numbers, the list obtained is called $List2$. The following algorithm constructs $List3$.

Algorithm 3
Begin
1: Find a job J_1 from V with maximum C.N in G
2: **for** $j := 2$ to n **do**
3: $G' :=$ the subgraph of G induced by $V\backslash\{J_1, J_2, ..., J_{j-1}\}$.
4: Find a job J_j from $V\backslash\{J_1, ..., J_{j-1}\}$ with maximum C.N in G'.
5: **end for**
6: $List3 := (J_1, ..., J_n)$
end

If in the for-loop of Algorithm 3 we choose a job J_j with a minimum compatibility number in G' we obtain $List4$. $List5$ is a random permutation of $(1, 2, 3, ..., n)$. The next algorithm produces $List6$.

Algorithm 4
Begin
1: determine a job J_1 from V with maximum C.N
2: $j := 1$
3: **for** $j = 2$ to n **do**
4: **for** all $J_k \in V\backslash\{J_1, J_2, ..., J_{j-1}\}$ **do**
5: n_{J_k} :=the number of jobs in $V\backslash\{J_1, J_2, ..., J_{j-1}\}$ that are compatible with J_i
6: **end for**
7: determine among these jobs J_k a job J_j such that $n_{J_j} = max(n_{J_k})$
8: **end for**
9: $List6 := (J_1, ..., J_n)$
end

$List7$ is obtained by a similar algorithm to Algorithm 4 except that at step 7 we choose J_j satisfying $n_{J_j} = \min(n_{J_k})$. In $List8$ the jobs are sorted according

to the Shortest Processing Time rule called the SPT list. Finally $List9$ is based on the Longest Processing Time rule and is called the LPT list.

5.3 The proposed heuristics

Suppose (H) is a heuristic and that some jobs have already been scheduled by (H). Let $sch(H)$ denote the subset of jobs already scheduled by (H). If J_j is a job to be scheduled by (H) at some instant t let $\Psi(t, J_j)$ denote the set of jobs of $sch(H)$ having a part of processing in the time interval $[t, t+p_j]$. We say that the job J_j is ready for processing at time t if all the jobs of $\Psi(t, J_j)$ are compatible with J_j, otherwise J_j is not ready for processing at time t.
Next we present the Heuristic scheme that generates all the proposed heuristics. For any job J_j let p_j and c_j will represent the processing and the completion time of the job J_j respectively. For any processor Pi, s_i will denote the earliest time at which Pi becomes free.

Algorithm $HeuristicScheme$
Begin

choose a list L, say $L = (J_1, J_2, ..., J_n)$; $t := 0$
for all $i = \overline{1, m}$ **do**
  $s_i := 0$
end for
schedule job J_1 on processor $P1$ at time t
for $j = 2$ to n **do**
  while (J_j is not scheduled) **do**
    $schedule := false$
    find the first free processor Pk and its s_k; $t := s_k$
    for $r = j$ to n **do**
      if (J_r is unscheduled and ready at time t) **then**
        schedule J_r on Pk at t
        schedule $schedule := true$
        break
      end if
    end for
    if ($schedule := false$) **then**
      determine the set $\Psi(t, J_j)$
      $t := max\{c_i : J_i \in \Psi(t, J_j)$ and J_i is not compatible with $J_j\}$
    end if
  end while
end for

end

Each choice of L at step1 in this scheme induces a heuristic for the problem $P/G = (V, E)/C_{\max}$. Thus we derive nine heuristics that are called $H1, H2, ..., H9$ respectively.

Remark 2. 1- The break instruction in the above scheme makes the program exit from the for-loop in which it is contained.
2- The heuristic $H9$ corresponds to the LPT-list and is also referred to as the LPT-heuristic

6 Experimental results

All the heuristics have been coded with Matlab 7.0 and tested on a pentium IV PC computer with a 3.4 GHZ and 2 GB Ram. We have used two classes of randomly generated instances: instances with variable processing times and instances with unit processing times. Besides the parameters m and n we have used the parameter d which is the density (in percentage) of the compatibility graph. For both classes the different combinations of the parameters m, n and d are as follows: $m \in \{2, 5, 10, 20\}$, the values $m = 2$ and $m = 5$ are each associated with seven values of n such that $n \in \{10, 20, 50, 100, 250, 500, 1000\}$, the value $m = 10$ corresponds to seven values of n where $n \in \{10, 20, 50, 100, 250, 500, 1000\}$ and the last value $m = 20$ is associated with seven values of n such that $n \in \{30, 50, 100, 250, 350\ , 500, 1000\}$. The different values used for d are $10, 20, ..., 90$. For both classes the instances are generated as follows: for each triplet (m, n, d) for which n is different from 1000 we have generated 100 randomly generated instances according to the uniform distribution in which 25 instances are generated with $p_i \in \{1, 2, ..., 10\}$, 25 instances with $p_i \in \{1, 2, ..., 20\}$, 25 instances with $p_i \in \{50, 51, ..., 100\}$ and 25 instances with $p_i \in \{1, 2, ..., 100\}$. For $n = 1000$, we have done the same except that we have used 40 instances (rather than 100 instances) involving 4 sets of 10 instances each rather than of 25 instances as previously done. The generated instances have been grouped into three groups : low density, medium density and high density and these correspond to instances whose compatibility graph densities belong to $\{10, 20, 30\}, \{40, 50, 60\}$and $\{70, 80, 90\}$ respectively. In total we have used 22940 instances for each class.

After extensive experiments the first observation is that the three heuristics $H1, H4$ and $H9$ (that is the LPT-heuristic) are considerably better than the others in the case of variable processing times and that the heuristics $H1$ and $H4$ are the best in the unit processing times case. Therefore we only have compared these three heuristics in the case of variable processing times and the heuristics $H1$ and $H4$ in the unit processing times case. We have obtained four tables corresponding to $m \in \{2, 5, 10, 20\}$. The results for the case $m = 5$, variable processing times are represented in table 3. In each table the first row represents the number of times (in percentage) for which the corresponding heuristic is best. The MD and the AD rows represent the maximum and the average deviations from the lower bound respectively. The last row AT represents the average CPU time in seconds for each heuristic. The main observations are: in the case of variable processing times the LPT-heuristic is in general considerably better than both heuristics $H1$ and $H4$. In the unit processing times case is concerned we have observed that $H4$ is best relatively to $H1$. The deviations of the LPT-

heuristic (v.p.t case) and those of the heuristic $H4$ (u.p.t case) are summarized in table 2 lines 1-3, lines 4-8 respectively .
Analysis of the results obtained: In the case of variable processing times the LPT-heuristic is expected to lead to better solutions compared to the others since it takes into account the processing times of the jobs and in the same checks the compatibility, in contrast to the others which use only the compatibility. This heuristic is somewhat a compromise between the processing times and the compatibility. However, in the unit processing times case the compatibility has an effect since the processing times are equal. The superiority of the heuristics H1 and H4 can be explained by the fact that both of them give the priority to the jobs with less compatibility numbers to pass first since it is more likely that the schedule of the jobs with higher compatibility numbers keeps the maximum completion time reduced.

Table 2. Deviations of the LPT-heuristic(v.p.t) and heuristic $H4$ (u.p.t)

Low den.	$n \leq 100, m \leq 20$ Av-dev ≤ 0.664, Max-dev $= 1.387$
Med. den.	$n \leq 1000, m \leq 10$ Av-dev ≤ 0.972, Max-dev $= 1.634$
High den.	$n \leq 1000, m \leq 20$ Av-dev ≤ 0.987, Max-dev $= 1.980$
Low den.	$n \leq 1000, m \leq 10$ Av-dev ≤ 1.368, Max-dev $= 1.964$
Med. and High.den.	$n \leq 1000, m \leq 10$ Av-dev ≤ 0.789, Max-dev $= 1.4$
Low.den.	$m = 20, n \leq 100$ Av-dev ≤ 0.684, Max-dev $= 1.308$
Med.den.	$m = 20, n \leq 100$ Av-dev ≤ 1.029, Max-dev $= 1.5$
High.den.	$m = 20, n \leq 1000$ Av-dev ≤ 0.702, Max-dev $= 1.6$

7 Conclusion

In this paper we have studied the problem of scheduling jobs non-preemptively on identical parallel processors and the aim is to minimize the makespan subject to the compatibility constraints. We have studied the complexity of the problem for bipartite graphs and their complements. In addition we have devised several polynomial time heuristics with acceptable performances for the problem without release times. The effectiveness of such heuristics have been evaluated by extensive experiments on randomly instances, showing that the LPT-heuristic outperforms all the proposed ones when the processing times are variable and that $H4$ is the best in the case of unit processing times.

References

1. Graham R.L., Lawler E.L., Lenstra J.K. and Rinnooy Kan A.H.G. Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann Discrete Math 5: 287–326 (1979).

Table 3. Experimental results m=5, v.p.t case

	Low dens.			**Med. dens.**			**High dens.**		
n	**H1**	**H4**	**H9**	**H1**	**H4**	**H9**	**H1**	**H4**	**H9**
n=10 Best	42.333	73.333	76.333	41.000	62.000	59.333	78.333	77..000	69.333
MD	0.657	0.543	0.355	0.625	0.667	0.696	0.500	0.500	0.684
AD	0.105	0.052	0.048	0.150	0.110	0.118	0.078	0.080	0.115
AT	0.008	0.007	0.009	0.008	0.008	0.008	0.005	0.004	0.005
n=20 Best	14.000	38.667	64.333	26.667	40.667	46.333	46.667	40.333	33.667
MD	0.614	0.471	0.568	1.089	0712	1.100	0.875	0.600	0.716
AD	0.200	0.144	0.116	0.308	0.272	0.266	0.167	0.181	0.211
AT	0.020	0.018	0.022	0.014	0.020	0.015	0.009	0.013	0.013
n=50 Best	4.333	28.000	68.000	29.667	38.667	37.333	37.000	35.000	39.000
MD	1.169	1.003	0.991	1.000	0.868	0.825	0.286	0.337	0.349
AD	0.488	0.420	0.372	0.473	0.450	0.458	0.074	0.075	0.091
AT	0.047	0.047	0.041	0.035	0.035	0.034	0.019	0.023	0.025
n=100 Best	3.333	18.667	78.667	34.000	40.333	29.333	24.333	33.000	51.333
MD	0.719	0.708	0.668	0.109	0.094	0.131	0.109	0.094	0.029
AD	0757	0.697	0.628	0.291	0.282	0.299	0.032	0.028	0.029
AT	0.166	0.174	0.154	0.115	0.110	0.090	0.029	0.023	0.027
n=250 Best	4.333	13.000	82.667	42.000	54.667	12.000	14.333	13.333	75.000
MD	1.428	1.384	1.324	0.321	0.287	0.305	0.044	0.027	0.054
AD	0.838	0.796	0.723	0.088	0.083	0.113	0.012	0.011	0.006
AT	1.146	1.216	1.059	0.538	0.517	0.416	0.066	0.054	0.061
n=500 Best	6.000	15.667	79.000	37.667	51.333	18.333	12.000	9.000	81.000
MD	1.275	1.196	1.070	0.132	0.122	0.159	0.020	0.013	0.021
AD	0.682	0.659	0.600	0.025	0.023	0.046	0.006	0.006	0.002
AT	5.696	6.042	5.105	1.673	1.587	1.350	0.167	0.135	0.151
n=1000 Best	13.333	26.667	60.833	38.333	35.833	37.500	5.000	7.500	88.333
MD	0.974	0.961	0.851	0.027	0.037	0.070	0.0080	0.006	0.010
AD	0.483	0.472	0.436	0.005	0.005	0.015	0.003	0.003	0.001
AT	31.039	32.808	26.965	5.126	4.884	4.536	0.488	0.392	0.416

2. Baker B.S., Coffman E.G. Mutual Exclusion Scheduling. Theoretical Computer Science 162: 225–243 (1996).
3. Halldorson M M., Kortsarz G., Proskurowski A., Salman R., Shachnai H., and Telle J.A. Multicoloring trees. Informationand Computation 180(2), 113-129 (2003).
4. Bodlaender H.L., Jansen K. Restrictions of graph partition problems part I. Theoretical Computer Science 148: 93–109 (1995).
5. Even G., Halldorson M M., Kaplan L. and Ron D. Scheduling with conflicts: online and offline algorithms. J. Sch. 12: 199-224 (2009).
6. Lenstra J.K., Rinnooy Kan A.H.G. Computational complexity of discrete optimization. In Lenstra JK and AHG Rinnooy Kan and P Van Emde Boas(eds), Interfaces Between Computer Science and Operations Research, Proceedings of a symposium held at the Matematisch Centrum, Amsterdam: 64–85 (1979).
7. Cherian J., Maheshwari S.N. Analysis of preflow push algorithms for maximum network flow. SIAM Journal on Computing 18: 1057–1086 (1989).
8. Graham R.L. Bounds for certain multiprocessing anomalies. Bell System Technical Journal 45:1563–1581 (1966).
9. Sakai S., Togasaki M, Yamazaki K. A. note on greedy algorithms for maximum weighted independent set problem. Discrete Applied Mathematics 126: 313–322 (2003).