Scheduling Two Parallel Machines with a Single Server to Minimize Forced Idle Time

Keramat Hasani[1], Svetlana A. Kravchenko[2], Frank Werner[3]

[1]*Department of Computer Engineering, Malayer Branch, Islamic Azad University, Malayer, Iran*
hasani@iau-malayer.ac.ir

[2]*United Institute of Informatics Problems, Surganova St. 6, 220012 Minsk, Belarus*
kravch@newman.bas-net.by

[3]*Fakultät fur Mathematik,Otto-von-Guericke-Universität Magdeburg, Postfach 4120, 39016 Magdeburg, Germany*
frank.werner@ovgu.de

Abstract

In this paper, we consider the problem of scheduling a set of jobs on two parallel machines with setup times. The setup has to be performed by a single server. The objective is to minimize the forced idle time. The problem of minimizing the forced idle time (interference problem) is known to be unary NP-hard for the case of two machines and equal setup and arbitrary processing times. We propose an MILP model and a heuristic algorithm which are tested on instances with up to 100,000 jobs. The computational results indicate that the algorithm has an excellent performance even for very large instances, where mostly an optimal solution is obtained within a very small computational time.

Keywords: Scheduling, parallel machines, single server, interference problem, forced idle time.

MSC classification: 90 B 35

May 5, 2014

1. Introduction

This paper studies a deterministic scheduling environment on two identical parallel machines with a single server. The problem under consideration can be described as follows. A set of *n* independent jobs $\{J_1,\ldots,J_n\}$ has to be scheduled on two identical parallel machines M_1 and M_2 without preemptions. Each machine can

process at most one job at a time. For each job J_j, there is given its processing time p_j. Before processing any job, say job J_j, on a machine, say machine M_q, job J_j has to be loaded on M_q, which requires some time s_j called the loading time for job J_j. This loading, which is also called a setup, is performed by a single server. During such a setup, the job J_j, the machine M_q and the server are involved into this process for s_j time units, i.e., no other job can be processed on this machine or can be loaded by the server during this setup. Having completed a setup, the server is free to perform other setups, and the machine M_q has to start the processing of job J_j. Simultaneous requests for the server by machines will necessarily result in a machine idle time. A schedule specifies one time interval for each job on one machine. During such an interval, the prescribed job has to be loaded and processed on the prescribed machine. A schedule is feasible if no two intervals on the same machine overlap.

In the literature, mostly the problem $P2,S1||C_{max}$ of minimizing the makespan has been considered. For instance, complexity issues were addressed in (Hall et al. 2000) and (Brucker et al. 2002). Recent heuristics with an excellent performance were given e.g. in (Hasani et al. 2014a) and (Hasani et al. 2014b).

We consider the related problem $P2,S1||IT$, where the forced idle time or interference has to be minimized. Following (Koulamas 1996), we define IT as the amount of time in list scheduling when some machine is idle due to the unavailability of the server for setting up a job when needed, i.e., we disregard the idle time on a machine after all its processing is completed but before the other machine completes the processing of jobs. The consideration of this idle time as well would lead to the makespan problem. In (Koulamas 1996), it has been shown that problem $P2,S1||IT$ is unary NP-hard. In that paper also local search and beam search algorithms have been given and compared on instances with up to 300 jobs. (Koulamas and Smith 1988) gave a look-ahead heuristic for the case of continuously arriving jobs. In (Kravchenko and Werner 1997), the problem $P2,S1|s_i = s|IT$ was considered. The authors proved the unary NP-hardness of the problem and presented some polynomially solvable cases. In (Abdekhodaee and Wirth 2002), two forward and backward heuristics were proposed and tested. The results were compared only for instances with 100 jobs.

We note that the interference problem has found applications in Flexible Manufacturing Systems, where a robot is shared among several pieces of equipment for tool change and part setup purposes (Koulamas 1996).

The remainder of the paper is as follows. In Section 2, we present a mathematical model for this problem. In Section 3, a hybrid heuristic algorithm is given. Finally, in Section 4, we present computational results for instances with up to 100,000 jobs. Section 5 gives some concluding remarks.

2. A mathematical model

Next, we describe a mathematical model IT which finds an optimal schedule in the special class of schedules, where all jobs J_k, $k = 1,...,n,$ from the list π have to be scheduled in a staggered order, i.e., the job J_1, is scheduled on the first machine, job J_2 is scheduled on the second machine, job J_3 scheduled on the first machine and so on.

Let the variable $O_{i,j}$ define the position of the job J_j in the list π, i.e.,

$$O_{i,j} = \begin{cases} 1 & \text{if job } J_j \text{ is the } i\text{-th scheduled job,} \\ 0 & \text{otherwise.} \end{cases}$$

Moreover, let the variable ct_i denote the completion time of the i-th job J_j in the list π, st_i the starting time of loading the i-th job J_j in the list π, and cl_i the completion time of loading the i-th job J_j in the list π. Then, the minimization of interference is equivalent to the minimization of $\sum_{i=1}^{n-1} IT_i$, where IT_i, i = 2,…, n-1, denotes the idle time between the processing of the (i-1)-th and (i+1)-th loaded job, and IT_1, denotes the idle time which happens while loading the first job.

Since each position in the list π can be occupied by only one job, the equality

$$\sum_{i=1}^{n} O_{i,j} = 1 \tag{1}$$

holds for each $j = 1,\ldots,n$.

Since each job has to be placed on one position in the list π, the equality

$$\sum_{j=1}^{n} O_{i,j} = 1 \tag{2}$$

holds for each i =1,…,n.

Since between the starting time st_i and the finishing time ct_i of the i-th job J_j one has to load and to process job J_j, the equality

$$ct_i - st_i = \sum_{j=1}^{n} O_{i,j}(s_j + p_j) \tag{3}$$

holds for each i =1,…,n.

Since between the starting time st_i and the completion of the loading time cl_i of the i-th job J_j, one has to load the job J_j, the equality

$$cl_i - st_i = \sum_{j=1}^{n} O_{i,j} s_j \tag{4}$$

holds for each i =1,…,n.

Since the list π defines the order for loading all the jobs, the inequality

$$st_i \geq cl_{i-1} \tag{5}$$

holds for each $i = 2,\ldots,n$.

Since we consider schedules, where all jobs are processed in a staggered order, the inequality

$$st_i \geq ct_{i-2} \tag{6}$$

holds for each $i = 3,\ldots,n$.

During the loading of the first job, loading the second job in the list π has to be postponed. Thus, the equality

$$IT_1 = cl_1 - st_1 \tag{7}$$

holds.

An idle time may happen between the completion time of the $(i-1)$-th job and the starting time of the $(i+1)$-th job. Thus, for each $i = 2,...,n-1,$ the equality

$$IT_i = st_{i+1} - ct_{i-1}, \tag{8}$$

holds.

Now, to minimize the interference, one has to minimize

$$\sum_{i=1}^{n-1} IT_i \tag{9}$$

Here $O_{i,j}$ are binary variables; ct_i, st_i, cl_i, and IT_i are non-negative variables; p_j and s_j are the processing and setup times known in advance.

3. A heuristic algorithm

In this section, we present a heuristic algorithm to minimize the interference in the scheduling problem with a single server. This hybrid algorithm combines a constructive algorithm called *GenerateSchedule* with a tabu search algorithm to find near-optimal schedules.

One can see that an optimal schedule for the problem $P2,S1||IT$ can be found in the class of list schedules, i.e., one can consider a list (sequence) of the jobs π, where all jobs are placed into a list and the algorithm *GenerateSchedule* schedules the first unscheduled job from the list whenever a machine becomes idle. In fact, the algorithm generates a solution by sequencing the jobs in such a way that an idle time among the jobs is avoided as far as possible and by means of tabu search, the generated schedule is iteratively improved. In the following, we first explain algorithm *GenerateSchedule* in detail.

Algorithm *GenerateSchedule*

BEGIN

0. Determine the list of jobs π, where all jobs are sequenced in non-increasing order of their setup times.

1. Denote the last job in the list π by F and set $L(F)$ to be equal to the processing time of F;

2. WHILE scheduling all jobs DO

2.1. If the job F is an unscheduled job, schedule it;

2.2. Starting from the beginning of the list π, take the first unscheduled job J_k, such that $s_k \leq L(F)$ and $L(F) \neq p_k + s_k$. In the case when it is impossible to find such a job, go to 2.3;

2.2.1. Schedule J_k on the earliest available machine, say M_q, and set $L(F)$ to be equal to the difference of the completion time of the machine M_{3-q} and the completion time of the machine M_q, where M_{3-q} is the machine which processes F and M_q is the machine which processes J_k;

2.2.2. IF $L(F) > 0$, go to 2.2;

2.2.3. IF $L(F) < 0$, denote J_k by F and calculate $L(F)$, i.e., set $L(F)$ to be equal to the difference of the completion time of the machine M_q and the completion time of the machine M_{3-q}, where M_q is the machine which processes J_k, and M_{3-q} is the other machine. Go to 2;

2.2.4. IF $L(F) = 0$, go to 2.3;

2.3. Starting from the end of the ordered list π, determine an unscheduled job J_k. Denote J_k by F and let $L(F)$ denote the processing time of F. Go to 2;

END.

The complexity of the algorithm *GenerateSchedule* is $O(n^2)$.

The algorithm *GenerateSchedule* considers a list of jobs which is ordered according to non-increasing setup times as input (step 0), and the last job in the ordered list is always considered as the first scheduled job and also as an initial job F (step 1).

To choose the next job, this algorithm looks for those jobs for which their loading part can fully coincide with the overlapping part of the job F. For this purpose, an unscheduled job with the 'fittest' setup time (the largest one which is not larger than $L(F)$ and also its length is not equal to $L(F)$) is considered. If there are several jobs with an equal setup time, the earliest job in the list is chosen. To do so, the ordered list is searched from the beginning and as soon as a fit job is found, it is considered as the next scheduled job (step 2.2). By this strategy, no idle time will occur for scheduling the next job, since the previous job has been completely loaded and the new job can be loaded by the server without any postponement. Moreover, since choosing a job the length of which is equal to the overlapping part of the job F leads to an unavoidable idle time for the next two jobs, our algorithm avoids such a situation as much as possible. When such a fit job is found, having scheduled the found job, if it is completed after completing the job F, i.e., $L(F) < 0$, the job F will be changed to the currently scheduled job (step 2.2.3). However, when such a fit job cannot be found or no overlapping part of F is left, i.e., $L(F) = 0$, the occurrence of a server idle time is unavoidable. To minimize the idle time caused in such cases, the

algorithm tries to find an unscheduled job for which the least amount of its loading time is left to be loaded outside of the overlapping part. If there is more than one eligible job with the same setup time, in such a case, the latest job in the list will be chosen (step 2.3).

To illustrate the algorithm *GenerateSchedule*, we consider the following example.

Example 1. Consider a problem with 6 jobs, 2 machines and a single server. The setup and processing times given as follows:

$s_1 = 9, p_1 = 2, s_2 = 4, p_2 = 4, s_3 = 5, p_3 = 3, s_4 = 1, p_4 = 8, s_5 = 4, p_5 = 3, s_6 = 1, p_6 = 3.$

First, the jobs have to be ordered according to non-increasing setup times and for equal setup times, a non-decreasing order of the processing time is considered. Thus, the ordered list of jobs is $\pi = (J_1, J_3, J_5, J_2, J_6, J_4)$. The last job in the list (J_4) is considered as the first scheduled job, and we set $F = J_4$. Now, due to the length of the overlapping part of F, denoted by $L(F)$, the fittest job from the list is job J_5. Here the job J_3 cannot be chosen, since the length of J_3 is equal to $L(F)$ and choosing J_3 will create an idle time, and between J_5 and J_2, which have exactly the same setup time, the job J_5 is chosen, since J_5 is before J_2 in the list π. By scheduling the job J_5, $L(F)$ is updated. As illustrated in Fig. 1, we have $L(F) = 1$.

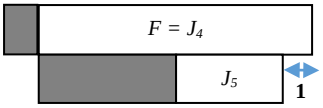

Fig. 1: Adding J_5 to the schedule

Since the job F has still an overlapping part, another fittest job has to be found. Among the unscheduled jobs, the fittest one which can be overlapped with F is the job J_6. By scheduling the job J_6, we have $L(F) = 0$ and J_4 can no longer be used as the job F. Thus, we change the job F to the job J_6. This situation is illustrated in Fig. 2.

Fig. 2: Adding J_6 to the schedule

Now, we have $L(F) = 3$ and there is no unscheduled jobs which can be fully overlapped with the processing part of job *F*. Therefore, a forced idle time will happen. We can make it as small as possible by choosing the job J_2. By adding the job J_2, we get $L(F) = -1$ and *F* is changed to the job J_2 (see Fig. 3).

Fig. 3: Adding J_2 to the schedule

Furthermore, job J_3 is selected to be overlapped with the current job F and an idle time happens, and finally the only remaining job J_1 is scheduled. Fig. 4 displays the final schedule. The sum of the idle times is 3.

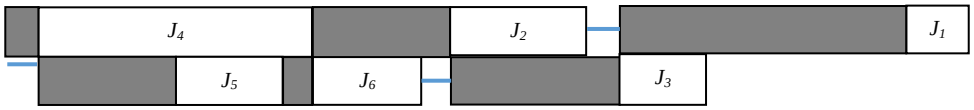

Fig. 4: the obtained schedule

The schedule presented in Example 1 is not optimal and was obtained according to the ordered list $\pi = (J_1, J_3, J_5, J_2, J_6, J_4)$. Only by exchanging jobs J_6 and J_4 in the list π and calling algorithm *GenerateSchedule* with the updated input list π again, an optimal schedule will be obtained.

Therefore, for further improvements of the constructed schedule, we have developed a tabu search algorithm. The first step of this algorithm sorts all jobs in non-increasing order of the setup times and for the jobs with equal setup times, considers a non-decreasing order of the processing times.

The second step finds the groups of jobs with equal setup times. In principle, in each iteration our tabu search algorithm calls *GenerateSchedule* with an updated ordered list of jobs. The neighbourhood function exchanges the positions of two jobs with equal setup times and different processing times in that list. Moreover, as a stopping criterion, the authors used the first of the following events:

- within the last 500 iterations, no improvement of the best objective function value was obtained, or
- the currently best sum of interference times, *BestSumIT*, is equal to min{ s_i }, i = 1,…,n.

The tabu search algorithm can be described as follows.

Algorithm *Tabu Search*

BEGIN

1. Sort all jobs in non-increasing order of the setup times and for the jobs with equal setup times, consider a non-decreasing order of the processing times and assume that $\pi = (J_1, J_2, \ldots, J_n)$ is the resulting ordered list of the jobs;

2. Find groups of jobs with equal setup times in the list π and for each of these groups, determine the vector $d(i,N_i)$, where i denotes the starting position of the group in the list and $N_i \geq 2$ denotes the number of jobs in this group;

3. Initialization:
Set *BestSumIT* = *GenerateSchedule*(π); Set *BestOrder* = π;
Set *TabuSize* = 10; Set *TabuList* = Ø;

4. Looking for the best schedule:
WHILE (stopping criterion is not met) DO
π' = randomly determined neighbour of list π by exchanging two jobs belonging to the same group (by means of vector d);
NewSumIT = *GenerateSchedule*(π'); Δ = *NewSumIT* – *BestSumIT*;
IF (π' is not tabu) THEN
IF ($\Delta \leq 0$) THEN
Set $\pi = \pi'$;
IF (*NewSumIT* < *BestSumIT*) THEN
BestSumIT = *NewSumIT* ;
END IF;
Make π' to be a tabu and update *TabuList*;
END IF;
END IF;
ELSE
IF (*NewSumIT* = *BestSumIT*) THEN
Set $\pi = \pi'$;
END IF;
END WHILE;

5. Output *BestSumIT* together with the corresponding schedule;
END.

4. Computational results

Both the algorithms *Tabu Search* and *GenerateSchedule* have been implemented using the Java programming language. The test instances for the model IT have been solved using CPLEX 10.1. All instances have been run on a personal computer Intel(R) Core(TM) i5-2430M CPU @2.4GHz with 2GB of memory available for working storage.

For testing our hybrid heuristic and the mixed integer linear programming model, the data has been generated in the same way as it has been described in (Koulamas 1996). Therefore, we consider two different cases of instances. In the *uncorrelated* case, we have $p_i \overset{d}{=} U(0, 100)$. Thus, the processing times p_i are uniformly distributed in the interval (0, 100), and the setup times s_i are distributed uniformly in the interval (0, 100*L*), where *L* is the server load. In the *correlated* case, we have $s_i = L * p_i$.

For all values of n, 5 instances were generated for each $L \in \{0.7, 0.8, 0.95\}$ and for both cases. The model IT was used for the instances with $n \in \{8, 20, 50, 100, 200, 300\}$ with the run time limit of $\min\{(300/8)n, 3600\}$ seconds the same as (Hasani et al. 2013). The hybridization of algorithm *GenerateSchedule* and *Tabu Search* was used for the instances with $n \in \{8, 20, 50, 100, 200, 300, 500\}$ jobs and for the instances with $n \geq 500$, only the algorithm *GenerateSchedule* was used.

Next, the results obtained for the instances with $n \leq 500$ for the model IT and the hybrid heuristic are evaluated and compared with the results obtained in (Koulamas 1996) and (Abdekhodaee and Wirth 2002). The computational results with the model IT and the hybrid heuristic for the same instances with uncorrelated values of s_i and p_i with $n \in \{8, 20, 50, 100, 200, 300\}$ are given in Table 1. In the columns Mean IT and Max IT, we present the average and maximal values of the relation 'total interference time/total length of jobs', and in the column Mean Time we present the average computational time in seconds.

		Hybrid heuristic			model IT		
L	n	Mean IT	Max IT	Mean Time	Mean IT	Max IT	Mean Time
0.7	8	0.024282	0.075812	0.003	0.024621	0.075812	0.06
	20	0.03024	0.065063	0.04	0.007196	0.024719	203.93
	50	0.005583	0.019709	0.021	0.007531	0.018067	1875.06
	100	0.000116	0.000124	0.01	0.006983	0.01853	3600.0
	200	0.001156	0.003564	0.27	0.021461	0.045554	3600.0
	300	0.000039	0.000041	0.01	0.047992	0.056385	3600.0
0.8	8	0.079308	0.156944	0.024	0.034461	0.074643	0.04
	20	0.041202	0.077799	0.03	0.019471	0.052525	451.84
	50	0.008985	0.035043	0.04	0.011711	0.022975	1875.01
	100	0.016647	0.02946	0.15	0.024035	0.035588	3600.0
	200	0.000679	0.001728	0.32	0.021222	0.030555	3600.0
	300	0.001356	0.002552	0.71	0.055913	0.067435	3600.0
0.95	8	0.084395	0.14876	0.02	0.051527	0.102715	0.04
	20	0.047717	0.161022	0.02	0.023216	0.08884	502.03
	50	0.03732	0.080978	0.05	0.029948	0.058729	1875.28
	100	0.016227	0.037548	0.11	0.026299	0.042046	3600.0
	200	0.012012	0.035622	0.31	0.045615	0.083711	3600.0
	300	0.011069	0.017451	0.63	0.079025	0.09233	3600.0

Table 1. Comparison of the model IT and the hybrid heuristic for uncorrelated instances

Comparing the obtained results in the case of uncorrelated instances with the results in (Koulamas 1996) and (Abdekhodaee and Wirth 2002), the following summary can be given for the hybrid heuristic and the model IT:

- For n = 8, the model IT can always obtain an optimal schedule while for the hybrid heuristic, 33% of the instances were optimally solved. However, the hybrid heuristic was better in terms of computational time.
- For n = 20, the model IT is superior to the hybrid heuristic in terms of the quality of the results and this model can obtain an optimal schedule for about 47 % of the instances, while the hybrid heuristic can obtain an optimal schedule for about 7% of the instances. However, the hybrid heuristic was clearly better in terms of computational time.
- For $n \in \{50, 100, 200, 300\}$, the hybrid heuristic was superior to the model IT in terms of computational time and the quality of the results, i.e, using the hybrid heuristic, 33.3% of the schedules were optimally solved for n = 50, 40% were optimally solved for n = 100, 20% were optimally solved for n = 200, and 47% were optimally solved for n = 300. However, for all values of the server load and $n \in \{100, 200, 300\}$, our results substantially outperform the results presented in (Koulamas 1996) and (Abdekhodaee and Wirth 2002). It must be noted that in (Koulamas 1996), values of Min IT up to 0.108 and of Max IT up to 0.120 were reported, and in (Abdekhodaee and Wirth 2002), for n = 100, a value of Min IT up to 0.05 was reported.

		Hybrid heuristic			model IT		
L	*n*	Mean IT	Max IT	Mean Time	Mean IT	Max IT	Mean Time
0.7	8	0.045922	0.065527	0.02	0.039931	0.058405	0.05
	20	0.012695	0.026282	0.13	0.008542	0.012702	528.62
	50	0.002863	0.008282	0.03	0.002004	0.00284	1687.43
	100	0.000474	0.000604	0.06	0.000681	0.000919	3494.38
	200	0.000192	0.000237	0.2	0.000557	0.000675	3600.0
	300	0.000126	0.00016	0.5	0.000373	0.000512	3600.0
0.8	8	0.062004	0.076389	0.02	0.060162	0.076389	0.078
	20	0.021235	0.03352	0.06	0.011971	0.016201	672.28
	50	0.005957	0.007546	0.04	0.003712	0.004757	1875.22
	100	0.00148	0.002054	0.12	0.001093	0.001182	3600.0
	200	0.000349	0.000659	0.4	0.000811	0.001078	3600.0
	300	0.00015	0.000181	0.86	0.000612	0.000677	3600.0
0.95	8	0.087428	0.105485	0.01	0.087246	0.105485	0.10
	20	0.042204	0.049766	0.06	0.040656	0.045667	750.09
	50	0.014631	0.017689	0.04	0.014746	0.018563	1875.64
	100	0.006294	0.007603	0.01	0.007748	0.010746	3600.0
	200	0.002912	0.004592	0.323	0.010141	0.011972	3600.134
	300	0.001538	0.002104	0.4458	0.005484	0.00656	3600.184

Table 2. Comparison of the model IT and the hybrid heuristic for correlated instances

The computational results with the model IT and the hybrid heuristic for small and medium size instances with correlated values of s_i and p_i for $n \in \{8, 20, 50, 100, 200, 300\}$ are given in Table 2.

Comparing the obtained results for the hybrid heuristic and the model IT with the results in (Koulamas 1996) and (Abdekhodaee and Wirth 2002) in the case of correlated instances, the following summary can be given:

- For $n \in \{8, 20, 50, 100\}$, the model IT obtained slightly better results than the hybrid heuristic. However, in terms of computational time, the hybrid heuristic is significantly superior. The model IT obtained an optimal schedule for 100 % of the instances with $n = 8$ and for about 46.7% of the instances with $n = 20$, while the hybrid heuristic can obtain an optimal schedule for 53.3 % of the instances with $n = 8$, for 6.7 % of the instances with $n = 20$, for 33.3 % of the instances with $n = 50$, and for about 6.7 % of the instances with $n = 100$. Our results are clearly better for $n = 100$ and all values of the server load than those presented in (Koulamas 1996) and (Abdekhodaee and Wirth 2002). It must be noted that in (Koulamas 1996), for $n = 100$, values of Mean IT up to 0.091 and of Max IT up to 0.108 were reported, and in (Abdekhodaee and Wirth 2002) for $n = 100$, a value of Min IT about 0.01 was reported.
- For $n \in \{200, 300\}$, the hybrid heuristic works slightly better than the model IT in terms of the quality of the results. However, in terms of computational time, the model IT is not competitive to the hybrid heuristic. The percentage of optimally solved instances with $n = 200$ jobs is about 6.7% for the hybrid heuristic and for instances with $n = 300$, neither the model IT nor the hybrid heuristic can find an optimal schedule. Our results for $n = \{200, 300\}$ were substantially better for all values of the server load than those presented in (Koulamas 1996), where values of Mean IT up to 0.096 and of Max IT up to 0.112 were reported.

Next, the computational results with the heuristic for the large and very large instances with both correlated and uncorrelated data generation are presented. Here**,** no computational results for other algorithms are available in the literature.

We note that in Table 3, the hybridization of algorithm *GenerateSchedule* and *Tabu Search* was used only for $n = 500$ and for the remaining instances, only algorithm *GenerateSchedule* was used.

		Uncorrelated			Correlated		
L	n	Mean IT	Max IT	Mean Time	Mean IT	Max IT	Mean Time
0.7	500	0.000272	0.001199	1.16	0.000084	0.000097	1.30
	1000	0.000712	0.003478	0.02	0.000108	0.000159	0.02
	2500	0.000010	0.000024	0.10	0.000064	0.000151	0.10
	5000	0.000004	0.000009	0.39	0.000049	0.000116	0.38
	7500	0.000002	0.000002	0.88	0.000068	0.000115	1.10
	10000	0.000001	0.000001	1.70	0.000004	0.000005	1.56
	25000	0.000004	0.000016	13.76	0.000002	0.000002	13.04
	50000	0.000000	0.000000	77.09	0.000045	0.000115	79.78
	75000	0.000001	0.000005	195.13	0.000067	0.000115	166.53
	100000	0.000000	0.000000	365.59	0.000023	0.000112	355.05
0.8	500	0.000713	0.003454	0.84	0.000097	0.000112	2.13
	1000	0.000869	0.002968	0.02	0.000066	0.000141	0.02
	2500	0.000076	0.000191	0.13	0.000094	0.000133	0.12
	5000	0.000083	0.000392	0.48	0.000024	0.000083	0.47
	7500	0.000004	0.000012	1.09	0.000072	0.000124	0.84
	10000	0.000005	0.000020	1.92	0.000049	0.000122	1.95
	25000	0.000002	0.000010	15.88	0.000043	0.000110	16.02
	50000	0.000001	0.000002	69.79	0.000061	0.000105	69.69
	75000	0.000002	0.000011	168.21	0.000087	0.000113	188.26
	100000	0.000000	0.000000	331.47	0.000065	0.000112	365.18
0.95	500	0.006442	0.011397	1.83	0.000403	0.000448	1.44
	1000	0.001968	0.006380	0.01	0.000233	0.000286	0.01
	2500	0.001368	0.004103	0.09	0.000142	0.000199	0.10
	5000	0.000287	0.001348	0.36	0.000114	0.000159	0.36
	7500	0.000131	0.000331	0.83	0.000067	0.000137	1.06
	10000	0.000153	0.000409	1.65	0.000076	0.000117	1.49
	25000	0.000067	0.000278	13.28	0.000047	0.000112	12.64
	50000	0.000003	0.000009	72.53	0.000043	0.000109	75.56
	75000	0.000003	0.000011	183.51	0.000043	0.000103	168.39
	100000	0.000002	0.000007	331.98	0.000021	0.000099	348.17

Table 3. Results for both correlated and uncorrelated, large size and very large size instances

It can be noted that even for the large instances with uncorrelated data, n = 100,000 and L = 0.95, the maximal value of the relation ‘total interference time/total length of jobs’ was only 0.000007, and the average value of the relation ‘total interference time/total length of jobs’ was 0.000002. These results were obtained within no more than 332 seconds. For the case of correlated instances with 100,000 jobs, the maximal value of the relation ‘total interference time/total length of jobs’ for L = 0.8 was only 0.000112, and the average value of the relation ‘total interference time/total length of jobs’ was 0.000065. These results were obtained within no more than 366 seconds.

In Figure 5, the percentage of optimally solved instances obtained by the heuristic is shown. For the interference problem, we considered the schedules with an interference equal to the minimal s_i value as an optimal one and for the other schedules with n = {8, 20, 50, 100, 200, 300}, we also discovered the optimality through a comparison with those schedules considered as an optimal one by CPLEX.

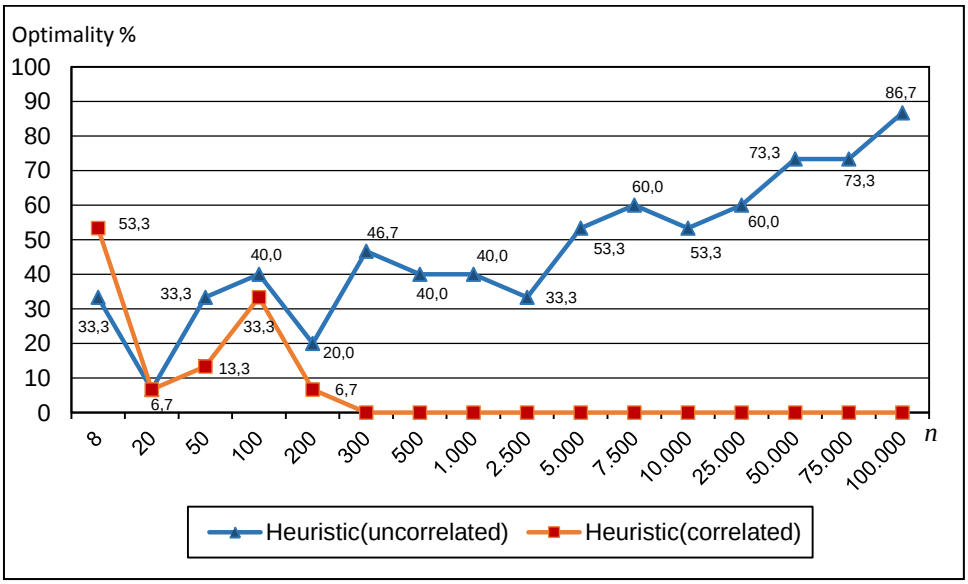

Figure 5. The percentage of optimally solved (correlated and uncorrelated) instances using the heuristic for all values of L from 0.7 to 0.95

Next, the model IT and the heuristic are compared from an overall point of view. In Figure 6 and Figure 7, respectively, the average values of the relation 'total interference time/total length of jobs' in dependence on the number of jobs are presented for instances with uncorrelated and correlated values of s_i and p_i (the average value is taken over all instances with all values L considered).

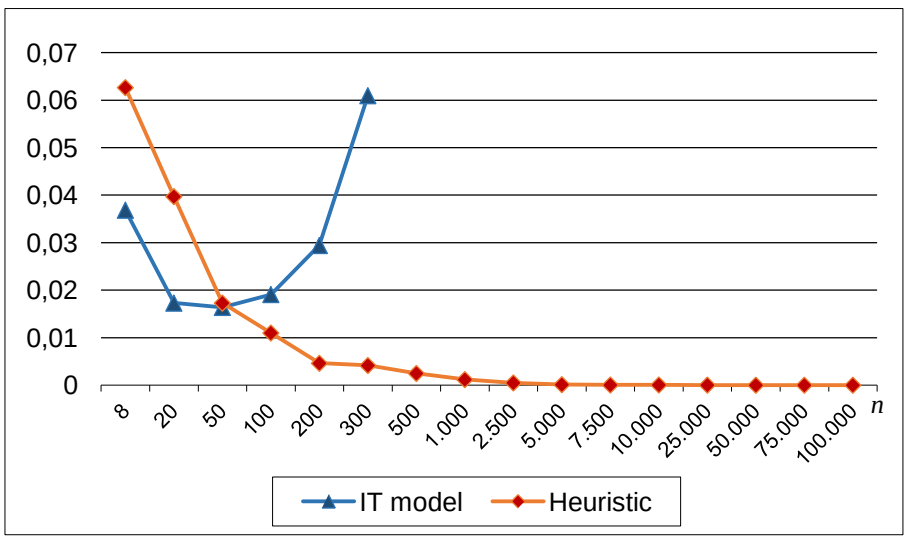

Figure 6. Average values of the relation ‘total interference time/total length of jobs’ for all uncorrelated instances over all values of L from 0.7 to 0.95

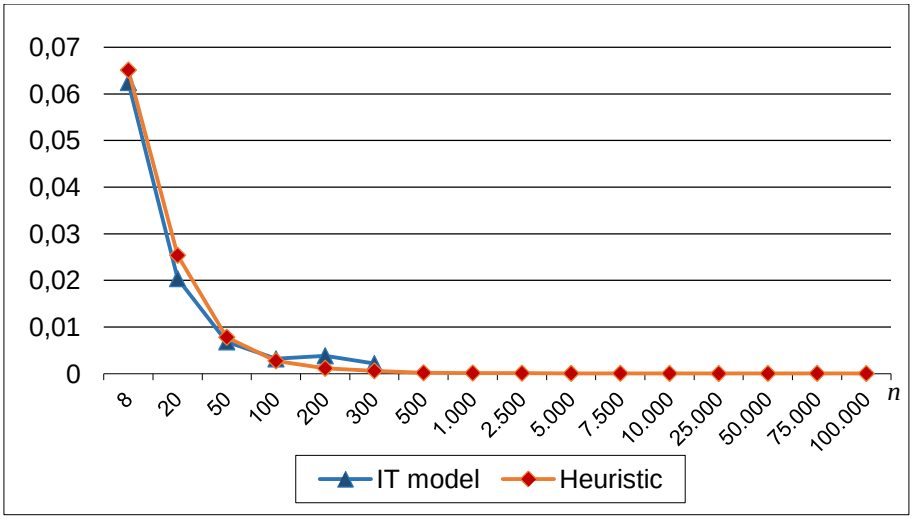

Figure 7. Average values of the relation ‘total interference time/total length of jobs’ for all correlated instances over all values of L from 0.7 to 0.95

It turned out that for both cases, although the number of jobs grows, the quality of the results obtained by the heuristic is increasing. However, for the uncorrelated instances, with increasing the number of jobs, the quality of the results obtained by the IT model is decreasing. In contrast, for the correlated instances, the IT model behaves like the heuristic.

Next, the influence of the values of the server load L on the quality of the final solution is evaluated and also the quality of the results obtained for the correlated and uncorrelated instances is compared.

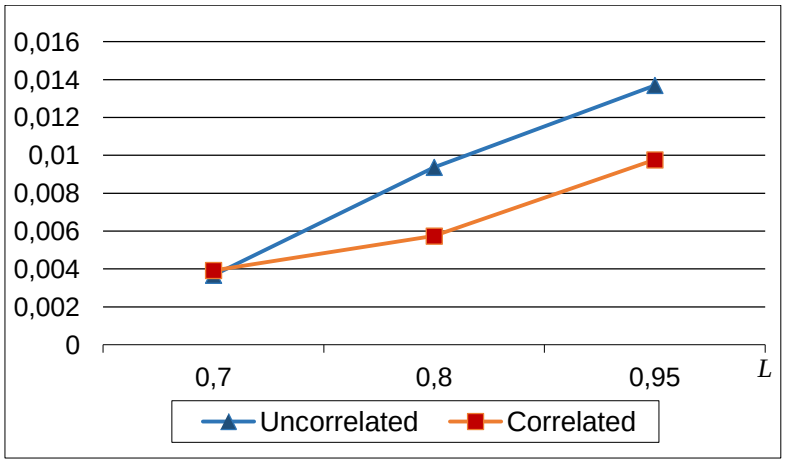

Figure 8. The dependence of the average value of the relation ‘total interference time/total length of jobs’ on the value L is shown for all (correlated and uncorrelated) instances

In Figure 8, the dependence of the average value of the relation ‘total interference time/total length of jobs’ on the value *L* is shown for all instances and for both correlated and uncorrelated instances when applying the heuristic (for the small instances the hybrid one and for the large instances only algorithm *GenerateSchedule* as discussed above are considered). From Figure 8, it can be seen that from an overall point of view, the uncorrelated instances were harder for our heuristic. Moreover, for both cases, the size of the server load *L* influences the quality of results. The larger total values of interference time occur for the instances including jobs with larger loading times.

5. Concluding remarks

In this paper, a mixed integer linear programming model and a hybridization of a constructive algorithm and a tabu search algorithm have been developed for the problem of scheduling a set of jobs on two identical parallel machines with a single server to minimize the total interference time. The mixed integer linear programming model and the heuristic have been tested for instances with up to 300 and 100,000 jobs, respectively, for both correlated and uncorrelated cases.

The heuristic performed extremely well in terms of the quality of the results and also in terms of the low computational times, especially for large and very large instances. It significantly outperformed the results presented in (Koulamas 1996) and (Abdekhodaee and Wirth 2002).

For small size instances, the mixed integer linear programming model performed well and was somehow even a bit superior to the heuristic. However, in terms of computational time, it was not competitive to the heuristic. The presented model also outperformed the results presented in (Koulamas 1996) and (Abdekhodaee and Wirth 2002).

For future work, it is intended to investigate several types of fast heuristics for the interference problem with a single server with an arbitrary number of machines.

References

Abdekhodaee, A. and A. Wirth (2002). "Scheduling parallel machines with a single server: some solvable cases and heuristics." Computers & Operations Research **29**: 295-315.

Brucker, P., C. Dhaenens-Flipo, S. Knust, S.A. Kravchenko and F. Werner (2002). "Complexity results for parallel machine problems with a single server." Journal of Scheduling **5**: 429 - 457.

Hall, N.G., C.N. Potts and C. Sriskandarajah (2000). "Parallel machine scheduling with a common server." Discrete Applied Mathematics **120**: 223 - 243.

Hasani, K., S.A. Kravchenko and F. Werner (2014a). "Simulated annealing and genetic algorithms for the two-machine scheduling problem with a single server." To appear in International Journal of Production Research **52**: 2014, 15 pages (DOI: 10.1080/00207543.2013.874607) .

Hasani, K., S.A. Kravchenko and F. Werner (2014b). "Minimizing the makespan for the two-machine scheduling probelm with a single server: Two algorithms for very large instances". Preprint 01/14, Faculty of Mathematics, Otto-von-Guericke-University Magdeburg, 2014, 19 pages.

Hasani, K., S.A. Kravchenko and F. Werner (2013). "Block models for scheduling jobs on two parallel machines with a single server." Computers & Operations Research **41**: 94-97.

Koulamas, C.P. (1996). "Scheduling two parallel semiautomatic machines to minimize machine interference." Computers and Operations Research **23**(10): 945-956.

Koulamas, C.P. and M. L. Smith (1988). "Look-ahead scheduling for minimizing machine interference." International Journal of Production Research **26**: 1523-1533.

Kravchenko, S. and F. Werner (1997). "Parallel machine scheduling problems with a single server." Mathematical and Computer Modelling **26**(12): 1-11.