Minimizing the Makespan for the Two-Machine Scheduling Problem with a Single Server: Two Algorithms for Very Large Instances

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

February 10, 2014

Abstract

In this paper, we consider the problem of scheduling a given set of *n* jobs on two identical parallel machines with a single server. Each job must be processed on one of the machines. Prior to processing, the server has to set up the relevant machine. The objective is to minimize the makespan. For this unary NP-hard problem two fast algorithms with a complexity of $O(n^2)$ are presented. The performance of these algorithms is evaluated for instances with up to 10000 jobs. Computational results indicate that the algorithms have an excellent performance for very large instances so that the objective function values obtained are very close to a lower bound, and in many cases even an optimal solution is achieved. The superiority over existing algorithms is obtained by sequencing the jobs on two machines so that the machine idle time and the server waiting time are minimized. In doing so, we follow the characteristics of an optimal solution resulting from its relevant lower bound.

Keywords: Scheduling, Parallel machines, Single server, Lower bound, Idle time, Server waiting time

1. Introduction

In this paper, we consider a parallel machine scheduling problem with a single server and the minimization of total weighted completion time, i.e., problem $P2,S1||C_{\max}$ using the standard scheduling notation. This problem can be formulated as follows. A set of n jobs $1,\ldots,n$ has to be processed by a set of two parallel machines M_1 and M_2. Before processing, each job j has to be loaded on the machine, on which this job is processed at once after loading. The loading procedure requires both the server and the machine for s_j time units. The processing time p_j and the setup time s_j for each job j are known in advance. Each job can be processed by an arbitrary machine, and each machine and also the server can perform only one job at a time. We want to find a feasible schedule that minimizes the makespan.

It can be noted that scheduling problems with a single server have their applications in several manufacturing systems, among others in automated material handling systems, robotic cells or in the semiconductor industry, see Kim and Lee (2012).

This problem has first been considered by Kravchenko and Werner (1997) and Hall *et al.* (2000). It is strongly NP-hard since the problem $P2,S1|s_j = s|C_{\max}$ is strongly NP-hard, see Hall *et al.* (2000). Abdekhodaee and Wirth (2002) presented a mixed integer programming formulation, considered two special cases and gave two simple backward/forward *O(n log n)* heuristics for the general case. For the two-machine problem $P2,S1 \,||\, C_{max}$ under consideration, Abdekhodaee *et al.* (2006) presented and tested two versions of a greedy heuristic, a genetic algorithm and a version of the Gilmory-Gomory algorithm. The analysis started in Abdekhodaee *et al.* (2006) was extended in Gan *et al.* (2012), where two mixed integer linear programming formulations and two variants of a branch-and-price scheme were developed. Computational experiments have shown that for small instances with $n \in \{8,20\}$ jobs, one of the mixed integer linear programming formulations was the best whereas for the larger instances with $n \in \{50,100\}$, the branch-and-price scheme worked better, see Abdekhodaee *et al.* (2006). Hasani *et al.* (2014a) suggested several mixed integer programming formulations, based on blocks and setups, which turned out to be superior to the algorithms presented in Abdekhodaee *et al.* (2006) and Gan *et al.* (2012). Hasani *et al.* (2014b) developed two metaheuristics and the obtained results showed a superiority to all previous algorithms and models proposed in the recent literature for instances with up to 1000 jobs. Zhang and Wirth (2009) considered the on-line version of the problem $P2,S1 \,||\, C_{max}$. They analysed some special cases of this problem and proved an asymptotic competitive ratio for some fast heuristics. The interested reader is referred to Brucker *et al.* (2002) and Werner and Kravchenko (2010) for additional information on server scheduling models.

It can also be noted that the problem $P,S1 \mid\mid C_{max}$ including an arbitrary number of machines has been considered in Kim and Lee (2012), where several heuristics and MIP models have been suggested and compared on instances with up to 40 jobs. In particular, they also tested their heuristics on a small number of instances for the case of $m = 2$ machines, however, in the latter case only rather small instances with up to 20 jobs were considered. Huang *et al.* (2010) considered the problem with m parallel machines, a single server and the C_{max} criterion. However, in their model the setup times are sequence-dependent (the setup time depends on the job previously processed), and the set of machines is dedicated. They developed a hybrid genetic algorithm and demonstrated the effectiveness of their algorithm on instances with up to 100 jobs and 10 machines. It should be mentioned that an excellent survey of scheduling problems with setup times or costs has been given by Allahverdi *et al.* (2008).

Hasani *et al.* (2014a) proposed some mixed integer programming models applied to instances with up to 250 jobs. In Hasani *et al.* (2014b), large instances with up to 1000 jobs were considered for the first time. The obtained average gap from the lower bound for the instances with 1000 jobs was not better than 6% within a time limit of 3600 seconds. It seems that using their proposed heuristics for instances with more than 1000 jobs may not be efficient or even may be impossible. This fact motivated us to develop a new approach being more efficient and faster for scheduling the jobs of very large instances. In this paper, two different types of instances are identified which cover all possible generated instances and for each of these types, an appropriate algorithm is proposed. Our experiments turned out that the obtained deviations from the lower bound for both algorithms are very small and mostly about 0% within no more than 372 seconds for the hardest instances with even 10000 jobs.

The remainder of the paper is organized as follows. Section 2 presents some basic concepts for the problem under consideration. Section 3 describes our proposed algorithms. In Section 4, computational results for both algorithms are given, and the performance is compared with that of the algorithms existing in the literature. Section 5 gives some concluding remarks.

2. Basic Concepts

To evaluate the quality of the solutions, the following lower bound was proposed in Abdekhodaee and Wirth (2002).

$$LB = \max\{LB_1, LB_2, LB_3\},$$

where

$$LB_1 = \frac{1}{2}\left(\sum_{i\in J}(s_i + p_i) + \min_{i\in J}\{s_i\}\right),$$

$$LB_2 = \sum_{i\in J} s_i + \min_{i\in J}\{p_i\},$$

$$LB_3 = \max_{i\in J}\{s_i + p_i\}.$$

We found out that the relevant lower bound on the optimal objective function value for all instances which are generated by means of a uniform distribution in such a way as we will describe in Section 4, is either LB_1 or LB_2 while LB_3 does not play a role. In fact, the considered bounds have some of the characteristics of an optimal or near-optimal solution, i.e., LB_1 corresponds to a sequence of the jobs, where the first job in the sequence is that with shortest loading time and, if there is no gap between the jobs, as shown in Figure 1. Similarly, LB_2 corresponds to a sequence of the jobs, where no server waiting time occurs, and the last job in the schedule is that with smallest processing time as shown in Figure 2.

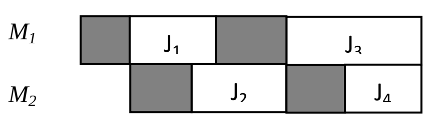

Figure 1. An optimal schedule with $LB = LB_1$

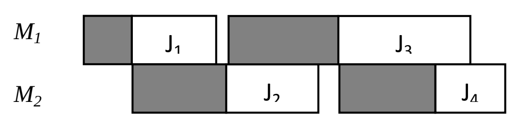

Figure 2. An optimal schedule with $LB = LB_2$

Therefore, we can conclude that, by following the characteristics of an optimal schedule, reaching an optimal or a schedule close to an optimal one is possible. In the next section, we will describe our methods in more detail.

3. Two heuristic algorithms

To minimize the makespan for the scheduling problem with two parallel machines and a single server, two algorithms are proposed which work according to the general structure of optimal schedules. As we described in the previous section, one can find two types of optimal schedules, where the first one can be generated based on minimizing the machine idle time and the other tries to minimize the gaps between the loading of two jobs (i.e., the server waiting times). One can easily prove that for the first type of optimal schedules, we always have $LB = LB_1$ and for the second type, we always have $LB = LB_2$. In the following, we propose Algorithm *Min-idle* for instances with $LB = LB_1$ and Algorithm *Min-loadgap* for the other case.

3.1 Algorithm *Min-idle*

In this section, we describe the proposed algorithm which is appropriate for the case of instances with $LB = LB_1$. In this case, we try to minimize the gap between completing a job and starting the next job (i.e., the machine idle time). However, it does not guarantee to find an optimal solution but often a near-optimal schedule or even an optimal schedule is obtained. The following example considers a scheduling instance belonging to the class $LB=LB_1$ using Algorithm *Min-idle*.

Example 1. Consider an instance with five jobs and the setup and processing times given as follows:

$s_1= 7$, $p_1 = 2$, $s_2 = 3$, $p_2 = 5$, $s_3 = 4$, $p_3 = 7$, $s_4 = 6$, $p_4 =10$, $s_5 = 1$, $p_5 = 3$.

The lower bound on the optimal function value is 24.5 and the bound is reached by LB_1. According to LB_1, the first job to be scheduled is that having the smallest setup time. Therefore, we consider the job J_5 as the first job denoted by $J_1^{'}$. Then we must decide about the second job according to the processing time of job $J_1^{'}$. To minimize the gap as much as possible, the best way is to choose a job whose loading time is large enough but is also not larger than the processing time of job $J_1^{'}$. Thus, we have to select the job J_2 as a second job ($J_2^{'}$). By scheduling job $J_2^{'}$, no overlapping part of job $J_1^{'}$ denoted by $L(J_1^{'})$ remains. As shown in Figure 2, job $J_1^{'}$ is run on machine M_1 and $J_2^{'}$ is run on machine M_2. Therefore, we consider

$$L(J_1^{'}) = C(M_1) - C(M_2) = 4 - 9 = -5,$$

where $C(M)$ denotes the completion time of the last job scheduled on machine M. The obtained negative value shows that no longer other jobs can be overlapped with job $J_1^{'}$. However, we have

$$L(J_2^{'}) = C(M_2) - C(M_1) = 5.$$

Thus, we can continue with job $J_2^{'}$ (see Figure 3).

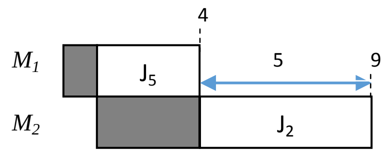

Figure 3: Adding J'_2 to the schedule.

The third unscheduled job, which we have to determine, must have the largest setup time which is not greater than $L(J'_2)$. Among the remaining unscheduled jobs, job J_3 will be chosen. By scheduling job J_3, we have $L(J'_2)<0$. Therefore, we consider job J_3 as job J'_3 and calculate $L(J'_3)$ which is equal to 6 (see Figure 4).

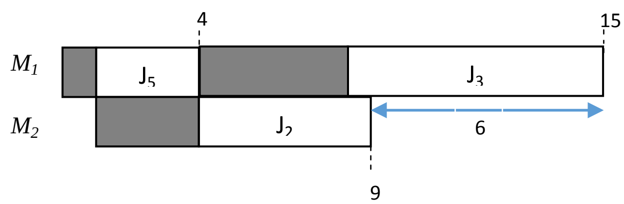

Figure 4: Adding J'_3 to the schedule.

As illustrated in Figure 5, for the fourth job, according to $L(J'_3)$, the best choice is job J_4.

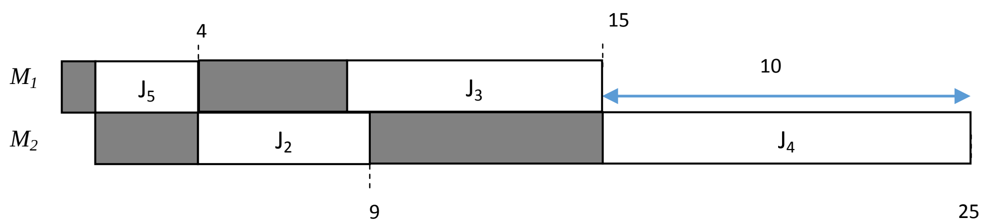

Figure 5: Adding J'_4 to the schedule.

Again, the value $L(J'_3)$ is calculated and a negative value is obtained. Therefore, we consider job J_4 as J'_4. Finally, the last job is scheduled and the final solution depicted in Figure 6 is obtained.

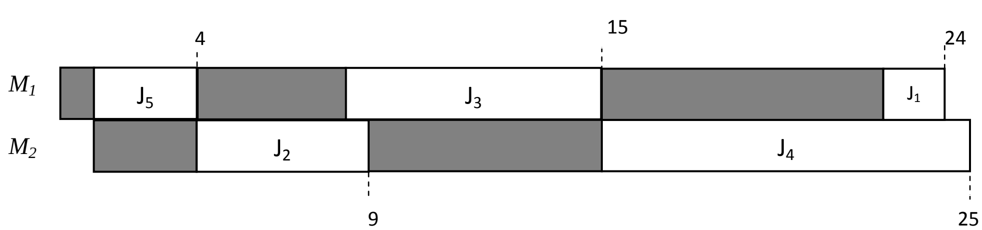

Figure 6: Adding $J_5^{'}$ to the schedule.

As it can be seen from Figure 6, the C_{max} value is 25, and the obtained schedule is optimal. The pseudo-code of Algorithm *Min-idle* is as follows.

Algorithm *Min-idle*

BEGIN

Let i = 1, *solution* = Ø;

Sort all jobs i = 1,...,n in non-increasing order of the setup times s_i and assume that { J_1 , J_2 ,…, J_n } is an ordered list of the jobs;

Let $J_1^{'} = J_n$ (the first job with the smallest setup time);

WHILE (Scheduling all jobs) DO

BEGIN

Add $J_i^{'}$ to *solution*;

Find the earliest available machine M_q and assign job $J_i^{'}$ to it;

st1 = the sum of the starting time and the setup time of the last scheduled job on the machine M_{1-q} ;
st2 = $C(M_q)$ (denoting the completion time on machine M_q);

$St = max\ (st1,st2)$, $C(M_q) = St + s(J_i^{'})+p(J_i^{'})$; (*St is the staring time of the job* $J_i^{'}$*).*

IF ($St + s(J_i^{'})) < \mathrm{C}(M_{1-q})$)

$L(J_i^{'}) = C(M_q) - C(M_{1-q})$; $L(J_i^{'})$ *is the length of the part of the job* $J_i^{'}$ *which can be overlapped with other jobs.*

found = False;

FOR k =1 to n DO

IF ($L(J_i^{'}) \geq s(J_k)$ and J_k and $J_i^{'}$ are two different jobs and J_k is an unscheduled job)

BEGIN

found = true;

Add J_k to the solution list;

Find the earliest available machine M_q and assign the job $J_i^{'}$ to it;

st1 = the sum of the starting time and the setup time of the last job on the machine M_{1-q};

$st2 = C(M_q)$, $St(J_k) = max\ (st1,st2)$, $C(M_q) = St(J_k) + s(J_k)+p(J_k)$;

```
        L( J_k ) = C(M_{1-q}) – C(M_q);
        IF (L( J'_i ) = 0)
        BEGIN
           Find an unscheduled job with the smallest setup time and consider it as J'_{i+1} , i++;
           Exit FOR;
        END
        Else IF (L( J'_i ) < 0)
        BEGIN
            IF (St + s( J'_i )) < C(M_{1-q}))
               L( J'_i ) = C(M_q) – C(M_{1-q});
             J'_{i+1} = J_k ; i++;
             Exit FOR;
         END
     END
     IF (not found)
     BEGIN
        Find an unscheduled job with the smallest setup time and consider it as J'_{i+1};
        i++;
     END
 ENDWHILE
 Cmax = max(C(M_q),C(M_{1-q}));
 Return  solution;
END.
```

The complexity of Algorithm *Min-idle* is $O(n^2)$.

3.2 Algorithm *Min-loadgap*

In this section, we describe the algorithm we propose for instances with $LB = LB_2$. In this case, we try to minimize the gap between the completion time of the loading of a job and the start of the loading of the next job. It must be noted that this algorithm cannot guarantee to achieve an optimal solution but usually the solution obtained can be close to an optimal one or even be optimal. The following example considers an instance belonging to the class $LB=LB_2$ using Algorithm *Min-loadgap*.

Example 2. Consider an instance with five jobs and the setup and processing times given as follows:

$s_1= 8$, $p_1 = 2$, $s_2 = 6$, $p_2 = 3$, $s_3 =5$, $p_3 = 3$, $s_4 = 4$, $p_4 =5$, $s_5 = 8$, $p_5 = 7$.

The lower bound is obtained as $LB_2 = 33$. It must be noted that in Algorithm *Min-loadgap* the jobs are scheduled in staggered order, i.e., job J_1' is scheduled on the first machine, job J_2' is scheduled on the second machine, job J_3' is scheduled on the first machine, and so on.

First, we choose a job with a minimal setup time and also from now, we consider a job with the minimal processing time as the last job. Thus, the job J_4 is chosen as the first job J_1', and J_1 is selected as the last job J_5'. For the second position of the sequence (schedule), a job with a loading time sufficiently small but also not smaller than p_4 has to be selected. Among the unscheduled jobs, here the best choice is job J_3. Therefore, we take J_3 as the second chosen job J_2'. The third position in the sequence is filled according to the processing time of job J_3. Thus, job J_2 will be chosen to be scheduled at the third position. Similarly, job J_5 will be chosen to be scheduled at the fourth position. Finally, job J_5' will be scheduled. It must be noted that in algorithm Min-loadgap, to accelerate the scheduling process, first the jobs are sorted in non-decreasing order of the setup times. The schedule obtained is illustrated in Figure 6.

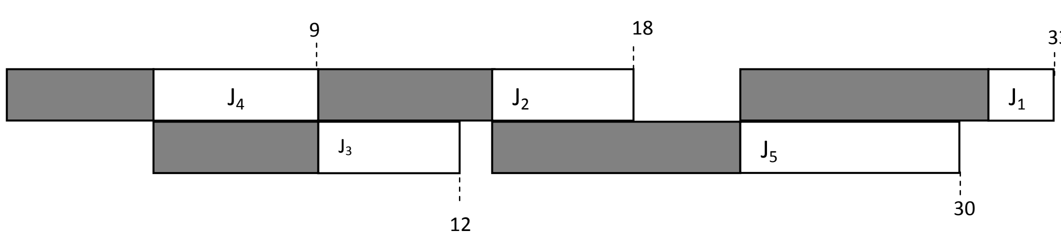

Figure 6: An optimal schedule for Example 2.

As it can be seen from Figure 6, the resulting C_{max} value is 33, and the obtained schedule is optimal. Algorithm *Min-loadgap* is as follows.

Algorithm *Min-loadgap*

BEGIN

Sort all jobs i = 1,...,n in non-decreasing order of the setup times s_i and assume that { J_1 , J_2 ,…, J_n } is the ordered list of the jobs;

Find a job with smallest processing time and let denote it as $J_n^{'}$;

Let J_1 = the first job in the ordered list. If $J_1 = J_n^{'}$, let J_1 = the second job in the ordered list;

Let q = 1; schedule the job J_1 on M_q;

Let i = 1, *solution* = ∅;

WHILE (i < n-1) DO

BEGIN

found = *false*;

For k = 0 to n DO

IF (p(J_i) ≤ s(J_k) and J_k and J_i are two different jobs and J_k is unscheduled and J_k is not the job already selected as $J_n^{'}$)

BEGIN

found = true;

Add J_k to *solution*;

$q = 1-q$, process J_k on M_q;

st1 = the sum of the starting time and the setup time of the last job on machine M_{1-q}

$st2 = C(M_q)$, $St(J_k) = max(st1,st2)$, $C(J_k) = St(J_k) + s(J_k)+p(J_k)$;

$C(M_q) = C(J_k)$, $i = k$;

Exit FOR;

END

IF (*not found*)

For k = n to 0 DO

IF(p(J_i) > s(J_k) and J_k and J_i are two different jobs and J_k is unscheduled and J_k is not the job already selected as $J_n^{'}$)

BEGIN

found = true;

Add J_k to *solution*;

q = 1-q , start the job J_k on M_q;

$st1$ = the sum of the starting time and the setup time of the last job on machine M_{1-q};

$st2 = C(M_q)$, $St(J_k) = max\ (st1,st2)$, $C(J_k) = St(J_k) + s(J_k)+p(J_k)$;

$C(M_q) = C(J_k)$, $i = k$;

Exit FOR;

END

ENDWhile

Add $J_n^{'}$ to *solution*;

q = 1-q , start the job $J_n^{'}$ on M_q; update $C(M_q)$;

$Cmax = max(C(M_q), C(M_{1-q}))$;

Return *solution*;

END.

The complexity of Algorithm *Min-loadgap* is $O(n^2)$.

4. Computational results

The data has been generated in the same way as it has been described in Hasani *et al.* (2014a). The instances were randomly generated for each server load value $L \in \{0.1, 0.5, 0.8, 1, 1.5, 1.8, 2\}$ with $p_j \overset{d}{=} U(0, 100)$. Thus, the processing times p_j are uniformly distributed in the interval (0, 100), and the setup times s_j are distributed uniformly in the interval (0, 100L). For a comparison with the results from Hasani *et al.* (2014b), the same instances were used. Thus, for $n \in \{50, 100, 200, 250, 300, 400, 500, 600, 700, 800, 900, 1000\}$, 5 instances and for $n \in \{8, 20\}$, 10 instances were generated randomly for each of the above values of L and for the additional values of n, 10 instances were also randomly generated.

Both Algorithms *Min-idle* and *Min-loadgap* have been implemented using the Java programming language. They have been run using JDK 1.3.0, with 2GB of memory available for working storage on a personal computer Intel(R) Core(TM) i5-2430M CPU @2.4GHz.

Algorithm *Min-idle* was used for the instances with $LB = LB_1$, and Algorithm *Min-loadgap* was used for the instances with $LB = LB_2$. From our experiments, it turned out that for all instances with $L \in \{0.1, 0.5, 0.8\}$, we always have $LB = LB_1$ and for all instances with $L \in \{1.5, 1.8, 2\}$, we always have $LB = LB_2$. However, for the instances with $L = 1$, both bounds have to be taken into account and therefore, if $LB = LB_1$, then only Algorithm *Min-idle* is used while for $LB = LB_2$, only Algorithm *Min-loadgap* is used.

The computational results with Algorithms *Min-idle* and *Min-loadgap* for small-sized, medium-sized and large-sized instances are given in Table 1 and Figure 7, and the results for very large-sized instances with up to 10000 jobs are given in Table 2. In column 1, the maximum run times are given in seconds, in column 2, the number n of jobs is given, and in columns 3 - 10, the values R_{avg} denoting the average value of the relation $\frac{C\max}{LB}$ and R_{max} denoting the maximum value of the relation $\frac{C\max}{LB}$ among all instances for a particular value of L are given. Finally, in the last column, the percentage (LB%) of instances is given, where the particular algorithm has obtained the lower bound.

T_{max}	n	L	0.1	0.5	0.8	1	1.5	1.8	2	LB%
0.001s	8	R_{avg}	1.063851	1.083760	1.103816	1.079312	1.034216	1.032276	1.019124	21.5
		R_{max}	1.162436	1.144475	1.219753	1.173033	1.097826	1.180722	1.101361	
0.002s	20	R_{avg}	1.022241	1.035102	1.067376	1.080847	1.032863	1.020372	1.016807	21.5
		R_{max}	1.049073	1.060932	1.125754	1.141383	1.138258	1.045398	1.044609	
0.002s	50	R_{avg}	1.012915	1.019622	1.039196	1.037914	1.012539	1.004642	1.002518	20
		R_{max}	1.022812	1.032048	1.081148	1.074102	1.046256	1.014442	1.012020	
0.003s	100	R_{avg}	1.008932	1.006095	1.019904	1.032133	1.006771	1.007497	1.004851	17.1
		R_{max}	1.014008	1.011242	1.034018	1.051858	1.021720	1.023131	1.015941	
0.004s	200	R_{avg}	1.002223	1.003501	1.002449	1.022286	1.003083	1.002772	1.000332	17.1
		R_{max}	1.004395	1.004975	1.003771	1.029082	1.008698	1.004708	1.001662	
0.007s	250	R_{avg}	1.001274	1.003112	1.012326	1.019115	1.001843	1.002747	1.000195	20.0
		R_{max}	1.003335	1.006223	1.035282	1.023918	1.003279	1.005391	1.000973	
0.012s	300	R_{avg}	1.002818	1.001335	1.008061	1.019074	1.001949	1.001801	1.001060	2.9
		R_{max}	1.004806	1.002209	1.019244	1.032481	1.005118	1.002465	1.002357	
0.025s	400	R_{avg}	1.001723	1.002621	1.005480	1.018065	1.001689	1.000017	1.000326	28.6
		R_{max}	1.004118	1.003473	1.009137	1.025457	1.006669	1.000085	1.001139	
0.048s	500	R_{avg}	1.001608	1.001492	1.006849	1.019009	1.000472	1.000312	1.000231	22.9
		R_{max}	1.003123	1.002546	1.011298	1.025995	1.001485	1.000987	1.001153	
0.084s	600	R_{avg}	1.001714	1.001373	1.004262	1.010358	1.001051	1.000683	1.000758	22.9
		R_{max}	1.002384	1.001833	1.008633	1.013592	1.003919	1.003322	1.001622	
0.122s	700	R_{avg}	1.000748	1.000685	1.002555	1.010412	1.000996	1.000000	1.000233	28.6
		R_{max}	1.001239	1.001107	1.006718	1.026745	1.001552	1.000000	1.000726	
0.182s	800	R_{avg}	1.001087	1.001179	1.002329	1.014871	1.000172	1.000759	1.000235	31.4
		R_{max}	1.001886	1.001577	1.004340	1.022341	1.000858	1.002482	1.001174	
0.254s	900	R_{avg}	1.000772	1.000551	1.001911	1.008458	1.000401	1.000034	1.000000	31.4
		R_{max}	1.001178	1.000747	1.003417	1.018013	1.000619	1.000171	1.000000	
0.350s	1000	R_{avg}	1.000807	1.000566	1.001989	1.010157	1.000777	1.000414	1.000146	14.3
		R_{max}	1.001517	1.000894	1.003356	1.016946	1.002464	1.001269	1.000348	

Table 1. Computational results with Algorithms *Min-idle* and *Min-loadgap* for small, medium and large instances.

Comparing the obtained results for Algorithms *Min-idle* and *Min-loadgap* with the results in Hasani *et al.* (2014b) for the simulated annealing algorithm (SA) and the genetic algorithm (GA) and using the same instances, the following summary can be given:

As it can be seen from Figure 7, for the instances with $n \leq 400$, the heuristics proposed in Hasani *et al.* (2014b) are quite superior. However, the better results are only obtained within a time limit much larger than for the methods proposed in this paper. Despite the weakness of our methods dealing with small-sized problems, one can use the solution obtained from our algorithms as an initial solution in any heuristic to enhance the efficiency of the heuristic and also to increase the chance of finding a schedule close to an optimal one within a shorter time. In fact, the efficiency of our algorithms is beheld for larger instances, where the best known heuristics are rather weak and in some cases unable to get acceptable results within a reasonable time.

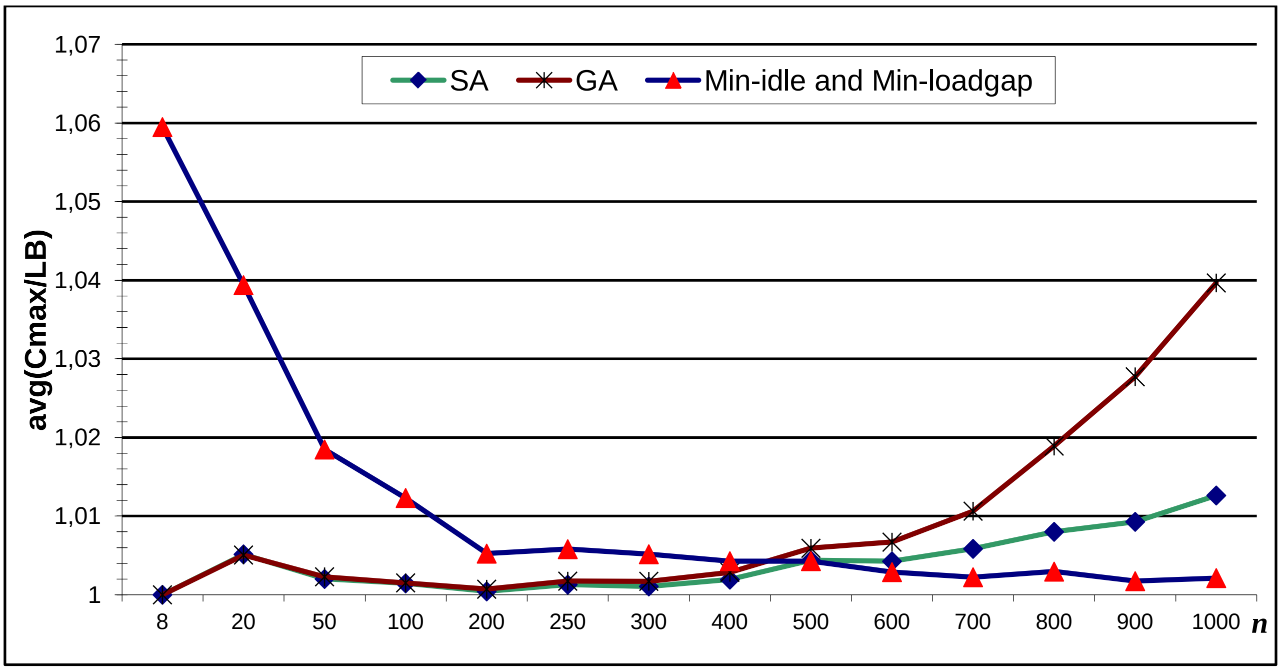

Figure 7. Comparison of the average value of the relation Cmax/LB of Algorithms *Min-idle* and *Min-loadgap* with Algorithms SA and GA.

In the following, we compare the performance of our proposed algorithms on the larger instances mentioned in Hasani *et al.* (2014b).

(1) For n = 500, the maximal value of the relation $\dfrac{C\max}{LB}$ is 1.02599 and the average value of the relation $\dfrac{C\max}{LB}$ is 1.004281, while in Hasani *et al.* (2014b), for Algorithm SA the

maximum value of the relation $\frac{C\max}{LB}$ is 1.02494 and the average value of the relation $\frac{C\max}{LB}$ is 1.004372. However, in Hasani *et al.* (2014b), Algorithm SA reached the lower bound for 28.6% of the instances, while our experiments have shown that the proposed algorithms reached the lower bound for about 22.9% of the instances and could solve the instances within no more than 0.048 seconds. However, in Hasani *et al.* (2014b), the results were obtained only in within no more than 3600 seconds.

(2) For n = 600, the maximal value of the relation $\frac{C\max}{LB}$ is 1.013592 and the average value of the relation $\frac{C\max}{LB}$ is 1.002885, while in Hasani *et al.* (2014b), for Algorithm SA the maximum value of the relation $\frac{C\max}{LB}$ is 1.023348 and the average value of the relation $\frac{C\max}{LB}$ is 1.004253. Moreover, in Hasani *et al.* (2014b), Algorithm SA reached the lower bound for 17% of the instances, while our experiments have shown that the proposed algorithms reached the lower bound for about 22.9% of the instances and could solve the instances at within at most 0.084 seconds. However, in Hasani *et al.* (2014b), the results were obtained within no more than 3600 seconds.

(3) For n = 700, the maximal value of the relation $\frac{C\max}{LB}$ is 1.026745 and the average value of the relation $\frac{C\max}{LB}$ is 1.002232, while in Hasani *et al.* (2014b), for Algorithm SA the maximum value of the relation $\frac{C\max}{LB}$ is 1.031786 and the average value of the relation $\frac{C\max}{LB}$ is 1.005844. Moreover, in Hasani *et al.* (2014b), Algorithm SA reached the lower bound for 5.7% and Algorithm GA for 8.6% of the instances, while our experiments have shown that the proposed algorithms reached the lower bound for about 28.6% of the instances and could solve the instances within at most 0.122 seconds. However, in Hasani *et al.* (2014b), the results were obtained only within no more than 3600 seconds.

(4) For n = 800, the maximal value of the relation $\frac{C\max}{LB}$ is 1.022341 and the average value of the relation $\frac{C\max}{LB}$ is 1.002947, while in Hasani *et al.* (2014b), for Algorithm SA the maximum value of the relation $\frac{C\max}{LB}$ is 1.037611 and the average value of the relation $\frac{C\max}{LB}$ is 1.008009. Moreover, in Hasani *et al.* (2014b), Algorithm SA reached the lower bound for 0% and Algorithm GA for 5.7% of the instances, while our experiments have shown that the proposed algorithms reached the lower bound for about 31.4% of the instances and could solve the instances within no more than 0.182 seconds. However, in Hasani *et al.* (2014b), the results were obtained only within no more than 3600 seconds.

(5) For n = 900, the maximal value of the relation $\frac{C\max}{LB}$ is 1.018013 and the average value of the relation $\frac{C\max}{LB}$ is 1.001733, while in Hasani *et al.* (2014b), for Algorithm SA the maximum value of the relation $\frac{C\max}{LB}$ is 1.044494 and the average value of the relation $\frac{C\max}{LB}$ is 1.009279. Moreover, in Hasani *et al.* (2014b), Algorithm SA reached the lower bound for 0% and Algorithm GA for 2.9% of the instances, while our experiments have shown that the proposed algorithms reached the lower bound for about 31.4% of the instances and could solve the instances within no more than 0.254 seconds. However, in Hasani *et al.* (2014b), the results were obtained only within no more than 3600 seconds.

(6) For n = 1000, the maximal value of the relation $\frac{C\max}{LB}$ is 1.016956 and the average value of the relation $\frac{C\max}{LB}$ is 1.002125, while in Hasani *et al.* (2014b), for Algorithm SA the maximum value of the relation $\frac{C\max}{LB}$ is 1.06515 and the average value of the relation $\frac{C\max}{LB}$ is 1.012628. Moreover, in Hasani *et al.* (2014b), none of the algorithms could reach the lower bound for some instance, while our experiments have shown that the

proposed algorithms reached the lower bound for about 14.3% of the instances and could solve the instances within no more than 0.35 seconds. However, in Hasani *et al.* (2014b), the results were obtained only within no more than 3600 seconds.

In the following, the computational results with Algorithms *Min-idle* and *Min-loadgap* for very large instances are presented in Table 2. Here no computational results are available for other algorithms in the literature.

It can be noted that even for the largest instances with n = 10000, the relation $\frac{C\max}{LB}$ is only 1.006228 and the average value of the relation $\frac{C\max}{LB}$ is even 1.00077. The proposed algorithms are extremely fast and in the worst case for n = 10000, the required run time is no more than 371.3 seconds. The obtained C_{max} values for all instances are very close to the lower bound and, on average, the lower bound has been reached for about 31.1% of all very large instances.

T_{max}	N	L	0.1	0.5	0.8	1	1.5	1.8	2	LB%
1.2s	1500	R_{avg}	1.000629	1.000535	1.002319	1.008121	1.000143	1.000089	1.000061	30.0
		R_{max}	1.001070	1.001113	1.004509	1.013470	1.000809	1.000703	1.000561	
2.8s	2000	R_{avg}	1.000506	1.000396	1.001998	1.006908	1.000325	1.000109	1.000010	27.1
		R_{max}	1.000782	1.000807	1.004186	1.010141	1.001952	1.000452	1.000079	
9.6s	3000	R_{avg}	1.000359	1.000220	1.001611	1.004867	1.000087	1.000053	1.000009	34.3
		R_{max}	1.000689	1.000349	1.004081	1.008649	1.000416	1.000322	1.000086	
22.0s	4000	R_{avg}	1.000354	1.000287	1.001401	1.005392	1.000009	1.000000	1.000052	34.3
		R_{max}	1.000598	1.000514	1.003417	1.009914	1.000089	1.000000	1.000198	
41.6s	5000	R_{avg}	1.000301	1.000153	1.001249	1.004787	1.000003	1.000032	1.000010	30.0
		R_{max}	1.000449	1.000250	1.002739	1.008070	1.000029	1.000144	1.000055	
74.3s	6000	R_{avg}	1.000240	1.000166	1.001266	1.002965	1.000038	1.000018	1.000012	27.1
		R_{max}	1.000414	1.000263	1.003887	1.006454	1.000209	1.000072	1.000081	
113.8	7000	R_{avg}	1.000191	1.000195	1.001460	1.003339	1.000004	1.000018	1.000008	34.3
		R_{max}	1.000234	1.000335	1.003295	1.007223	1.000036	1.000090	1.000048	
169.5s	8000	R_{avg}	1.000238	1.000148	1.000821	1.003552	1.000030	1.000010	1.000018	27.1
		R_{max}	1.000335	1.000245	1.002562	1.007418	1.000108	1.000064	1.000118	
277.2s	9000	R_{avg}	1.000217	1.000134	1.000686	1.003270	1.000011	1.000019	1.000000	35.7
		R_{max}	1.000336	1.000258	1.001635	1.004311	1.000112	1.000079	1.000000	
371.3s	10000	R_{avg}	1.000202	1.000128	1.000481	1.002413	1.000015	1.000007	1.000010	31.4
		R_{max}	1.000273	1.000225	1.000726	1.004933	1.000096	1.000071	1.000068	

Table 2. Computational results with Algorithms *Min-idle* and *Min-loadgap* for very large instances.

In Figure 8, the average values of the relation $\frac{C\max}{LB}$ in dependence on the number of jobs are presented (the average value is taken over all instances with all values L considered).

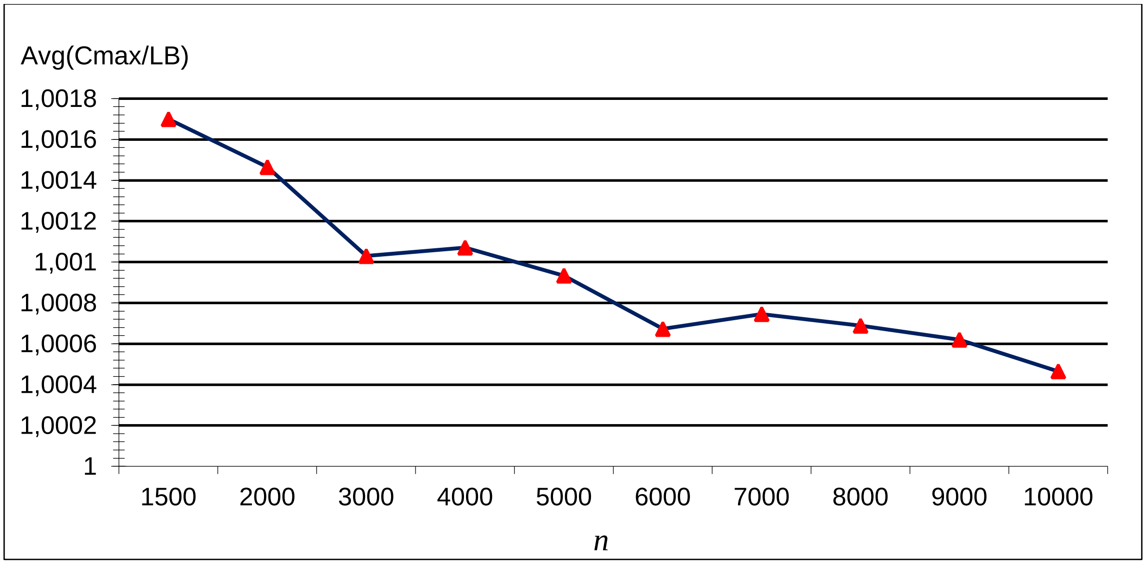

Figure 8. Average values of the relation of Cmax/LB for all instances over all values of L from 0.1 to 2.

The interesting point, which becomes clear from Figure 8, is that, by increasing the number of jobs, the average values of the relation $\frac{C\max}{LB}$ are decreasing. As an exception, this value becomes only slightly larger from n = 3000 to n = 4000 and also from n = 6000 to n = 7000. However, in general, there is a clear decreasing trend for this relation. Often an increase of the number of jobs is expected to reduce the quality of the results. However, our experiments have shown that, for instances with more than 10000 jobs, even the deviations of the C_{max} value from the lower bound are decreasing further.

5. Concluding remarks

In this paper, the problem $P2, S1 || C_{\max}$ was considered. Two algorithms were proposed to solve very large instances. The first algorithm was developed only for instances with $LB = LB_1$, and the second one for those instances with $LB = LB_2$. The instances were generated in the same way as in previous works so that the results can be compared with those existing in the recent literature. However, while in Hasani *et al.* (2014a), only instances with up to 250 jobs and in Hasani *et al.* (2014b), only instances with up to 1000 jobs have been considered, here algorithms have been compared on much larger instances with up to 10000 jobs.

The algorithms performed extremely well in terms of the deviation from the known lower bounds and also in terms of the low computational times for large and very large instances. They outperformed the results in the only existing literature for large instances within only some seconds for the largest instances. However, for small-sized and medium-sized instances, our obtained results were not superior. Nevertheless, our preliminary experiments revealed that, using the schedules obtained by our algorithms as initial solutions in a metaheuristic algorithm can contribute to a significant superiority. In addition, even for very large instances, one can continue the optimization process with another heuristic to get further improvements.

The strategy of sequencing the jobs in such a way that the machine idle times and server waiting times are to be minimized turned out to be essential for the high quality of our algorithms.

For future work, it is intended to investigate several types of heuristics for the single server problem with an arbitrary number of machines and minimizing the makespan and for the two-machine problem with a single server and a sum objective function such as total completion time or total tardiness.

References

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

Abdekhodaee, A., Wirth, A. & Gan, H.S., 2006. Scheduling two parallel machines with a single server: The general case. *Computers & Operations Research* 33, 994-1009.

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

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

Gan, H.S., Wirth, A. & Abdekhodaee, A., 2012. A branch-and-price algorithm for the general case of scheduling parallel machines with a single server. *Computers & Operations Research* 39, 2242-2247.

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

Hasani, K., Kravchenko, S.A. & Werner, F., 2014a. Block models for scheduling jobs on two parallel machines with a single server. *Computers & Operations Research,* 41 (94-97).

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

Huang, S., Cai, L. & Zhang, X., 2010. Parallel dedicated machine scheduling problem with sequence-dependent setups and a single server. *Computers & Industrial Engineering*, 58 (1), 165-174.

Kim, M.-Y. & Lee, Y.H., 2012. MIP models and hybrid algorithm for minimizing the makespan of parallel machines scheduling problem with a single server. *Computers & Operations Research*, 39, 2457-2468.

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

Werner, F. & Kravchenko, S.A., 2010. Scheduling with multiple servers. *Automation and Remote Control*, 71, 2109-21.

Zhang, L. & Wirth, A., 2009. On-line scheduling of two parallel machines with a single server. *Computers & Operations Research* 36, 1529 - 1553.