Minimizing Mean Flow Time for the Two-Machine Scheduling Problem with a Single Server

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

[1]Islamic Azad University, Malayer Branch, Malayer, Iran
keramat@newman.bas-net.by

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

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

July 12, 2013

Abstract

In this paper, we consider the problem of scheduling a set of jobs on two parallel machines to minimize the sum of completion times. Each job requires a setup which must be done by a single server. It is known that this problem is strongly NP-hard. We propose two mixed integer linear programming models and a simulated annealing algorithm. The performance of these algorithms is evaluated for instances with up to 250 jobs.

Keywords: Scheduling, Parallel machines, Single server, Mixed integer linear programming, Simulated annealing

1. Introduction

The problem considered in this paper can be described as follows. There are *n* independent jobs and two identical parallel machines. For each job *j*, *j = 1*, …, *n*, its processing time p_j is known. Denote by $p_{\max}$ the maximal value of the processing times, i.e., $p_{\max} = \max\{p_j \mid j = 1,\ldots,n\}$. Before processing, a job must be loaded on the same machine M_q, $q = 1,2$, where it is processed, which requires a known setup time s_j. During such a setup, the machine M_q is also involved into this process for s_j time units, i.e., no other job can be processed on this machine during this setup. All setups have to be done by a single server which can handle at most one job at a time. The objective is to determine a feasible schedule which minimizes the sum of completion times. So, using the common scheduling notation, we consider the problem $P2,S1||\sum C_j$. This problem is unary NP-hard, since it is known that problem $P2,S1|s_j = s|\sum C_j$ is unary NP-hard, see Hall et al. (2000). Some special cases of this problem

were already considered. The problem $P2,S1|p_j = p|\sum C_j$ is binary NP-hard, see Brucker et al. (2002). There is a polynomial algorithm for the problem $P2,S1|s_j = 1|\sum C_j$, see Hall et al. (2000). For the problem $P3,S1|s_j = 1|\sum C_j$, Brucker et al. (2002) developed a polynomial algorithm with the complexity $O(n^7)$. For the problem $P,S1|s_j = 1|\sum C_j$, Kravchenko and Werner (2001) proposed an algorithm which creates a schedule with the following estimation:

$$\sum_{i=1}^{n} \widetilde{C}_i - \sum_{i=1}^{n} C_i^* \le n'(m-2),$$

where $n' = |\{i \mid p_i < m-1\}|$. In Wang and Cheng (2001), a $(5-\frac{1}{m})$- approximation algorithm was proposed for the problem $P,S1||\sum w_j C_j$, and it was shown that the SPT schedule is a $\frac{3}{2}$-approximation for the problem $P,S1|s_j = s|\sum C_j$.

For the problem $P2,S1||\sum C_j$, nothing is known about heuristic algorithms. However, some results for close models are known. In Weng et al. (2001), the problem of scheduling a set of independent jobs on unrelated parallel machines with job sequence dependent setup times and the minimization of weighted mean completion time was considered. Seven heuristic algorithms were presented and compared with each other and the best algorithm was selected. The heuristics were tested for up to 120 jobs and 12 machines. In Dunstall and Wirth (2005), the problem with identical parallel machines, jobs divided into families and sequence-independent set-up times was considered. The objective was to minimize the weighted sum of completion times. Several heuristics were proposed for instances with up to 8 families, 25 jobs and 5 machines. In Azizoglu and Webster (2003), several branch-and-bound algorithms for the identical parallel machine scheduling problem with family set-up times and the objective of minimizing total weighted flow time were presented. They applied their methods to problems with up to 25 jobs, 8 families, and 5 machines. In Guirchoun et al. (2005), a parallel machine scheduling problem with a server was considered, however, the general requirements that loading requires the server and the machine were omitted. Thus, the considered problem was modelled as a two-stage hybrid flow shop with no-wait constraint between the two stages. A mathematical formulation for the problem was proposed and some polynomially solvable special cases were considered.

In this paper, we consider the problem $P2,S1||\sum C_j$ and develop two mixed integer linear programming models and a simulated annealing algorithm. We apply these methods to instances with up to 250 jobs.

2. Preliminary results

One can see that an optimal schedule for the problem $P2,S1||\sum C_j$ can be found in the class of list schedules, i.e., one can consider a sequence of the jobs, where all jobs are placed into a list and the algorithm schedules the first unscheduled job from the list whenever a machine becomes idle.

Example 1. We consider an instance with five jobs and the setup and processing times given in Table 1. Let the current job sequence be $\pi_0 = (3, 1, 4, 2, 5)$.

j	**1**	**2**	**3**	**4**	**5**
s_j	2	2	1	2	1
p_j	4	3	5	4	2

Table1. Setup and processing times of the jobs in Example 1.

According to the given sequence, we assign the jobs in a greedy manner to the machines. In this way, we get the job sequence (3, 4, 5) on machine M_1 and the job sequence (1, 2) on machine M_2. Thus, we get the schedule described in Figure 1, where the grey parts represent the setups. The total completion time value is 53.

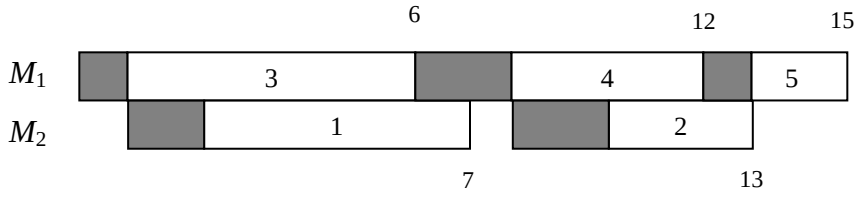

Figure 1: A schedule with five jobs resulting from π_0.

Denote by L_j the sum $s_j + p_j$ for $j = 1,\ldots,n$. Further, we will suppose that all jobs are ordered in such a way that $L_1 \le \ldots \le L_n$ holds.

Statement 1. For the list schedule corresponding to the sequence $\{1,\ldots,n\}$, where $L_1 \le \ldots \le L_n$, inequality

$$\frac{\sum \tilde{C}_j}{\sum C_j^*} \le 2$$

holds. Here $\sum \tilde{C}_j$ is the sum of the completion times for the list schedule corresponding to the sequence $\{1,\ldots,n\}$, and $\sum C_j^*$ is the sum of the completion times for an optimal schedule.

Proof: It is straightforward to see that

$$\tilde{C}_j \le \sum_{k=1}^{j-1} s_k + \frac{1}{2}\sum_{k=1}^{j-1} p_k + L_j = \frac{1}{2}\sum_{k=1}^{j-1} s_k + \frac{1}{2}\sum_{k=1}^{j-1} L_k + L_j$$

holds. Since

$$\frac{1}{2}\sum_{k=1}^{j-1} s_k + \frac{1}{2}L_j \le \frac{1}{2}\sum_{k=1}^{j} L_k \le C_j^* \quad \text{and} \quad \frac{1}{2}\sum_{k=1}^{j-1} L_k + \frac{1}{2}L_j \le C_j^*,$$

We obtain $\tilde{C}_j \le 2C_j^*$. ■

Further, we will use two lower bounds:
For the first lower bound LB_1 we have $\sum_j C_j^1 = LB_1$, where

$$C_j^1 = L_j + L_{j-2} + \dots .$$

For the second lower bound LB_2 we have $\sum_j C_j^2 = LB_2$, where

$$C_j^2 = L_j + ss_{j-1} + ss_{j-2} + \dots ,$$

and $ss_1, \dots, ss_n$ are the setup times ordered according to a non-decreasing order of their values, i.e., $ss_1 \le \dots \le ss_n$.

2. M1 model

The **M1** model is constructed to find an optimal schedule for the problem $P2, S1 || \sum C_j$, and an optimal schedule belongs to the class of list schedules. Suppose one has a schedule s defined by a list π. Let ft_j be the time of completing job j, and let st_j denote the starting time of loading the job j. Moreover, let mt_j denote the completion time of loading the job j, and let $O_{i,j}$ be a binary variable which defines the order of the jobs i and j in the list π, i.e.,

$$O_{i,j} = \begin{cases} 1 & if \quad st_i \le st_j \\ 0 & otherwise \end{cases}.$$

Let $p_{i,j} = 1$ if $mt_i \le st_j \le ft_i$, i.e., job j is started after starting but before finishing the processing of job i. Now, the minimization of $\sum C_j$ is equivalent to the minimization of $\sum_j ft_j$. Since $O_{i,j}$ defines the order of the jobs i and j in the list π,

$$O_{i,j} + O_{j,i} = 1, \quad i \ne j$$

must hold. Both inequalities

$$ft_j - st_j \ge s_j + p_j \quad \text{and} \quad mt_i - st_j \ge s_j$$

are satisfied by definition. Since there are only two machines, the inequality $\sum_{i=1}^{n} p_{i,j} \le 1$ holds.

Let P be a large number. We use

$$P = \frac{\sum_{i=1}^{n} p_i - p_{\max}}{2} + \sum_{i=1}^{n} s_i + p_{\max} .$$

Now, if job j is before job i in the list π, then $st_j - mt_i \ge 0$. However, if job j is after job i in the list π, then

$$st_j - mt_i \ge (1 - p_{i,j})\, p_i ,$$

i.e., if the job j does not overlap with job i, then $st_j - mt_i \ge p_i$ holds. Thus, in general we obtain the inequality

$$st_j - mt_i \ge (O_{i,j} - p_{i,j})\, p_i - O_{j,i} P .$$

One can see that, if $p_{i,j} = 1$, i.e., job j overlaps with job i, then $st_j - mt_i \le p_i$ holds. It follows that the inequality

$$st_j - mt_i \le p_{i,j}\, p_i + (1 - p_{i,j}) P \text{ holds.}$$

Finally, we obtain the model

$$\sum_j ft_j \to \min$$

$$\text{s.t.} \qquad O_{i,j} + O_{j,i} = 1, \quad i \neq j$$

$$ft_j - st_j \geq s_j + p_j$$

$$mt_j - st_j \geq s_j$$

$$\sum_{i=1}^{n} p_{i,j} \leq 1$$

$$st_j - mt_i \geq (O_{i,j} - p_{i,j})\, p_i - O_{j,i} P$$

$$\sum_j ft_j \geq LB$$

Here $O_{i,j}$ and $p_{i,j}$ are binary variables; ft_i, st_i, and mt_i are positive variables; p_j and s_j are the processing and setup times known in advance, $i = 1,\ldots,n$ and $j = 1,\ldots,n$, P is a constant number and $LB = \max\{LB_1, LB_2\}$.

3. M2 model

The **M2** model is constructed to find an optimal schedule, where all jobs from the list π have to be scheduled in a staggered order, i.e., taking the new job from the list, the algorithm has to change the machine. We observed that the M2 model cannot find an optimal schedule for the problem $P2, S1 \| \sum C_j$, but the schedules produced by this model appear to be rather close to the lower bound.

Next, we describe the M2 model. Suppose one has a staggered schedule s defined by a list π. Let the variable $N_{i,j}$ define the position of the job j in the list π, i.e.,

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

Let the variable ft_i be the time of completing the i-th job j in the list π. Let st_i be the starting time of loading the i-th job j in the list π, and let mt_i be the completion time of loading the i-th job j in the list π. Then, the minimization of $\sum C_j$ is equivalent to the minimization of $\sum_i ft_i$. Since each position in the list π can be occupied by only one job, the equality $\sum_i N_{i,j} = 1$ holds. Since each job has to be placed at some position in the list π, the equality $\sum_j N_{i,j} = 1$ holds. The inequality

$$ft_i - st_i \geq \sum_j N_{i,j} (s_j + p_j)$$

holds since between the starting time st_i and the finishing time ft_i of the i-th job j, one has to load and to process the job j. The inequality

$$mt_i - st_i \geq \sum_j N_{i,j} s_j$$

holds since between the starting time st_i and the finishing time mt_i of the i-th job j, one has to load the job j. The inequality $st_i \geq mt_{i-1}$ holds since the list π defines the order for loading all the jobs. The inequality $st_i \geq ft_{i-2}$ holds since we consider schedules where all jobs are processed in a staggered order.
Thus, the model M2 can be described in the following way:

$$\sum_i ft_i \to \min$$

$$\text{s.t.} \quad \sum_i N_{i,j} = 1$$

$$\sum_j N_{i,j} = 1$$

$$ft_i - st_i \geq \sum_j N_{i,j}(s_j + p_j)$$

$$mt_i - st_i \geq \sum_j N_{i,j} s_j$$

$$st_i \geq mt_{i-1}$$

$$st_i \geq ft_{i-2}.$$

Here $N_{i,j}$ are binary variables; ft_i, st_i, and mt_i are positive variables; p_j and s_j are the processing and setup times known in advance; $i = 1,\ldots,n$ and $j = 1,\ldots,n$.

4. Simulated annealing algorithm

This section presents a simulated annealing algorithm to minimize total completion time for the two-machine scheduling problem with a single server. Simulated annealing tries to avoid cycling by randomization and simulates an annealing process in physics, see e.g. Kirkpatrick et al. (1983). In any iteration, a neighbor is determined by means of random decisions. In the case when the generated neighbor has a better objective function value than the starting solution, the neighbor is always accepted as the new starting solution while in the case of a worse neighbor, this solution is only with a certain probability accepted. For the quality of the results by a simulated annealing algorithm, the chosen neighborhood and the cooling scheme applied are important. Below we give more details.

Neighborhood

The definition of an appropriate neighborhood for a current scheduling solution has usually a large influence on the quality of the final solution. The algorithm presented later is based on the generation of a neighbor in a specific neighborhood. For permutation problems, one can use e.g. the following operators for generating a neighbor:

- **Swap operator**: Here two randomly selected jobs are swapped. Given π_0 in Example 1 and assume that the two randomly selected positions $a = 3$ and $b = 5$, we obtain the sequence

  $Swap(\pi_0, a, b) = Swap(\pi_0, 3, 5) = (3, 1, 5, 2, 4)$.
- **Swap two adjacent positions:** Here two randomly selected adjacent jobs are swapped. Given π_0 in Example 1 and assume that the two randomly selected positions $a = 3$ and $b = 4$, we obtain the sequence

  $SwapAdj(\pi_0, a, b) = SwapAdj(\pi_0, 3, 4) = (3, 1, 2, 4, 5)$.
- **Swap Block operator**: Here a randomly selected block of ℓ jobs is swapped with another randomly selected block of ℓ jobs as well. The block length ℓ is a randomly selected integer from the set $\{2, 3, \ldots \left\lfloor \frac{n}{2} \right\rfloor\}$. If the block length ℓ has been chosen, we randomly determine a position a of the first job of the first block and a position b of the first job of the second block. Here we have $a + \ell \leq b \leq n - \ell + 1$. Given the sequence π_0 in Example 1 and assume that first the block length $\ell = 2$ and then the positions $a = 2$ and $b = 4$ have been selected. Then we obtain the sequence

  $SwapBlock(\pi_0, a, b, \ell) = SwapBlock(\pi_0, 2, 4, 2) = (3, 2, 5, 1, 4)$.
- **Insert operator**: Here one randomly selected job is removed from its position a and is put on a randomly determined new position b. Given π_0 in Example 1 and assume that the two randomly selected positions are $a = 2$ and $b = 4$, we obtain the sequence

  $Insert(\pi_0, a, b) = Insert(\pi_0, 2, 4) = (3, 4, 2, 1, 5)$.
- **Insert Block operator**: Here one randomly selected block is removed from its position a and it is put on a randomly determined new position b. The block length ℓ is a randomly selected integer from the set $\{2, 3, \ldots n - b\}$. If the block length ℓ has been chosen, we randomly determine two positions a and b. Here we have $b + \ell \leq n$. Given the sequence π_0 in Example 1 and assume that first the block length $\ell = 2$ and then the positions $a = 2$ and $b = 4$ have been selected. Then we obtain the sequence

  $InsertBlock(\pi_0, a, b, \ell) = InsertBlock(\pi_0, 2, 4, 2) = (3, 2, 5, 1, 4)$.
- **Reverse Block operator**: Here a part of the sequence with length ℓ is reversed. Given the sequence π_0 in Example 1 and assume that the block length $\ell = 3$ and then the position $a = 2$ have been selected. Then we obtain the sequence

  $ReverseBlock(\pi_0, a, \ell) = ReverseBlock(\pi_0, 2, 3) = (3, 2, 4, 1, 5)$.
- **Insert Block and reverse operator:** Here after applying insert the block operator, the inserted block has been inversed. Given the sequence π_0 in Example 1 and assume that first the block length $\ell = 2$ and then the positions $a = 2$ and $b = 4$ have been selected. Then we obtain the sequence

  $InsertBlock(\pi_0, a, b, \ell) = InsertBlock(\pi_0, 2, 4, 2) = (3, 2, 5, 4, 1)$.

In any iteration of our simulated annealing algorithm, each of these seven operators is used and seven neighbors are randomly generated. Then the neighbor with the best makespan value among them is taken as the generated neighbor and compared with the current starting solution by the simulated annealing acceptance criterion. We have found that the composite neighborhood worked better than each of the single neighborhoods.

Cooling scheme

Typical cooling schemes used in a simulated annealing algorithm are a geometric, an exponential, a Lundy-Mees and a linear reduction scheme. We tested some different cooling schemes for the problem under consideration and found that often the geometric scheme is slightly superior. In many other applications to scheduling problems, a geometric cooling scheme is also preferred. Therefore, in the following we test exclusively geometric schemes. The geometric cooling scheme reduces the current temperature to the new temperature in the next epoch according to

$$T_k = \alpha\, T_{k-1},\ k = 0,1,2,\ldots$$

where $0 < \alpha < 1$. We have found that the initial temperature should be chosen such that about 25 percent of worse solutions should be accepted at the beginning. For the problem under consideration, we have chosen the initial temperature $T = 15$. On the other hand, the final temperature should be low enough so that worse solutions do not longer replace solutions with better objective function values. Moreover, in our experiments it turned out that the value α = 0.999 worked good. We have chosen an epoch length of 1, i.e., after each iteration the temperature is reduced. Alternatively, only after a period with constant temperature, one may reduce the temperature. In fact, updating the temperature must be done in such a way that, when the defined run time limit denoted by *TL* is going to be finished, the temperature becomes very close to zero. According to the geometric cooling scheme, at the final stage of the algorithm $T_N = \alpha^N T_0$. Therefore, we have

$$N = \log_{\alpha} \frac{T_N}{T_0}.$$

With the given value T_N = 0.0005, the algorithm will be finished after N = 10304 iterations and T will be updated every TL/10304 seconds.

Algorithm

In our simulated annealing algorithm, we used a randomly generated starting solution. As a stopping criterion, we used the first of the following events:

- a maximal run time limit, or
- within the last 2000 iterations, no improvement of the best objective function value was obtained, or
- for the currently best sum of completion time value *BestSumC*, the inequality *BestSumC* – *LB* < 1 holds.

The complete simulated annealing algorithm is given below as Algorithm 1, where *Rand*(0,1) denotes a uniformly distributed random number form the interval (0,1).

Algorithm 1. Simulated Annealing (SA)

```
BEGIN
  Generate an initial feasible solution X and determine the objective function value SumC(π);
  BestSol := π; BestSumC := SumC (π)
  T := initial temperature;
  WHILE (stopping criterion is not met) DO
     π' := best neighbor among the generated neighbors of π;
     ΔC := SumC (π') – SumC (π);
     prob := Rand (0,1);
     IF ((ΔC ≤ 0) or (prob < e^(-ΔC/T))) THEN
        π := π'; SumC (π) := SumC (π');
        IF (SumC (π) < BestSumC) THEN
           BestSumC := SumC (π); BestSol := π;
        END IF
     END IF
    Update T according to the chosen cooling scheme;
  END WHILE
  Output BestSol together with its SumC value;
END.
```

5 Computational Results

The performance of the proposed models and the heuristic SA has been tested on the data generated in the same way as it was described in Hasani et al. (2013). Moreover, for comparison purposes, we used the same run time limit of (300/8)n seconds for the instances with $n \in \{8, 20, 50\}$ and 3600 seconds for the other instances.

For $n \in \{8, 20\}$, the data sets were generated for server load values ranging between 0.1 and 2 with 0.1 increments, i.e., for each $L \in \{0.1, 0.2, \ldots, 2\}$, the value s_j is distributed uniformly in $(0, 100L)$. For each value of L, 10 instances were randomly generated with $p_j \overset{d}{=} \cup(0, 100)$, i.e., p_j is uniformly distributed in the interval (0, 100).

For $n \in \{50, 100, 200, 250\}$, 5 instances were generated for each $L \in \{0.1, 0.5, 0.8, 1, 1.5, 1.8, 2\}$ with

$$p_j \overset{d}{=} \cup(0, 100) \quad \text{and} \quad s_j \overset{d}{=} \cup(0, 100L).$$

The algorithms have been implemented by the java programming language and run using JDK 1.3.0, with 2GB of memory available for working storage, running on a personal computer Intel(R) Core(TM) i5-2430M CPU @2.4GHz.

In Tables 1 – 4, we present in the first column the number of jobs n, in the second column, the value of L, in the third column the model/algorithm applied, in columns 4 to 6, the minimal, average and maximal times used by the models/algorithms, and in columns 7 to 9, the minimal, average and maximum values of the relations C_{max}/LB (in Tables 5 and 6, the time limit is given in column 2 which is always exhausted by the algorithms).

Table 1: Results for *n=8*

n	*L*	mod/alg	min time sec	average time sec	max time sec	min C_{max}/LB	average C_{max}/LB	max C_{max}/LB
8	0.1	M1	33.0	48.4	59.7	1.00	1.00	1.00
		M2	0.0	0.1	0.2	1.00	1.02	1.05
		SA	9.5	10.5	13.4	1.00	1.00	1.00
	0.5	M1	2.4	20.8	58.3	1.00	1.00	1.00
		M2	0.0	0.1	0.1	1.00	1.01	1.02
		SA	10.6	13.2	15.4	1.00	1.00	1.00
	0.8	M1	1.5	9.1	18.8	1.00	1.00	1.00
		M2	0.0	0.1	0.1	1.00	1.01	1.03
		SA	11.0	13.8	15.5	1.00	1.00	1.00
	1.0	M1	2.6	8.2	18.4	1.00	1.00	1.00
		M2	0.0	0.0	0.1	1.00	1.00	1.00
		SA	13.2	14.3	15.7	1.00	1.00	1.00
	1.5	M1	0.6	2.8	6.0	1.00	1.00	1.00
		M2	0.0	0.0	0.1	1.00	1.02	1.06
		SA	13.0	15.0	15.9	1.00	1.00	1.00
	1.8	M1	0.3	2.5	6.5	1.00	1.00	1.00
		M2	0.1	1.1	2.2	1.00	1.00	1.01
		SA	15.5	15.0	11.7	1.00	1.00	1.00
	2.0	M1	0.3	2.4	6.0	1.00	1.00	1.00
		M2	0.0	0.0	0.0	1.00	1.00	1.00
		SA	12.5	14.9	16.5	1.00	1.00	1.00

In Table 1, we give the results for n=8. It can be seen that the model M1 and algorithm SA perform very well. However, the model M2 is very fast. It can also be observed that the model M1 requires larger computational times than algorithm SA for the instances with a small value of L while for the instances with a larger value of L it is opposite.

For n=20 (see Table 2), the model M2 is comparable with the heuristic SA both with respect to time and the quality of the solution. M2 takes much less time than heuristic SA and outperforms the model M1. Note that for the model M1, we used a time limit of 750 sec.

For n=50, 100, 200, the model M1 is unable to produce any solution for most examples, so we compare the heuristic SA only with model M2.

Table 2: Results for n=20

n	L	mod/alg	min time sec	ave time sec	max time sec	Min C_{max}/LB	Ave C_{max}/LB	Max C_{max}/LB
20	0.1	M1	750.0	750.0	750.0	1.04	1.07	1.13
		M2	0.0	0.1	0.5	1.00	1.01	1.02
		SA	41.4	44.1	48.2	1.00	1.00	1.01
	0.5	M1	750.0	750.0	750.0	1.07	1.09	1.13
		M2	0.4	28.4	96.0	1.01	1.05	1.13
		SA	55.5	60.4	65.1	1.01	1.03	1.07
	0.8	M1	750.0	750.0	750.0	1.05	1.09	1.11
		M2	0.2	10.3	29.7	1.02	1.05	1.11
		SA	56.3	62.7	71.6	1.01	1.03	1.06
	1.0	M1	750.0	750.0	750.0	1.09	1.14	1.19
		M2	10.6	91.5	340.9	1.05	1.08	1.10
		SA	57.6	64.9	68.6	1.05	1.06	1.08
	1.5	M1	750.0	750.0	750.0	1.10	1.14	1.20
		M2	2.1	8.7	17.2	1.03	1.07	1.10
		SA	61.4	66.0	67.6	1.03	1.06	1.10
	1.8	M1	750.0	750.0	750.0	1.06	1.10	1.16
		M2	0.1	1.1	2.2	1.00	1.04	1.06
		SA	52.7	58.4	64.2	1.00	1.03	1.05
	2.0	M1	750.0	750.0	750.0	1.02	1.06	1.15
		M2	0.0	0.9	3.1	1.01	1.03	1.08
		SA	62.3	70.9	79.4	1.01	1.03	1.07

Table 3: Results for n=50

n	L	mod/alg	min time sec	ave time sec	max time sec	min C_{max}/LB	ave C_{max}/LB	max C_{max}/LB
50	0.1	M2	0	77	238	1.00	1.00	1.01
		SA	369	459	586	1.00	1.00	1.00
	0.5	M2	177	980	1875	1.00	1.04	1.06
		SA	372	547	734	1.00	1.01	1.02
	0.8	M2	1069	1680	1876	1.03	1.06	1.11
		SA	463	510	619	1.02	1.03	1.04
	1	M2	1025	1622	1875	1.06	1.08	1.11
		SA	444	529	759	1.05	1.07	1.09
	1.5	M2	544	983	1531	1.03	1.05	1.10
		SA	331	436	509	1.03	1.05	1.08
	1.8	M2	577	664	782	1.03	1.05	1.07
		SA	336	410	540	1.02	1.05	1.07
	2.0	M2	531	609	691	1.02	1.06	1.11
		SA	338	420	519	1.02	1.05	1.10

For n=50 (see Table 3), algorithm SA is the best both with respect to time and the quality of the solution. However, for L=0.1 the model M2 gives similar results (and often even slightly better objective function values). Remind that model M2 looks for an optimal schedule within a very

special class of sub-schedules. For this reason, the results also show the distance from the considered subset of schedules to the optimal schedule.

Table 4: Results for n=100

n	L	mod/alg	min time sec	ave time sec	max time sec	min C_{max}/LB	ave C_{max}/LB	max C_{max}/LB
100	0.1	M2	15	233	633	1.00	1.00	1.00
		SA	1792	1865	1960	1.00	1.00	1.00
	0.5	M2	711	1039	1400	1.01	1.02	1.04
		SA	1843	2065	2212	1.00	1.01	1.01
	0.8	M2	1718	1847	2064	1.03	1.05	1.07
		SA	1592	1764	2086	1.01	1.02	1.04
	1	M2	1702	1861	2080	1.05	1.07	1.11
		SA	1925	2041	2332	1.02	1.04	1.07
	1.5	M2	1733	1837	1944	1.03	1.06	1.09
		SA	1525	1778	1886	1.02	1.05	1.07
	1.8	M2	1310	1683	1998	1.04	1.05	1.06
		SA	1561	1779	2432	1.03	1.04	1.05
	2.0	M2	920	1123	1354	1.01	1.03	1.05
		SA	1561	1833	2349	1.01	1.03	1.05

For n=100 (see Table 4) and L=0.8, 1, 1.5 algorithm SA produced better results than the model M2. However, for L=0.1, 0.5, 2.0 the model M2 gives similar results than heuristic SA (and often even a slightly better objective function value has been obtained).

Table 5: Results for n=200

n	time sec	L	mod/alg	min C_{max}/LB	ave C_{max}/LB	max C_{max}/LB
200	3600	0.1	M2	1.00	1.00	1.00
			SA	1.00	1.00	1.00
		0.5	M2	1.02	1.02	1.03
			SA	1.01	1.01	1.01
		0.8	M2	1.03	1.05	1.08
			SA	1.01	1.02	1.02
		1	M2	1.04	1.06	1.08
			SA	1.02	1.02	1.04
		1.5	M2	1.06	1.08	1.11
			SA	1.05	1.07	1.09
		1.8	M2	1.03	1.04	1.06
			SA	1.02	1.03	1.05
		2.0	M2	1.02	1.05	1.07
			SA	1.02	1.04	1.06

For n=200 (see Table 5), heuristic SA outperforms the model M2. Note that now the time is not sufficient to find an optimal solution for the model M2.

Table 6: Results for n=250

n	time sec	L	mod/alg	min C_{max}/LB	ave C_{max}/LB	max C_{max}/LB
250	3600	0.1	M2	1.00	1.00	1.00
			SA	1.00	1.00	1.00
		0.5	M2	1.01	1.04	1.06
			SA	1.01	1.01	1.02
		0.8	M2	1.04	1.06	1.08
			SA	1.01	1.02	1.02
		1	M2	1.06	1.07	1.08
			SA	1.02	1.02	1.03
		1.5	M2	1.04	1.07	1.12
			SA	1.03	1.05	1.09
		1.8	M2	1.03	1.05	1.06
			SA	1.02	1.03	1.04
		2.0	M2	1.02	1.05	1.07
			SA	1.02	1.04	1.05

For n=250 (see Table 6), algorithm SA outperforms the model M2.

Thus, up to 100 jobs, the model M2 produces solutions rather close to the optimal solution. Taking into account the very simple structure of the considered schedules, this result looks interesting. We also note that, from an overall point of view, the hardest instances in terms of the percentage deviation from the lower bound are those with L = 0.8, 1 and 1.5.

On the other hand, the $\sum C_j$ criterion appears to be much harder for the applied simulated annealing approach in comparison with the $C_{\max}$ criterion. Remind that a simulated annealing approach appears rather efficient for the problem $P2,S1||C_{\max}$, and it can work with instances containing up to 1000 jobs, see Hasani et al. (2013).

References

[1] Hall N., Potts C., Sriskandarajah C., 2000. Parallel machine scheduling with a common server. Discrete Applied Mathematics 102, 223-243.

[2] 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.

[3] Kravchenko S.A., Werner F. (2001). A heuristic algorithm for minimizing mean flow time with unit setups, Information Processing Letters 79, 291-296.

[4] Wang G., Cheng T.C.E. (2001). An approximation algorithm for parallel machine scheduling with a common server, Journal of the Operations Research Society 52, 234-237.

[5] Weng M. X., Lu J., Ren H. (2001). Unrelated parallel machine scheduling with setup consideration and a total weighted completion time objective. Int. J. Production Economics 70, 215-226.

[6] Dunstall S., Wirth A., 2005. Heuristic methods for the identical parallel machine flowtime problem with set-up times. Computers & Operations Research 32, 2479–2491.

[7] Azizoglu M., Webster S., 2003. Scheduling parallel machines to minimize weighted flowtime with family set-up times. Int. J. Production Research 41, 1199–1215.

[8] Guirchoun S., Martineau P., Billaut J.-C., 2005. Total completion time minimization in a computer system with a server and two parallel processors. Computers & Operations Research 32, 599–611.

[9] Hasani K., Kravchenko S.A., Werner F., 2013. Two heuristics for minimizing the makespan for the two-machine scheduling problem with a single server. Preprint 08/13, Faculty of Mathematics, Otto-von-Guericke-University Magdeburg, 20 pages.

[10] Kirkpatrick S., Gelatt C. D., Vecchi M. P., 1983. Optimization by Simulated Annealing. Science **220** (4598), 671–680.