Approximation algorithms for scheduling unrelated parallel machines with release dates

T V Avdeenko¹, Y A Mesentsev¹, I V Estraykh¹

1 Novosibirsk State Technical University, 20, Karla Marksa ave., Novosibirsk, 630073, Russia

E-mail: estrajx@corp.nstu.ru

Abstract. In this paper we propose approaches to optimal scheduling of unrelated parallel machines with release dates. One approach is based on the scheme of dynamic programming modified with adaptive narrowing of search domain ensuring its computational effectiveness. We discussed complexity of the exact schedules synthesis and compared it with approximate, close to optimal, solutions. Also we explain how the algorithm works for the example of two unrelated parallel machines and five jobs with release dates. Performance results that show the efficiency of the proposed approach have been given.

1. Introduction
Real world manufacturing systems usually include more than one machine. That is why, parallel machine scheduling problems have been studied by many researchers in recent years [9]. In parallel scheduling problem, set J of n jobs has to be scheduled on m parallel machines. The machines can be identical (with the same speed \( p_j \) of processing for each job \( j \)), uniform (with different speeds \( v_i \)) or unrelated (with different time \( p_{ij} \) of processing job \( j \) on machine \( i \)) [2].

Although, there is extensive literature on the parallel machine scheduling problem, the overwhelming majority of papers is limited to situations in which processing times or speed rates are the same across all machines (see, for example, [5, 6, 10]). Scheduling for the case of unrelated parallel machines, when machines are not identical to one another and cannot be completely correlated by simple rate adjustment, is one of the most difficult problems to solve, and is considered more rarely [4, 7, 12].

We consider the most difficult statement of the unrelated parallel machines problem without preemptions and with release dates, when the processing of job \( j \) cannot be started before its deterministic release date \( r_j \). We will consider makespan \( C_{max} = \max\{C_1, C_2, \ldots, C_n\} \) as a performance measure, where \( C_j \) is the completion time of job \( j \). Minimizing makespan not only completes all jobs as quickly as possible, but also allow maximizing the utilization of machines. Corresponding to notation given in [11], the problem is referred to as \( R \mid r_j \mid C_{max} \), that is, NP-hard in general.

To the best of our knowledge, there are only few papers devoted to \( R \mid r_j \mid C_{max} \), such as [1, 3, 8]. In [8], a heuristic and very effective particle swarm optimization (PSO) algorithm has been proposed to tackle the problem. The proposed heuristic was compared with the heuristic initially offered for identical parallel machines. The performance of the PSO algorithm has been analyzed for the case of small problem instances (18 jobs per 4 machines) and for the case of large problem instances (100 jobs per 10 machines).
In the present paper, we proposed efficient approaches that have better performance for large dimensions of the $R \mid r_j \mid C_{max}$ problem. A decrease in the makespan was not more than 5% from the optimal level for the instances of 100 jobs per 30 machines, which outperforms the PSO algorithm in [8] for the case of large problem instances.

2. Problem statement for unrelated parallel machines with release dates

Let us allocate a number of jobs to unrelated parallel machines with known speeds in order to minimize makespan. Preemptions are not allowed. Let us know the deterministic release dates of the parallel machines system. In this case, it is necessary to consider delays in jobs entering. Let us denote a delay of entering the $j$-th job as $\tau_j^0$. Now we have $T^0 = \|\tau_j^0\|$ as the initial schedule (job schedule in the input of the system). Let $x_{i,j}$ be Boolean variables if job $j$ is assigned to machine $i$; let $r_{i,j}$ be the time of processing job $j$ by machine $i$ ($T = \|r_{i,j}\|$); let $T^1 = \|r_j^0\|$ be the final schedule. We propose the following mathematical model for the problem:

\[
\sum_{j=1}^{J} x_{i,j} = 1, \quad j = 1, J, \quad (1)
\]

\[
\sum_{j=1}^{J} r_{i,j} x_{i,j} \leq \bar{r}_i, i = 1, I, \quad (2)
\]

\[
x_{i,j} = \begin{cases} 1, & \text{if the job } j \text{ is assigned to the machine } i, \\ 0, & \text{otherwise}, \end{cases}, \quad j = 1, J, \quad (3)
\]

\[
y_{i,j} \geq 0, \quad i = 1, I, \quad j = 1, J, \quad (4)
\]

\[
\hat{\tau}_{i,j} = \tau_j^0 - \sum_{j=1}^{J} (\tau_{i,j} + t_{i,j}) x_{i,j}, \quad i = 1, I, \quad j = 1, J, \quad (5)
\]

\[
\check{\tau}_{i,j} = \tau_{i,j} + y_{i,j} \geq 0, \quad i = 1, I, \quad j = 1, J, \quad (6)
\]

\[
\sum_{j=1}^{J} \check{\tau}_{i,j} x_{i,j} + \sum_{j=1}^{J} t_{i,j} x_{i,j} \leq \lambda, \quad i = 1, I, \quad (7)
\]

where $x_{i,j}$ is variables-assignments that have to be determined, $\hat{\tau}_{i,j}$ is actual delays of processing the $j$-th job by the $i$-th machine after finishing a previous job assigned to the machine. Compensating variables $y_{i,j}$ are introduced to avoid appearing negative delays $\check{\tau}_{i,j}$. Indeed, (4) and (6) guarantee that the following relation is valid:

\[
\check{\tau}_{i,j} = \begin{cases} \hat{\tau}_{i,j}, & \text{if } \tau_{i,j} \geq 0, \\ 0, & \text{otherwise}. \end{cases}
\]

The relations (1) - (3) are typical for an assignment problem. Relation (1) ensures assignment of any job to only one machine; relation (2) ensures assignment of not less than $\bar{r}_i$ and not more than $\bar{r}_i$ jobs to any machine $i$. Relations (5) - (6) give us actual delays of processing the $j$-th job by the $i$-th machine after it finishes processing the previous job.

Let us give the following criterion for model (1)-(7):

\[
\lambda \rightarrow \min. \quad (8)
\]

Now (1) - (8) define an optimal processing speed problem (makespan problem) for unrelated parallel machines with release dates for each job. Makespan criterion (7)-(8) has the effect of balancing the load over different machines, which is an important objective in practice.

\[
\max \{ \sum_{j=1}^{J} (q_{i,j} + \check{\tau}_{i,j}) x_{i,j} \} \rightarrow \min
\]
The schedule computed in the course of solving problem (1)-(8) is completely determined by optimal assignments \( x_{i,j}, i=\overline{1,I}, j=\overline{1,J} \) and actual delays for assignments \( \tau_{i,j}, i=\overline{1,I}, j=\overline{1,J} \). The schedule at the output of the system is \( \tau_j^* = \sum_{i=1}^{I} (\tau_{i,j} + t_{i,j}) x_{i,j}, j=\overline{1,J} \), the same being for optimal solution \( \tau_j^{**} = \sum_{i=1}^{I} (\tau_{i,j} + t_{i,j}) x_{i,j}^{*}, j=\overline{1,J} \).

The main difficulty of solution of problem (1)-(8) consists in the presence of Boolean variables, recursive functions and knapsack restrictions, defining the problem as NP-hard.

3. Approaches to unrelated parallel machines scheduling with release dates

One possible approach to solving problem (1)-(8) consists in its transformation to a usual mathematical programming form with subsequent direct solving the reduced system obtained as a result of transformation. Such reduction is conditioned by the fact that relations (5)-(8) contain recursive functions of actual delays \( \tau_{i,j} \). Relation (5) determines \( \tau_{i,j} \) in the explicit form as recursive functions \( 1 \), \( 0 \). Expansion of the recursions and further linearization of task (1)-(8) leads to the next task of linear programming with Boolean variables. Briefly let us describe corresponding transformations. Let us make a relevant convert.

Disclosure of recursions in (5)-(7) with the subsequent linearization leads to the following problem of the mixed programming [13]:

\[
\sum_{i=1}^{I} x_{i,j} = 1, \quad j = \overline{1,J},
\]

\[
b_j \leq \sum_{i=1}^{I} x_{i,j} \leq b_j, \quad i = \overline{1,I},
\]

\[
x_{i,j} = \begin{cases} 
1, & \text{if the job } j \text{ is assigned to the machine } i, \\
0 & \text{otherwise, } j=\overline{1,J}, i=\overline{1,I}
\end{cases}
\]

\[
u_{i,j,k} = \begin{cases} 
\hat{f}_{i,j,k} (x_{i,k}, x_{i,k+1}, \ldots, x_{i,j-1}) = x_{i,j} x_{i,k} \sum_{i=k+1}^{j-1} x_{i,j}, & \text{if the relation is true} \\
0 & \text{otherwise, } j=\overline{1,J}, i=\overline{1,I}, k=\overline{1,J}
\end{cases}
\]

\[-K + 2 \leq x_{i,j} + x_{i,k} - \sum_{i=k+1}^{j-1} x_{i,j} - ku_{i,j,k} \leq 1, \quad i = \overline{1,I}, \quad j = \overline{1,J}, \quad k = \overline{1,J}, \quad K = j - k + 1,
\]

\[-y_{i,j} + \sum_{k=1}^{j-1} (\tau_{i,j}^0 + t_{i,k}) u_{i,j,k} \leq \tau_j^0, \quad i = \overline{1,I}, \quad j = \overline{1,J},
\]

\[\sum_{j=1}^{J} (\tau_{i,j}^0 + t_{i,j}) x_{i,j} + \sum_{j=1}^{J} y_{i,j} - \sum_{j=1}^{J} \sum_{k=1}^{I} (\tau_{i,j}^0 + t_{i,k}) u_{i,j,k} \leq \lambda, \quad i = \overline{1,I},
\]

\[y_{i,j} \geq 0, \quad i = \overline{1,I}, \quad j = \overline{1,J},
\]

\[\lambda \rightarrow \min,
\]

where \( K = j - k + 1 \), \( y_{i,j} \) are continuous variables for compensation of negative delays.

In spite of theoretical solvability of reduced task (9)-(17), the approach is to be considered only as the initial stage of general solving process. The number of variables and restrictions in the reduced task increases \( J/2 \) times in comparison with (1)-(8). Thereby formal complexity of (1)-(8) is transformed to computational complexity of (9)-(17).

Another possible approach consists in simplifying the conditions of initial task (1)-(8). Instead of conditions (4)-(7) containing recursive functions, two criteria are introduced in the following way:
\[
\sum_{j=1}^{J} x_{i,j}^0 \leq \beta, \quad i = 1, J, \tag{18}
\]

\[
\sum_{j=1}^{J} t_{i,j} x_{i,j} \leq \lambda, \quad i = 1, J, \tag{19}
\]

\[
\lambda \rightarrow \min, \tag{20}
\]

\[
\beta \rightarrow \min. \tag{21}
\]

It is clear that the second approach to assigning jobs to the machines assumes a compromise solution both on the “pure” completion without accounting delays and on the uniform distribution of the delays between the machines. Then the Pareto-optimal solution of (1)-(3), (18)-(21) is determined, all components of which are admissible for (1)-(8) and (18)-(21). The best schedule for (1)-(8) is separated from the Pareto-optimal solution. Thereby, we succeed in eliminating the majority of variants and ensuring acceptable complexity and accuracy for a number of practical realizations of the task.

Numerical experiments with both models confirm that the second approach is efficient. While using (1)-(3), (18)-(21) for most examples, we have observed that a decrease in the makespan is not more than 5% from the optimal level. It is quite acceptable for many practical applications.

Development of the second approach has led to using decomposition of tasks (9)-(17) and (1)-(3), (18)-(21), and applying a general scheme of dynamic programming with sifting a part of possibilities. This allows us to certainly improve an accuracy of the solution, however, at the expense of decreasing computation time.

Finally, application of the dynamic programming method directly to initial statement (1)-(8) permits to develop one more algorithm ensuring compromise between computational complexity and the solution accuracy. This approach is proposed in the next chapter.

4. Dynamic programming method with sifting variants for unrelated parallel machines release dates scheduling

Dynamic programming is possibly the only method that can be directly applied to system (1)-(8) because of recursive functions (5). But its direct application is not efficient. The attempt to find the exact solution of (1)-(8) by the dynamic programming method leads to the complete enumeration of all admissible possibilities. It is not difficult to count total number \( N \) of such possibilities. For example, let \( s \) be the number of the stage, \( b_0 = 0 \) and \( b = J \) in (2), then \( N = I \cdot (1+2+\ldots+2^{j-1}+\ldots+2^{j-1}) = I \cdot 2^j - I \). Therefore, the method of dynamic programming applied to system (1)-(8) has exponential complexity and cannot be used in the pure form for the problems of actual dimensions.

To build an efficient approximate algorithm, we use a general scheme of dynamic programming making sifting of the locally worst variants at the certain stages of dynamic programming.

Let all jobs \( j = 1, J \) be ordered by the values of the initial delays (input schedule \( \{t_j\} \)). Then, in accord with the dynamical programming scheme, we define the numbers of stages \( s = 1,J \). Let \( f_{i,s}(\tau_i^0, t_{i,s}, x_{i,s}), \quad i = 1, J \) be the time when machine \( i \) finishes processing job \( j \) at stage \( s \), \( \phi_i(\tau_i^0, t_{i,j}, x_{i,j}) \), \( i = 1, J, \quad j = 1,s \) be conditionally minimal time of finishing all jobs at the stages from one to \( s \)-th,

\[
f_{i,s}(\tau_i^0, t_{i,s}, x_{i,s}) = \max\{0, [\tau_i^0 x_{i,s} - \phi_i d-1(\tau_{i-1}^0, t_{i-1}, x_{i-1})]\} + t_{i,s} x_{i,s}, \quad i = 1, J, \quad j = 1,s-1. \tag{22}
\]

Recurrent Bellman’s relation for this task is

\[
\phi_i(\tau_i^0, t_{i,j}, x_{i,j}) = \max_j \{ f_{i,s}(\tau_i^0, t_{i,j}, x_{i,j}) + \phi_i d-1(\tau_{i-1}^0, t_{i-1}, x_{i-1})\}, \quad j = 1,s-1, \tag{23}
\]

\[
\phi_i(\tau_i^0, t_{i,j}, x_{i,j}) = \max_j \{ \phi_i(\tau_i^0, t_{i,j}, x_{i,j})\}, \quad j = 1,s, \quad i = 1, J. \tag{24}
\]

To achieve a minimum makespan, according to criterion (7)-(8), one should find \( \min(\phi_j(\tau_j^0, t_{j,j}, x_{j,j})) \).
There are different variations of sifting a part of intermediate schedules at the stages of dynamic programming. If we discarded all variants of the schedule at stage $s$ except locally the best, we would obtain a greedy algorithm. If we left all intermediate schedules, we would obtain complete enumeration of all admissible variants. In the latter case, at the first stage, we will have $1 \cdot 2^0$, at the second stage $- 1 \cdot 2^1$, at the $s$-th stage $- 1 \cdot 2^{s-1}$ variants of intermediate schedule. If we want to find a compromise between accuracy and speed, the number of intermediate schedules has to polynomially depend on the number of variables in task (1)-(8).

Let us consider one of such compromises. Define a maximally possible number of variants $S$, which we leave at stage $s$ for further analysis. For the description convenience, let us assume that $S = 2^K$, where $K'$ is a certain constant. For example, if $K' = 10$, then $S = 1024$. Since the number of variants of intermediate schedules doubles at each stage, we propose sifting a half of the locally worst variants at each stage starting with $K' + 1$-th. Let us calculate a general number of schedule variants obtained for this case. For the first stage, we have $I$ variants, at stage $K'$ we have $1 \cdot 2^{K'-1}$ variants, at the stages from $K'+1$ to $J$ -th, we have $1 \cdot 2^k$ variants. Now the general number of intermediate schedules $N'$ is

$$N' = I \cdot (1 + 2 + \ldots + 2^{K'-1} + 2^{K'} \ldots + 2^k) = I \cdot [2^{K'} - 1 + (J - K') \cdot 2^k] \Rightarrow N' = I \cdot 2^{K'} (J - K') - 1$$ (25)

Since $S = 2^K$ is a certain constant, (25) represents polynomial dependence of complexity of this method on the problem (1)-(8) dimension. For example, let us compare $N$ with $N'$ for $K' = 10$, $J = 1000, I = 100 : N = 100 \cdot (2^{1000} - 1)$, $N' = 100 \cdot 1024(1000 - 10 + 1) - 100 = 10478300$. Thus, we see quite usual complexity of the hybrid algorithm, which is a multiple of $N$ and, actually, infinite complexity of the initial dynamic programming method which is multiple to $N$. Below, we describe the hybrid algorithm of dynamic programming with sifting of variants.

**Algorithm A**

1. Input initial data $(\phi_i^0, t_{ji}, x_{ji}, j = 1, J, i = 1, I)$ and parameters $K', S = 2^K$. Let $\phi_{i,0}(\phi_i^0, t_{ji}, x_{ji}) = 0$, define the initial number of stages $s := 0$.
2. Increase the number of stage $s := s + 1$.
3. Check the number of the stage. If $s > J$, go to step 7, otherwise go to step 4.
4. Generate all admissible variants of the schedule at stage $s$ according to (22)-(24), compute $f_{ij}(\phi_i^0, t_{ij}, x_{ij}), \phi_{i,0}(\phi_i^0, t_{ij}, x_{ij})$, length of schedules $\phi_i(\phi_i^0, t_{ij}, x_{ij})$.
5. Check number $N'$ of variants $\phi_{i,0}(\phi_i^0, t_{ij}, x_{ij})$ at stage $s$. If $N' \leq S$, go to 2. Otherwise, go to 6.
6. Sift a half of variants $\phi_{i,0}(\phi_i^0, t_{ij}, x_{ij})$ with maximum schedule length $\phi_i(\phi_i^0, t_{ij}, x_{ij})$. Go to 2.
7. Choose variants of schedules with the minimal length. Obtaining the final schedule with reverse motion of dynamic programming.

**5. Some results for the hybrid algorithm**

Initial data for tasks (9)-(17), (1)-(3), (18)-(21) and (1)-(8), were generated by random selection. Dimensions of all tests are equal to 100 jobs per 30 machines. This dimension is restricted only by the possibility of determining the exact solution to problem (9)-(17), it is necessary to estimate closeness of the approximate solution to the exact one.

| $N_0$ | $t_{da}$ | $\lambda_{da}$ | $\lambda_{dp}$ | $p_{dp}$ | $\Delta_{dp}$ | $t_{dp}$ | $\lambda_{dp}$ | $p_{dp}$ | $\Delta_{dp}$ |
|-------|----------|----------------|----------------|----------|-------------|----------|----------------|----------|-------------|
|       | in hours |               |               |          |             |          |               |          |             |
| 1     | 00:07:18:20 | 389 | 7 | 379 | -2.64 | -10 | 159 | 362 | -7.46 | -27 |
| 2     | 00:15:29:11 | 335 | 7 | 366 | 8.47 | 31 | 155 | 353 | 5.099 | 18 |
| 3     | 00:09:21:13 | 401 | 8 | 396 | -1.26 | -5 | 153 | 390 | -2.82 | -11 |
| 4     | 00:01:42:10 | 356 | 7 | 373 | 4.56 | 17 | 154 | 367 | 2.997 | 11 |

**Table 1.** Comparison characteristics of the approaches
In table 1, the following parameter of algorithm $A$ is used: $J = 100$. $\lambda_{da}$ is a criterion value for compromise solution (1)-(3), (18)-(21), obtained by the special decomposition algorithm. $t_{da}$ (in sec) is computation time for the decomposition algorithm time. $\lambda_{dp}$ is a criterion value for algorithm $A$. $t_{dp}$ (in sec) is computation time for algorithm $A$; $\Delta_{dp} = \lambda_{dp} - \lambda_{da}$, $p_{dp} = (\lambda_{dp}/\lambda_{da} - 1)*100$ is a ratio of deviation from the criterion value of the decomposition algorithm for algorithm $A$. Equal values of $t_{dp}$ are explained by the same parameters of algorithm $A$.

6. Conclusion
The main result of the present paper is an efficient approach to an unrelated parallel machine with release dates $R$ | $r_j$ | $C_{max}$ scheduling based on mathematical formulation (1)-(8). One of the proposed algorithms is based on the scheme of dynamic programming with sifting of locally worse variants at each stage. An important practical result of the work is a possibility to achieve a compromise between accuracy and the hybrid algorithm speed by means of variation of its parameters. The approaches proved to be efficient and accurate for large dimensions of the $R$ | $r_j$ | $C_{max}$ scheduling problem (not more than a 5% decrease of the makespan from the optimal level for a parallel system consisting of 100 jobs on 30 machines).

References
[1] Bank J and Verner F 2001 Mathl. Comput. Modelling 33(4-5) 363–383
[2] Brucker P 2007 Scheduling Algorithms (Springer) p 372
[3] Gharbi A and Haouari M 2002 J. Scheduling 5(4) 329–355
[4] Kaabi J and Harrath Y 2014 JICIT 03(02) 238–245
[5] Kashan A H and Karimi B 2009 Comput. Ind. Eng. 56(1) 216–223
[6] Lee W C, Wu C C and Chen P 2006 Int. J. Adv. Manuf. Technol. 31(3-4) 328–334
[7] Lin Y K, Hsieh H T and Hsieh F Y 2012 World Acad. Sci. Eng. Technol. 6 1798–1803
[8] Lin Y K 2013 Math. Probl. Eng. 2013 1-9
[9] Mokotoff E 2001 Asia Pac. J. Oper. Res. 18 193–243
[10] Pfund M, Fowler J W, and Gupta J N D 2004 J. Chin. Inst. Ind. Eng. 21 230–241
[11] Pinedo M 2008 Scheduling Theory, Algorithms, and Systems (Springer) p 672
[12] Santos F C, Vilarinho P M 2010 Proc. WCE 2010 3 (London, U.K.) 2443–48
[13] Mesentsev Y A 2009 Informacionnye tehnologii 6 35–41