Multi-machine scheduling problem with setup times

Wojciech Bożejko[1]
Mariusz Uchroński[2] and Mieczysław Wodecki[3]*

[1]Institute of Computer Engineering, Control and Robotics
Wrocław University of Technology
Janiszewskiego 11-17, 50-372 Wrocław, Poland

[2]Wrocław Centre of Networking and Supercomputing
Wyb. Wyspańskiego 27, 50-370 Wrocław, Poland

[3]Institute of Computer Science, University of Wrocław
Joliot-Curie 15, 50-383 Wrocław, Poland

Abstract

In this paper we consider a multi-machine scheduling problem with setup times, which is determined in the literature as the flexible job shop problem. It belongs to the strongly NP-complete complexity class. We propose an algorithm based on the tabu search method. The new elimination criteria were used in the construction process of blocks of the critical path.

1 Introduction

In the review work of Panwalkara et al. [7] concerning tsks scheduling it was noted that 75 % of problems occurring in practice requires at least one setup dependent on the order of tasks. On the other hand, in the 15 % of the problems one should take into consideration a setup between all tasks. Nevertheless, in the vast majority of works in the field of tasks scheduling

*Corresponding author: Mieczysław Wodecki, email: mwd@ii.uni.wroc.pl

the setup times are not included. It applies to both problems of one- and multi-machine and different goal functions. Such issues are important from the theoretical and the practical perspective.

The problem considered in this work consists in assigning tasks to machines and determining the order of their performance in order to minimize the time consumed to perform all the tasks. The times of tasks execution and machine setup times between the tasks performed sequentially are given. In addition, the following constraints must be met:

(i) each task can be performed at the same time only on one, the correct type of machine,

(ii) none of the machines can perform more than one task at the same time,

(iii) the task execution cannot be interrupted,

(iv) the technological line of tasks execution must be preserved.

Thus, it is a well known in the literature flexible job shop problem (see, e.g., [2]) with an additional constraint associated with the setup of machines. In brief, this problem will be denoted by **FJSST**.

Accurate algorithms to solve the flexible job shop problem were presented in the works [4], [1] and [9]. They allow us to solve, in an acceptable time, instances of sizes up to several tasks and machines. Approximate algorithms have been described, among other things, in the work of [2] [3] and [8]. Since the problem **FJSST** lies in a generalization of the classic job shop problem, it belongs to strongly NP-complete class. In the further part of the work we introduce a new property of the problem, which was used in the construction of an algorithm based on the tabu search method.

2 Problem formulation

Considered problem can be formulated as follows: there is a set of tasks given $\mathcal{J} = \{1, 2, \ldots, n\}$, which must be executed on machines from set $\mathcal{M} = \{1, 2, \ldots, m\}$. There is a break down of machine sets into types (*slots*), i.e. into subsets of machines with the same functional properties. The task is a sequence of specific operations. Each operation must be performed on the appropriate type of machine in the set time. Before performing of any task in the stipulated time, one should setup the machine. The problem lies within the allocation of tasks to machines of appropriate type and setting the order of operations on machines to minimize the execution time of all tasks. In the operations certain constraints must be met (i)-(iv).

Let $\mathcal{O} = \{1, 2, \ldots, o\}$ be a set of all operations.

The set can be broken down into sequences corresponding to tasks, where the task $j \in \mathcal{J}$ is a sequence of o_j operations, which will be performed respectively on appropriate machines (i.e. in a technological order). The operations are indexed with numbers $l_{j-1} + 1, \ldots, lj - 1 + o_j$, where $l_j = \sum_{i=1}^{j} o_i$ is a number of first operations of j tasks ($j = 1, 2, \ldots, n$), wherein $l_0 = 0$ and $o = \sum_{i=1}^{n} o_i$. In turn, the set of machines $\mathcal{M} = \{1, 2, \ldots, m\}$ can be broken into q subsets of machines of the same type (slots), i.e.

$$\mathcal{M} = \mathcal{M}^1 \cup \mathcal{M}^2, \ldots, \mathcal{M}^q.$$

Operation $v \in \mathcal{O}$ should be performed in a slot $\mu(v)$, i.e. on one of the machines of the set $\mathcal{M}^{\mu(v)}$ in time $p_{v,j}$, where $j \in \mathcal{M}^{\mu(v)}$. Next, by $s_{i,j}$ ($i, j \in \mathcal{J}$) we denote the time of machine setup, i.e. the time required to prepare the machine to execute the operation j, if directly before j there was a i task performed. It must be noted that $s_{0,i}$ is the time of machine 'setup' (preparation), if the task i is performed as the first one, and $s_{i,0}$ is the time of machine 'disassembling', if the task i is performed as the last one. The considered here problem boils down to assigning operations to machines and determining the order of their execution to minimize the execution time of all tasks.

By

$$\mathcal{O}^k = \{v \in \mathcal{O} : \ \mu(v) = k\}$$

we denote a set of operations executed in k-th ($k = 1, 2, \ldots, q$) slot. The sequence of operations set

$$\mathcal{Q} = [\mathcal{Q}^1, \mathcal{Q}^2, \ldots, \mathcal{Q}^m],$$

such as for every $k = 1, 2, \ldots, q$

$$\mathcal{O}^k = \bigcup_{i \in \mathcal{M}^k} \mathcal{Q}^i \ \text{ and } \ \mathcal{Q}^i \cap \mathcal{Q}^j = \emptyset, \ i \neq j, \ i, j = 1, 2, \ldots, m,$$

we call *assignment of operations of the set $\mathcal{O}$ to machines from the set $\mathcal{M}$*. A sequence $[\mathcal{Q}^{t_{k-1}+1}, \mathcal{Q}^{t_{k-1}+2}, \ldots, \mathcal{Q}^{t_{k-1}+m_k}]$ is an ***allocation*** of machines to operation in i-th slot (in brief *allocation* in i-th slot).

Let $\mathcal{Q}$ be an arbitrary assignment of operations to machines and

$$\pi(\mathcal{Q}) = (\pi_1(\mathcal{Q}), \pi_2(\mathcal{Q}), ..., \pi_m(\mathcal{Q}))$$

concatenation (joining) m of sequences (permutations), where $\pi_i(\mathcal{Q})$ is a permutation of tasks performed on i-th machine.

It is worth to see that any feasible solution to the **FJSST** problem is a pair $(\mathcal{Q}, \pi(\mathcal{Q}))$, where $\mathcal{Q}$ is assignment of operations to machines, and $\pi(\mathcal{Q})$ concatenation of permutations designating the order of operations assigned to each of the machines which comply with the constraints (i)-(iv).

3 Graph representation of the solution

Any feasible solution $\Theta = (\mathcal{Q}, \pi(\mathcal{Q}))$ can be represented as a directed graph (network) of burdened vertices and arcs $G(\Theta) = (\mathcal{V}, \mathcal{R} \cup \mathcal{E}(\Theta))$. In the graph $\mathcal{V}$ is a set of vertices and $\mathcal{R} \cup \mathcal{E}(\Theta)$ is a set of arcs, wherein

1) $\mathcal{V} = \mathcal{O} \cup \{s, c\}$, where s i c are additional operations representing, respectively, the 'beginning' and 'end '.

   Vertex $v \in \mathcal{V} \setminus \{s, c\}$ has two characteristics:

   - $\lambda(v)$ – number of the machine on which the operation should be executed $v \in \mathcal{O}$,
   - $p_{v,\lambda(v)}$ – weight of vertex equal to time of operation execution $v \in \mathcal{O}$ on machine $\lambda(v)$.

   Weighs of added vertices $p_s = p_c = 0$.

2) $\mathcal{R} = \bigcup_{j=1}^{n} \left[\bigcup_{i=1}^{o_j - 1} \{(l_{j-1} + i, l_{j-1} + i + 1)\} \cup \{(s, l_{j-1} + 1)\} \cup \{(l_{j-1} + o_j, c)\} \right].$

   The set $\mathcal{R}$ has arcs:

   - connecting successive operations of the same task. Weight of arc is 0
   - from s vertex to the first operation of each task. Its weight is equal to the time of the preparation of the machine to perform the first operation,
   - from the last operation of each task to c vertex. Weigh of this arc is equal to the time 'disassembly' of the machine after the last operation.

3) $\mathcal{E}(\Theta) = \bigcup_{k=1}^{m} \bigcup_{i=1}^{|\mathcal{O}^k|-1} \{(\pi_k(i), \pi_k(i+1))\}.$

   Arcs of this set combine operations on the same machine. Weigh of arc $(\pi_k(i), \pi_k(i+1))$ equals the time of machine setup between sequentially performed operations $\pi_k(i)$ oraz $\pi_k(i+1)$.

Arcs of the set $\mathcal{R}$ determine the order of operations for each task (technological order), and arcs from the set $\mathcal{E}(\Theta)$ the order of operations on each machine.

Remark 1 *The pair $\Theta = (\mathcal{Q}, \pi(\mathcal{Q})$ is a feasible solution to the **FJSST** problem if and only when the graph $G(\Theta)$ does not contain cycles.*

Remark 2 *Time of tasks execution, according to the solution* $\Theta = (\mathcal{Q}, \pi(\mathcal{Q}))$ *is equal to the length of the critical path (i.e. the longest path) from the vertex* s *to* c *in graph* $G(\Theta)$.

Solving the job shop problem with parallel machines and setup times boils down to determining such a feasible solution $\Theta = (\mathcal{Q}, \pi(\mathcal{Q}))$, for which the corresponding graph $G(\Theta)$ has the shortest possible critical path.

4 Operation blocks along the critical path

Let $\Theta = (\mathcal{Q}, \pi(\mathcal{Q}))$ be a feasible solution. By $C(s, c) = (s, v_1, v_2, \ldots, v_w, c)$, where $v_i \in \mathcal{O}$ $(1 \leq i \leq w)$ denote the critical path in the graph $G(\Theta)$. This path can be broken into subsequences of vertices (subpermutations of operations)

$$\mathcal{B} = [B^1, B^2, \ldots, B^r]$$

such as

(a) each subsequence contains subsequent operations executed directly one after another on the same machine,

(b) cross-section of two arbitrary subsequences is an empty set,

(c) each subsequence is a maximum (due to entering) the subset of the operations satisfying the critical path constraints (a)-(b).

Subsequence B^k $(k = 1, 2, ..., r)$ of operation from critical path executed on machine M_i $(i \in \mathcal{M})$ will be denoted as folows:

$$B^k = (\pi_i(a^k), \pi_i(a^k + 1), \ldots, \pi_i(b^k - 1), \pi_i(b^k)),$$

where $1 \leq a^k \leq b^k \leq |Q^i|$. Operations $\pi(a^k)$ i $\pi(b^k)$ are respectively *the first* and *the last* in a sequence.

For a fixed subsequence $B^k = (\pi_i(a^k), \pi_i(a^k + 1), \ldots, \pi_i(b^k))$ from the sequence of critical path operation by Φ^k we denote a set of all permutations of elements of a set $\{\pi_i(a^k + 1), \pi_i(a^k + 2), \ldots, \pi_i(b^k - 1)\}$. Let $\beta^* \in \Phi^k$ be a permutation such as

$$\Psi(\beta^*) = min\{\Psi(\gamma) : \ \gamma \in \Phi^k\}, \tag{1}$$

where

$$\Psi(\gamma) = s_{\pi_i(a^k),\gamma(1)} + \sum_{i=a^k+1}^{b^k-1} s_{\gamma(i),\gamma(i+1)} + s_{\gamma(b^k-1),\pi_i(b^k)}$$

is the lenght of a path $(\pi_i(a^k), \gamma(1), \gamma(2), \ldots, \gamma(b^k - 1), \pi_i(b^k))$. Permutation β^* realizes the shortest path between vertices $\pi_i(a^k)$ oraz $\pi_i(b^k))$.

The sequence of operations performed on k-th machine $\widehat{B}^k = (\pi_i(a^k), \beta^*, \pi_i(b^k))$ is called k-th **block**, and permutation β^* – is an internal block. It is symbolically shown in figure 1.

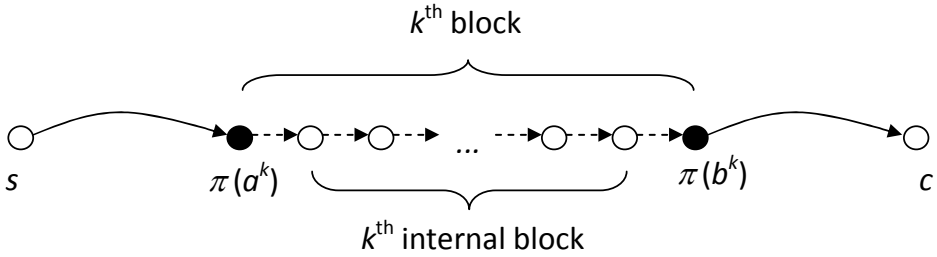

Figure 1: Subsequence from critical path.

By determining the blocks from the critical path of solution Θ we generate some new solutions of a value not greater than the value of solutions Θ. The procedure can be regarded as a form of local optimization (improvement Θ).

Theorem 1 *Let Θ be a feasible solution of the job shop problem with parallel machines and setups. If $\widehat{B}^k$ is a block from the critical path, then any change in the sequence of operations of the internal block does not generate solutions with goal function value smaller than the value of the function solution Θ.*

Proof.

Let $\Theta = (\mathcal{Q}, \pi(\mathcal{Q}))$ be a feasible solution of the considered problem. By $L^{\Theta}(u, v)$ we denote the length of the shortest path from vertex u to v in the graph $G(\Theta)$ and in order to count the length of u vertex and we do not count vertex v weight.

We consider k-th block

$$\widehat{B}^k = (\pi_i(a^k), \pi_i(a^k + 1), \ldots, \pi_i(b^k - 1), \pi_i(b^k))$$

from the critical path in graph $G(\Theta)$. The length of the way

$$L^{\Theta}(s, c) = L^{\Theta}(s, \pi_i(a^k)) + L^{\Theta}(\pi_i(a^k), \pi_i(b^k)) + L^{\Theta}(\pi_i(b^k), c). \qquad (2)$$

Let us assume that the solution Ω was generated from Θ by changing the order of operations of the internal block $\widehat{B}^k$. In graph $G(\Omega)$ there is a path $(s, \pi_i(a^k), \pi_i(a^k + 1), \ldots, \pi_i(b^k))$ from vertex s do c going among others through vertices $\pi_i(a^k)$ and $\pi_i(b^k)$, whose length

$$L^{\Omega}(s, c) = L^{\Omega}(s, \pi_i(a^k)) + L^{\Omega}(\pi_i(a^k), \pi_i(b^k)) + L^{\Omega}(\pi_i(b^k), c). \qquad (3)$$

It is easy to see

$$L^{\Theta}(s, \pi_i(a^k)) = L^{\Omega}(s, \pi_i(a^k)) \text{ and } L^{\Theta}(\pi_i(b^k), c) = L^{\Omega}(\pi_i(b^k), c).$$

Since the sequence of operations $(\pi_i(a^k), \pi_i(a^k + 1), \ldots, \pi_i(b^k - 1), \pi_i(b^k))$ creates block in graph $G(\Theta)$, thus, from definition (1)

$$L^{\Omega}(\pi_i(a^k), \pi_i(b^k)) \geq L^{\Theta}(\pi_i(a^k), \pi_i(b^k)).$$

With the use of (2), (3) and the above equation we obtain

$$L^{\Theta}(s, c) \geq L^{\Omega}(s, c). \tag{4}$$

Since the obtained, in graph, $G(\Omega)$ path from vertex s to c of $L^{\Omega}(s, c)$ lenght is one of possible paths between these vertices, so the length of the critical path in this graph is not less than $L^{\Omega}(s, c)$. Therefore, using (4) the value of the goal function for Ω solution is not less than the value for Θ solution. This fact completes the proof of the theorem.

The theorem shows that the possible improvement of the value of the current solution may be made only by:

- shifting of one of the operations from some internal block before the first or after the last operation of this block,
- transferring the operation to another, from the same slot, machine.

In the work [2] there appears a description of an algorithm based on the tabu search method of solving the job shop problem with parallel machines (excluding setup of machines). The algorithm uses the so-called golf neighborhood. It is generated by making such movements as move and transfer. The first one changes the order of operations on the machine, the second - the transfer of operations to another machine (from the same slot.) In solution algorithm, considered as the job shop scheduling problem with **FJSST** setups, we also used the golf neighborhood, however, for its generation, we must use elimination properties of blocks from the critical path (Theorem 1). The general idea of generating the neighborhood solution Θ can be presented as follows:

1. generate graph $G(\Theta)$,
2. determine the critical path in the graph $G(\Theta)$, and then break down a set of operations into the sub-permutations complying with the constraints (a)–(c),

3. determine in accordance with block definition the order of operations in sub-permutation,

4. generate golf neighborhood (a detailed description is included in the work [2]), using elimination properties of blocks (Theorem 1).

If B is a subsequence of the operation from the critical path in the graph G, then determining of the order of operation that fulfills the definition of block constraints requires the designation of the shortest path (permutations of elements) between the first and last operation in B. It is easy to notice that this procedure comes down to solving of the traveling salesman problem. Vertices are operations of B, and the distance between the vertices are the times of machine setups between operations. Solution to this problem is therefore NP-complete problem. This is the reason why to its solution we used algorithm 2-*opt*, one of the most popular approximate algorithms.

Algorytm 2-opt

Step 1: Determine the starting solution (arbitrary Hamilton cycle);

Step 2: Verify if there is a pair of edges, whose exchange to another pair generates Hamilton cycle with a smaller length. If it is possible, then determine new (shorter) Hamilton cycle; Repeat step 2, as long as it is possible.

A thorough analysis of the results of various approximation algorithms (and 2-*opt*) for the traveling salesman problem is included in the work [5]. It was clearly stated there that the average relative error of this algorithm (for reference data) in reference to the best currently known solutions is about 5%. The algorithm has a complexity $O(n^2)$.

Application of approximation algorithm to solve the traveling salesman problem means that one of the properties of block definition may not be satisfied. As a result, from the neighborhood solution Θ the 'good' elements may be eliminated. This drawback can be partially improved by using, in a small number of vertices, exact algorithm, or imroved approximate algorithm.

5 Computational experiments

Computational experiments were performed on a computer equipped with Intel Core i7 X980 CPU running under 64-bit Linux operating system Ubuntu 10.04. Test instances were generated on the basis of test examples for flexible job shop problem taken from the literature [3]. Machine setup times were randomly generated according to uniform distribution on the set $\{1, 2, \ldots, 10\}$.

The obtained results were compared with the values set by INSA construction algorithm [**?**]. Column 3 (flex) in Table 1 is the mean number of machines, on which a single operation can be performed. Next two columns show the average improvement of the solutions determined by INSA . Column TS – taboo search algorithm with the golf neighbourhood ([2]) and column TS_{2OPT} – an algorithm that uses elimination block properties (Theorem 1) in generating of the neighbourhood.

Table 1: Relative improvement of the starting solution.

Problem	$n \times m$	flex	TS	TS_{2OPT}
Mk01	10×6	2.09	-23.65	-23.65
Mk02	10×6	4.10	-42.96	-40.62
Mk03	15×8	3.01	-11.79	-23.05
Mk04	15×8	1.91	-15.78	-9.02
Mk05	15×4	1.71	-8.35	-25.38
Mk06	10×15	3.27	-19.50	-25.00
Mk07	20×5	2.83	-23.20	-23.20
Mk08	20×10	1.43	-7.18	-4.63
Mk09	20×10	2.53	-18.45	-17.13
Mk10	20×15	2.98	-8.02	-16.54
Average			-17.89	-20.82

Additional information

The work was partially financed from the research project of Ministry of Science and Higher Education no N N514 232237.

References

[1] Adrabiński A., Wodecki M., An algorithm for solving the machine sequencing problem with parallel machines, Zast. Matematyki XVI, 3(1979), 513–541.

[2] Bożejko W., Uchroński, Wodecki M.: Parallel hybrid metaheuristics for the flexible job shop problem. Computers & Industrial Engineering 59, 2010, 323–333.

[3] Brandimarte P.: Routing and scheduling in a flexible job shop by tabu search. Annals of Operations Research, 41, 1993, 157–183.

[4] Grabowski J., Adrabiński A., Wodecki M.: Algorytm rozwiązania ogólnego problemu kolejnościowego z równolegymi maszynami. Konferencja: Programowanie Matematyczne w Badaniach Ekonomicznych, Jachranka, 1978, 41–53.

[5] Johnson D., McGeoch L.: Experimental analysis of heuristics for the STST. In G.Gutin and A.Punnen, editors, The Traveling Salesman Problem and its Variations, Kluwer, 2002, 369–444,

[6] Nowicki E., Smutnicki C.: A fast tabu search algorithm for the job shop problem. Management Science, 42, 1996, 797-813.

[7] Panwalkar S.S., Dudek R.A., Smith M.L.: Sequencing research and the industrial scheduling problem. Symposium on the theory of scheduling and its applications (ed. Elmaghraby S.E.), Springer-Verlag, Berlin, 1973.

[8] Pezzella F., Morganti G., Ciaschetti G.: A genetic algorithm for the Flexible Job-schop Scheduling Problem. Comp. & Oper. Res., 35, 2008, 3202–3212.

[9] Pinedo M.: Scheduling: theory, algorithms and systems. Englewood Cliffs, NJ: Prentice-Hall, 2002.