Solving the flow shop problem by parallel tabu search

Wojciech Bożejko
Wrocław University of Technology
Institute of Engineering
Janiszewskiego 11-17
50-372 Wrocław, Poland
wbo@ict.pwr.wroc.pl

Mieczysław Wodecki
University of Wrocław
Institute of Computer Science
Przesmyckiego 20
51-151 Wrocław Poland
mwd@ii.uni.wroc.pl

Abstract

In this paper we present parallel tabu search algorithm for the permutation flow shop sequencing problem with the objective of minimizing the flowtime. We propose a neighbourhood using so-called blocks of jobs on a critical path and a backtrace jump method. By computer simulations it is shown that the performance of the proposed algorithm is comparable with the random heuristic technique discussed in literature. Other interesting property is the fact that the speedup of parallel implementation is equal or even greater than p, where p is the number of processors.

1. Introduction

We take under consideration the permutation flow shop scheduling problem described as follows. A number of jobs are to be processed on a number of machines. Each job must go through all the machines in exactly the same order and the job order is the same on every machine. Each machine can process at most one job at any point in time, and each job may be processed on at most one machine at any time. The objective is to find a schedule that minimizes the completion time of the last job. The problem is indicated by $F|m|C_{\max}$. Johnson [8] gives an O(nlogn) algorithm for $F|2|C_{\max}$ and Garey, Johnson and Seti [2] show that $F|3|C_{\max}$ is strongly NP-hard. The best available branch and bound algorithms are those of Ignall, Schrage [6], Lageweg, Lenstra and Rinnooy Kan [9] and Grabowski [4]. Their performance is not entirely satisfactory however, as they experience difficulty in solving instances with 20 jobs and 5 machines. Various local search methods are available for the permutation flow shop problem. Tabu search algorithms are proposed by Taillard [16], Reves [14] and Nowicki, Smutnicki [11]. Sequential simulated annealing algorithms are proposed by Osman, Potts [13], Ogbu, Smith [12], Ishibuchi, Misaki and Tanaka [7]. Parallel simulated annealing algorithm is proposed by Wodecki and Bożejko [19]. Reeves [15] proposes a genetic algorithm which uses the reorder crossover.

There are several methods of parallelization of tabu search method [18]. Eikelder, B. Aarts, Verhoven and E. Aarts show parallel tabu search algorithm for job shop scheduling problem [1]. In our paper we will present sequential and parallel tabu search algorithms for the permutation flow shop problem. Parallel version of our algorithm is based on parallel cooperative search processes with backtrack-jump list used to diversification of search process.

2 Problem definition and notation

The flow shop problem can be defined as follows, using the notation by Nowicki, Smutnicki [11] and Grabowski, Pempera [5] as a proper one for the problem considered. There are: a set of n jobs $J = \{1, 2, \ldots, n\}$, a set of m machines $M = \{1, 2, \ldots, m\}$. Job $j \in J$, consists of a sequence of m operations $O_{j1}, O_{j2}, \ldots, O_{jm}$. Operation O_{jk} corresponds to the processing of job j on machine k during an uninterrupted processing time p_{jk}. We want to find a schedule such that the maximum completion time is minimal.

Let $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ be a permutation of jobs $\{1, 2, \ldots, n\}$ and Π be the set of all permutations. Each permutation $\pi \in \Pi$ defines a processing order of jobs on each machine. We wish to find a permutation $\pi^* \in \Pi$ that:

$$C_{\max}(\pi^*) = \min_{\pi \in \Pi} C_{\max}(\pi),$$

where $C_{max}(\pi)$ is the time required to complete all

jobs on the machines in the processing order given by the permutation π. Completion time of job $\pi(j)$ on machine k can be found using the recursive formula:

$$C_{\pi(j)k}=\max\{C_{\pi(j-1)k},\, C_{\pi(j)k-1}\} + p_{\pi(j)k}\},$$

where $\pi(0) = 0$, $C_{0k} = 0$, $k = 1, 2, ..., m$, $C_{0j} = 0$, $j = 1, 2, ..., n$.

It is well known that $C_{max}(\pi) = C_{\pi(n)m}$.

For each permutation $\pi \in \Pi$ we define the following digraph:

$$D(\pi) = (O, AV(\pi) \cup AH(\pi)),$$

where $O = \{O_{j1}, O_{j2}, \ldots, O_{jm}\}$ is the set of nodes:

$$AV(\pi) = \bigcup_{i=1}^{m-1} \bigcup_{j=1}^{n} \{(O_{\pi(j)i}, O_{\pi(j)i+1})\},$$

is the set of vertical arcs and:

$$AH(\pi) = \bigcup_{i=1}^{m} \bigcup_{j=1}^{n-1} \{(O_{\pi(j)i}, O_{\pi(j+1)i})\},$$

is the set of horizontal arcs (see Fig. 1).

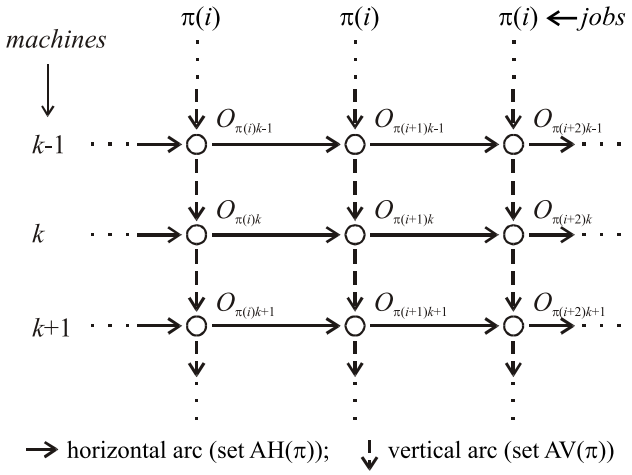

Figure 1. Digraph $D(\pi)$.

The longest path in graph $D(\pi)$ from node $O_{\pi(1)1}$ to $O_{\pi(n)m}$ is called a *critical path* with respected π. It's length is $C_{\max}(\pi)$.

The critical path is decomposed into subsequences $B_1, B_2, \ldots, B_m$ called *blocks* in π, where:

a) $B_k = (\pi(f_k), \pi(f_k + 1), ..., \pi(l_k - 1), \pi(l_k)), f_k \leqslant l_k, f_1 = 1, l_k = k$ and $\pi(l_k) = \pi(f_{k+1})$, $k = 1, 2, ..., m - 1$,

b) B_k contains operations processed on the same machine, $k = 1, 2, \ldots, m$,

c) two consecutive blocks contain operations processed on different machines.

In other words, the block is a maximal subsequence of the critical path, which contains operations processed on the same machine. Operations $\pi(f_k)$ and $\pi(l_k)$ in B_k are called the *first* and *last* ones, respectively.

Theorem 1 (Grabowski [4])

Let $G(\pi)$ be a graph with blocks B_k, $k = 1, 2, ..., m$. If graph $G(\omega)$ has been obtained from $G(\pi)$ by an interchange of jobs and if $C_{max}(\omega) < C_{max}(\pi)$, then in $G(\omega)$:

(i) at least one job $j \in B_k$ precedes job $\pi(f_k)$, for some $k = 2, ..., m$, or

(ii) at least one operation $j \in B_k$ succeeds job $\pi(l_k)$, for some $k = 1, 2, ..., m - 1$.

Theorem 1 gives the necessary condition to obtain a permutation such that $C_{max}(\omega) < C_{max}(\pi)$.

Let $B_k^f = B_k \backslash \{\pi(f_k)\}$ and $B_k^l = B_k \backslash \{\pi(l_k)\}$ to be subblocks of jobs of the k-th block without the first and last job. To get permutation ω from permutation π such as $C_{max}(\omega) < C_{max}(\pi)$ we will move jobs from the set B_k^f before the first job $\pi(f_k)$, and jobs from the set B_k^l after the last $\pi(l_k)$, $k = 1, 2, ..., m$.

For job $j \in B_k^f$ let

$$\Delta_k^f(j) = \begin{cases} p_{jk-1} - p_{\pi(f_k)k-1} & j \neq \pi(l_k), \\ p_{jk-1} - p_{\pi(f_k)k-1} + & \\ +p_{\pi(l_k-1)k+1} - p_{\pi(j)k+1} & j = \pi(l_k), \end{cases}$$

and for $j \in B_k^l$

$$\Delta_k^l(j) = \begin{cases} p_{jk+1} - p_{\pi(l_k)k+1} & j \neq \pi(f_k), \\ p_{\pi(f_k+1)k-1} - p_{\pi(j)k-1} + & \\ +p_{\pi(j)k+1} - p_{\pi(l_k)k+1} & j = \pi(f_k), \end{cases}$$

where $k = 1, 2, \ldots, m$ and $p_{\pi(i)j} = 0$, $i > n$, $j < 1$ or $j > k$.

Theorem 2 (Grabowski [4])

For each $\pi \in \Pi$, if β is the permutation obtained from π by moving job j, $(j \in B_k)$ before the first or after the last job in block B_k, then we have:

$$C_{\max}(\beta) \geqslant C_{\max}(\pi) + \Delta_k^f(j) \text{ or}$$
$$C_{\max}(\beta) \geqslant C_{\max}(\pi) + \Delta_k^l(j).$$

By moving job $j \in B_k$ before $\pi(f_k)$ or after $\pi(l_k)$ in π, we generate permutation β and the lower bound on the value $C_{\max}(\beta)$ is $C_{\max}(\beta) \geqslant C_{\max}(\pi) + \Delta_k^f(j)$ or $C_{\max}(\pi) + \Delta_k^l(j)$. Thus the values $\Delta_k^f(j)$ and $\Delta_k^l(j)$ can be used to decide which job should be moved.

3 Tabu search method

Nowadays the most effective methods to solve flow shop problem are based on the tabu search (TS) method ([5], [11]). The method was proposed by Glover [3]. Generally, it consists in improving the starting solution's value π^*. Algorithm generates neighbourhood of the current solution and seek the solution which has the minimal value of $C_{\max}(\beta)$, $\beta \in \mathcal{N}(\pi^*)$. This solution β is the starting solution in the next iteration of algorithm. Such a procedure allows for a possibility of increasing current solution's value (when new starting solution is sought), but it increases a chance to find global minimum. To prevent generating of recently considered solutions (making cycles), that solutions are recorded on a list of prohibited solutions, so-called "tabu" list (short-term memory).

A standard sequential tabu search algorithm can be written as follows.

Standard tabu search algorithm

Let $\pi \in \Pi$ be an initial solution;
pi^* - the best known solution;
T - tabu list;
$\pi^* \leftarrow \pi$;
Step 1
Generate the neighbourhood $\mathcal{N}(\pi)$ of the current solution π. Excluding from $\mathcal{N}(\pi)$ elements from tabu list T except $\beta \in \mathcal{N}(\pi)$ such that $C_{\max}(\beta) < C_{\max}(\pi^*)$;
Step 2 Find a solution $\delta \in \mathcal{N}(\pi)$ such, that
$C_{\max}(\delta) = min(C_{\max|}(\beta), \beta \in \mathcal{N}(\pi))$;
Step 3 **If** $C_{\max}(\delta) < C_{\max}(\pi^*)$ **then**
begin
$\pi^* \leftarrow \delta$;
Include δ to the list T;
$\pi \leftarrow \delta$;
end;
Step 4 **If** (*Stop condition* is true) **then** Stop;
else go to Step 1;

The *initial solution* π of the algorithm is found by the heuristic method NEH (Navaz, Enscore, Ham [10]).

The method of calculating neighbourhood, tabu list and stop condition are the basic elements of algorithm.

Let B_k $(k = 1, 2, \ldots, m)$ be the k-th block in permutation π, B_k^f and B_k^l the subblocks (see section 2). For job $j \in B_k^f$ by $N_k^f(j)$ let us denote a set of permutations created by moving job j to the beginning of block B_k (before the first job in block $\pi(f_k)$). Analogously, for job $j \in B_k^f$ by $N_k^l(j)$ let us denote a set of permutations created by moving job j to the end of the block B_k (after the last job in block $\pi(l_k)$). The *neighbourhood* of the solution π:

$$\mathcal{N}(\pi) = \bigcup_{j \in B_k} (N_k^f(j) \cup N_k^l(j)).$$

Tabu list is cyclic and consists of attributes of the latest considered solutions. Tabu list has permanently fixed length. If the list is not full, a new element is added to the list. If the list has its maximal length (so it's full) then the oldest element of the list is overwrited by the new one.

The algorithm stops (*Stop condition*) after *Max_iter* iterations.

Additionally, there is a backtracking mechanism applied in the algorithm (long-term memory). Some number of good solutions are recorded on backtracking list. Good solution - it means that the relative difference between this solution β and the best known (current) solution $\pi^*$is small or negative – less then ϵ:

$$\frac{C_{\max}(\beta) - C_{\max}(\pi^*)}{C_{\max}(\pi^*)} < \epsilon.$$

If there is no improvement of the best solution's objective function value after some number of iterations, algorithm jumps to the latest solution got from backtracking list (so the current solution π is overwrited by solution from the list). Current tabu list is overwrited too – algorithm got the tabu list connected with the backtracked solution from the backtracking list. So, on the backtracking list there are not only solutions, but tabu lists connected with solutions too.

4 Parallel concepts

The chosen model of parallel computing is the SIMD machine of p processors without shared memory – with time of communication between processors much longer than time of communication inside the process which is executing on one processor. The way of parallelism used here is concurrently executing some number of tabu search processes with broadcast the best solution of one processor to other processors when the new best solution is found. Processors has access to the same table of backtrack-jump solutions with tabu lists of each one. The backtracking list and the best known solution

π^* is stored by main processor which controls computing process.

Parallel tabu search algorithm written for a CREW PRAM model of computation is given below.

Parallel tabu search algorithm

parfor $j = 1...p$ (for each processor)
begin
Let $\pi \in \Pi$ be an initial solution for the first processor; For other processors, the initial solution is taken from the backtracking list (when they appears).
π^* - the best known solution; (local)
T - tabu list; (the local one too)
$\pi^* \leftarrow \pi$;
Step 1
Generate the neighourhood $\mathcal{N}(\pi)$ of the current solution π. From $\mathcal{N}(\pi)$ exclude elements from tabu list T except $\beta \in \mathcal{N}(\pi)$ such that $C_{\max}(\beta) < C_{\max}(\pi^*)$;
Step 2
For ($\beta \in \mathcal{N}(\pi)$, $\frac{C_{\max}(\beta) - C_{\max}(\pi^*)}{C_{\max}(\pi^*)} < \epsilon$)
Add β to the backtracking list by sending it to the processor which store a data.
Step 3
Find a solution $\delta \in \mathcal{N}(\pi)$ such that
$C_{\max}(\delta) = min(C_{\max|}(\beta), \beta \in \mathcal{N}(\pi))$;
Step 4
If $C_{\max}(\delta) < C_{\max}(\pi^*)$ **then**
begin
$\pi^* \leftarrow \delta$;
Send π^* and $C_{\max}(\pi^*)$ to the processor which store a data and take actual value of π^* and $C_{\max}(\pi^*)$ from this processor (if π^* is changed);
Include δ to the list T;
$\pi \leftarrow \delta$;
end;
Step 5
If (there were not any improvement of $C_{\max}(\pi^*)$ in last L iterations) **then**
begin
Take the new permutation π from backtracking list stored by the main processor.
Take the tabu list T for the permutation π from main processor.
end
Step 6
If (*Stop condition* is true) **then** Stop;
else go to Step 1;
end of parfor.

The frequency of communication between processors (broadcasting of π^*) is very important for this parallel algorithm performance. It must not be very often because of long time of communication between processors. In this implementation a processor is getting a new value of π^* only when it wants to broadcast its own π^* (so it exchanges and compares the best solutions with its own π^*).

The construction of parallel algorithm is done to minimize number of communication processes between processors and the main processor (processor which store a data). So the communication time is very short (comparing to the time of computations), especially in later stage of algorithm work (when found solution is close to the optimal one).

5. Computer simulations

The algorithms has been tested in several commonly used instances of various size and hardness levels:

a) 70 instances of 7 different sizes with 100, ..., 1000 operations ($n \times m$ = 20 × 5, 20 × 10, 20 × 20, 50 × 5, 50 × 10, 50 × 20, 100 × 5) due to Taillard [17], (from the OR-Library: http://mscmga.ms.ic.ac.uk/info.html),

b) 100 instances of 5 different sizes with 2000, ..., 10000 operations (200×5, 200×10, 200×20, 200×25, 200 × 50).

The computational results are presented in Table 1 and 2. We used the following parameter specifications in algorithms:

$\|T\|$	=	10	–	length of tabu list,
ϵ	=	0.25%	–	constant used in backtracking mechanism,
L	=	10	–	constant used in backtracking mechanism,
p	=	$1, 2, 4$	–	number of processors,

for 2000 iterations:

Max_iter	=	2000	–	for 1 processor implementation,
	=	1000	–	for 2 processors,
	=	500	–	for 4 processors.

for 4000 iterations:

Max_iter	=	4000	–	for 1 processor implementation,
	=	2000	–	for 2 processors,
	=	1000	–	for 4 processors.

All algorithms were implemented in Ada95 language and run on Sun Enterprise 4x400 MHz computer under

Solaris 7 operating system. Tasks of Ada95 language were executed in parallel as system threads.

The number of iterations of the algorithm is 2000 for one-processor implementation, 1000 for each of processor in the implementation for 2 processors and 500 for each of processor in the implementation for 4 processors (so we have the same complexity – the frequency of communication between processors is very rare so it hasn't any influence on complexity estimation). As we can see in the tables below, the results are better for parallel program. So we can say speedup is even greater then p, in a certain sense (four-processor parallel program needs less then 500 iterations to have the same results as sequential algorithm for 2000 iterations).

We are comparing solutions of our algorithm with the best known in literature approximate algorithm NEH (Navaz, Enscore, Ham [10]).

Below, because of their difficulty, are the results of Taillard benchmarks. The results of random tests were similar.

Table 1. Distance tabu search solutions and NEH compared to the best Taillard [17] solutions (2000 iterations).

$n \times m$	1 proc	2 proc	4 proc	NEH
20×5	0,96%	0,67%	0,45%	2,87%
20×10	3,03%	1,28%	1,41%	4,74%
20×20	2,02%	1,10%	1,05%	3,69%
50×5	0,33%	0,08%	0,15%	0,89%
50×10	2,86%	2,35%	2,25%	4,53%
50×20	3,71%	3,52%	3,19%	5,24%
100×5	0,25%	0,16%	0,12%	0,46%
generally	**1,88%**	**1,31%**	**1,23%**	**3,20%**

Table 2. Distance tabu search solutions and NEH compared to the best Taillard [17] solutions (4000 iterations).

$n \times m$	1 proc	2 proc	4 proc	NEH
20×5	0,96%	0,43%	0,42%	2,87%
20×10	2,84%	1,09%	1,30%	4,74%
20×20	1,82%	0,62%	0,62%	3,69%
50×5	0,33%	0,08%	0,14%	0,89%
50×10	2,81%	2,10%	1,81%	4,53%
50×20	3,49%	3,02%	2,86%	5,24%
100×5	0,25%	0,18%	0,09%	0,46%
generally	**1,79%**	**1,07%**	**1,03%**	**3,20%**

As we can see in Table 1. and Table 2., results of the parallel algorithm are the best for the large value of quotient n and m ($20 \times 5, 50 \times 5, 100 \times 5$). In such a case the size (length) of blocks is the most profitable for sequential and parallel algorithm performance. Besides, for 2000 iterations, improvement of results for parallel algorithm compared to sequential one was at the level of 30% for 2-processors implementation and at the level of 35% for 4-processors implementation. For 4000 iterations the difference between results of sequential and parallel algorithm was even greater - the value of the improvement was 40% for 2 processors and 42% for 4 processors, all parallel algorithms with the same number of iterations (as the sum of iterations on each processor) like sequential algorithm.

6 Conclusions

We discussed a new approach for the permutation of flow shop scheduling based on a backtrace-jump version of iterative improvement. The parallelism of algorithm makes performance of tabu search much better than the iterative improvement approach. The advantage is especially visible for large problems.

References

[1] Eikelder H., Aarts B., Verhoeven M., Aarts E., Sequential and parallel local search algorithms for job shop scheduling, in Voss S.(Ed.) Meta–heuristics. Kluwer Academic Publishers, Boston, 1999, pp. 359-371.

[2] Garey M.R., D.S. Johnson, R. Seti, The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research **1**, 1976, pp. 117-129.

[3] Glover F., Laguna M.,Tabu search, in Reeves C.R.,(Ed.), Modern Heuristic Techniques for Combinatorial Problems, McGraw-Hill, London, 1995, pp. 70-150.

[4] Grabowski J., A new algorithm of solving the flow-shop problem, Operations Research in Progress. D. Reidel Publishing Company, 1982, pp. 57-75.

[5] Grabowski J., J. Pempera, New block properties for the permutation flow-shop problem with application in TS. Journal of Oper. Res. Soc. **52**, 2001, pp. 210-220.

[6] Ignall E., L.E. Schrage, Application of the branch-and-bound technique to some flow–shop scheduling problems. Operations Research **13/3**, 1965, pp. 400-412.

[7] Ishibuchi H., S. Misaki, H. Tanaka, Modified Simulated Annealing Algorithms for the Flow Shop Sequencing Problem. European Journal of Operational Research **81**, 1995, pp. 388-398.

[8] Johnson S.M., Optimal two and three–stage production schedules with setup times included. Naval Research Logistic Quertely **1**, 1954, pp. 61-68.

[9] Lageweg B.J., J.K., Lenstra, A.H.G. Rinnooy Kan, A General Bouding Scheme for the Permutation Flow-Schop Problem, Operations Research **26**, 1978, pp. 53-67.

[10] Navaz M., E.E. Enscore Jr, and I. Ham, A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem, OMEGA **11/1**, 1983, pp. 91-95.

[11] Nowicki E., C. Smutnicki, A fast tabu search algorithm for the permutation flow-shop problem, European Journal of Operational Research **91**, 1996, pp. 160-175.

[12] Ogbu F., D. Smith, The Application of the Simulated Annealing Algorithm to the Solution of the $n/m/C_{max}$ Flowshop Problem, Comp. & Oper. Res. **17(3)**, 1990, pp. 243-253.

[13] Osman I., C. Potts, Simulated Annealing for Permutation Flow-Shop Scheduling. OMEGA **17(6)**, 1989, pp. 551-557.

[14] Reeves C., Improving the Efficiency of Tabu Search for Machine Sequencing Problems. Journal of Operational Research Society **44(4)**, 1993, pp. 375-382.

[15] Reeves C., A Genetic Algorithm for Flowshop Sequencing. Computers & Operations Research **22(1)**, 1995, pp. 5-13.

[16] Taillard E., Some efficient heuristic methods for the flow shop sequencing problem. European Journal of Operational Research **47(1)**, 1990, pp. 65-74.

[17] Taillard E., Benchmarks for basic scheduling problems. European Journal of Operational Research **64**, 1993, pp. 278-285.

[18] Talbi E.G., Hafdi Z., Geib J.M., Parallel tabu search for large optimization problems, in Voss S.(Ed.) Meta–heuristics. Kluwer Academic Publishers, Boston, 1999, pp. 345-358.

[19] Wodecki M., Bożejko W., Solving the flow shop problem by parallel simulated annealing. Lecture Notes in Computer Science **2328**, Springer Verlag, Heidelberg, 2002, pp. 236-247.