Solving the flow shop problem by parallel simulated annealing

Mieczysław Wodecki[1] and Wojciech Bożejko[2]

[1] Institute of Computer Science, University of Wrocław,
Przesmyckiego 20, 51-151 Wrocław, Poland,
mwd@ii.uni.wroc.pl

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

Abstract. In this paper we present two simulated annealing algorithms (sequential and parallel) for the permutation flow shop sequencing problem with the objective of minimizing the flowtime. We propose a neighbourhood using the so-called blocks of jobs on a critical path and specific accepting function. We also use the lower bound of cost function. By computer simulations on Taillard [17] and other random problems, it is shown that the performance of the proposed algorithms is comparable with the random heuristic technique discussed in literature. The proposed properties can be applied in any local search procedures.

Keywords: simulated annealing, flow shop sequencing, critical path's blocks, parallel algorithm.

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 [7] gives an $O(n \log n)$ algorithm for $F|2|C_{max}$ and Garey, Johnson & Seti [2] show that $F|3|C_{max}$ is strongly NP-hard. The best available branch and bound algorithms are those of Ignall, Schrage [5], Lageweg, Lenstra & Rinnooy Kan [9] and Grabowski [3]. 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], Reeves [14] and Nowicki, Smutnicki [11]. Simulated annealing algorithms are proposed by Osman, Potts [13], Ogbu, Smith [12] and Ishibushi, Misaki & Tanaka [6]. Reeves

[15] proposes a genetic algorithm which uses the reorder crossover. In our paper we will present sequential and parallel simulated annealing algorithm for the permutation flow shop problem.

Section 2 describes a solution space for the permutation flow shop problem. Sections 3 and 4 describes sequential and parallel annealing algorithms. Section 6 includes results.

2 Problem definition and notation

The flow shop problem can be defined as follows, using the notation by Nowicki, Smutnicki [11] and Grabowski, Pempera [4] 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 times 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).

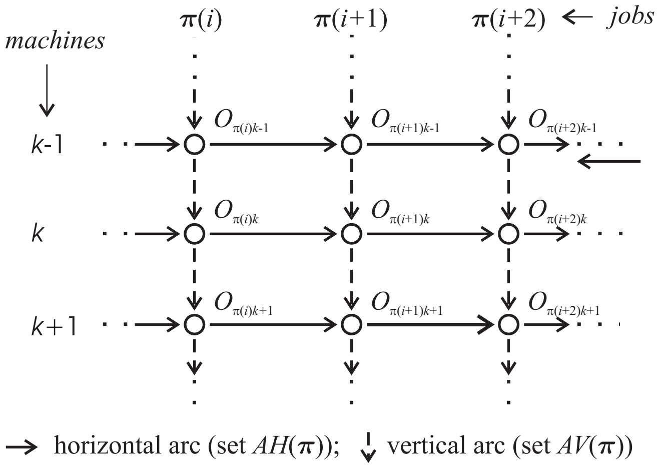

Fig. 1. Digraph $D(\pi)$.

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 [3]).
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 $\pi(l_k)$, $k = 1, 2, ..., m$, (Fig. 2).

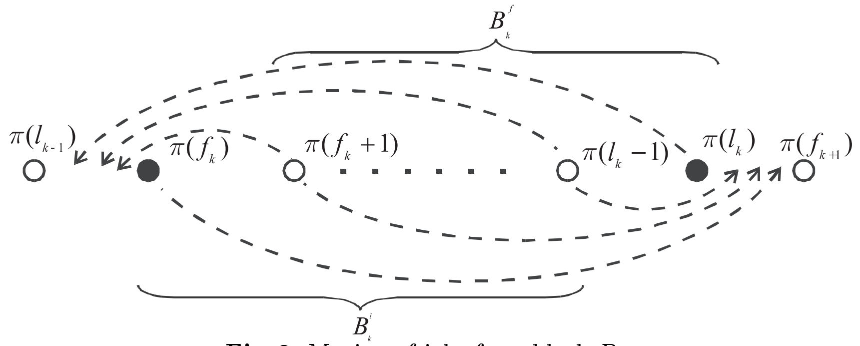

Fig. 2. Moving of jobs from block B_k.

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 [3]).
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:

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

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 $C_{\max}(\beta) \geqslant C_{\max}(\pi) + \Delta_k^l(j)$ or $C_{\max}(\pi) + \Delta_k^l(j)$. Thus the values $\Delta_k^l(j)$ and $\Delta_k^l(j)$ can be used to decide which job should be moved.

3 Simulated annealing method

Simulated annealing is a stochastic heuristic method for optimization problems, motivated by the physical process of crystallization. The method was first applied to combinatorial problems by Kirkpatrick et al. [8] and Černý [1].

In each iteration of simulated annealing a random perturbation is made to the current solution $\pi \in \Pi$, giving rise to the set $\mathcal{N}(\pi)$ of *neighbours*. A neighbour $\beta \in \mathcal{N}(\pi)$ is accepted as the next configuration with probability function $\Psi_t(\pi, \beta)$. The $\Psi_t(\pi,\beta)$ is known as *accepting function* and depends on control parameter t (*temperature*). Its value changes at suitably chosen intervals. In practice the function $\Psi_t(\pi,\beta)$ is chosen in this way that solutions corresponding to large increases in cost have a small probability of being accepted, whereas solutions corresponding to small increases in cost have a larger probability of being accepted. A standard simulated annealing algorithm can be written as follows.

Standard simulated annealing algorithm

Let $\pi \in \Pi$ be an initial solution; $\pi^* \leftarrow \pi$;
repeat
  while $i \leqslant L$ **do**
  begin
    $i \leftarrow i + 1$;
    Randomly generate a solution β
    from the neighbourhood $\mathcal{N}(\pi)$ of the current solution π;
    if $C_{max}(\beta) < C_{max}(\pi^*)$ **then** $\pi^* \leftarrow \beta$;
    if $C_{max}(\delta) < C_{max}(\pi)$ **then** $\pi \leftarrow \delta$ **else**
      if $\Psi_t(\pi, \beta) > random[0, 1)$ **then** $\pi \leftarrow \beta$
  end; $\{i\}$
  $i \leftarrow 0$; modify control parameter t;
until STOP;

In the algorithm π^* is the best solution and L – the number of iterations for fixed value of parameter t.

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

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:

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

We are proposing a new probability accepting function:

$$\Psi_t(\pi, \beta) = \exp[(-LB(\beta) + C_{max}(\pi^*)) \frac{-2\ln(t)}{C_{max}(\pi^*)}],$$

where $LB(\beta)$ is lower bound on the value $C_{\max}(\beta)$, theorem 2. The initial value of control parameter $t \leftarrow t_0$, $(0 < t \leqslant 1)$, where t_0 is accepting probability of the solution which is worse by half compared to the best solution π^*. To modify parameter t we use a geometric decreasing scheme: $t \leftarrow t * \mathbf{a}$, $(0 < \mathbf{a} < 1)$. If there is no improvement of best solution π^* after *T_iter* iterations, then $t \leftarrow t_0$.

The algorithm stops (STOP) after *Max_iter* iterations.

4 Parallel concepts

The chosen model of parallel computing is the SIMD machine of processors without shared memory – with time of communication between processors much longer then time of communication inside the process which is executing on one processor. There are two ways of parallelism used here. One method is simultaneous independent search – concurrently executing some number of independent simulated annealing algorithms without any communication between them and selecting the best solution from solutions obtained by all processes. The other method is to broadcast the best solution of one processor to the other processors when the new best solution is found.

Parallel simulated annealing algorithm with broadcast

Let $\pi \in \Pi$ be an initial solution (the same for each of P processors).
$\pi^* \leftarrow \pi$; (all variables are local ones)
parfor $j = 1, 2, ..., P$
  while $i \leqslant L$ **do**
  begin
    $i \leftarrow i + 1$;
    Randomly generate a solution β
    from the neighbourhood $\mathcal{N}(\pi)$ of the current solution π;
    if $C_{max}(\beta) < C_{max}(\pi^*)$ **then**
      $\pi^* \leftarrow \beta$; broadcast π^* to other processors with
      comparing to others ones (exchanging π^*);
    if $C_{max}(\delta) < C_{max}(\pi)$ **then** $\pi \leftarrow \delta$ **else**
      if $\Psi_t(\pi, \beta) > random[0, 1)$ **then** $\pi \leftarrow \beta$
  end; $\{i\}$
  $i \leftarrow 0$; modify control parameter t
end {parfor}

The frequency of communication between processors (broadcasting of π^*) is very important for this parallel algorithm performance. It must not be very often (long time of communication between processors!). In this implementation 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^*).

5 Computer simulations

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

a) 50 instances of 12 different sizes with 100, ... , 500 operations ($n \times m = 20 \times 5$, 20×10, 20×20, 50×5, 50×10) 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:

$t_0 = 0.5$ – initial value of control parameter,
$\mathbf{a} = 0.98$ – constant in control parameter formula,
$L = n$ – number of iterations for fixed parameter t,
$T\_iter = 10$ – number of iterations without improvement of the best solution after which parameter t is set to t_0.

All algorithms were implemented in Ada95 language and run on Sun Enterprise 4x400 MHz computer under Solaris 7 operating system. The maximal number of iterations *Max_iter* is 200 for one-processor implementation and 50 for each of processor in the implementation for 4 processors (so we have the same complexity, the value of speedup is 4 – 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 speedup is even greater then 4, in a certain sense (parallel program needs less then 50 iterations to have the same results as sequential algorithm for 200 iterations).

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

There are the results of Taillard benchmarks below. The results of random tests were similar.

Table 1. Distance simulated annealing solution and NEH compared to the best Taillard [17] solution.

$n \times m$	1 processor	4 processors independent	4 processors with broadcasting	NEH
20×5	0,87%	0,64%	0,62%	2,87%
20×10	2,29%	1,82%	1,70%	4,74%
20×20	1,94%	1,91%	1,82%	3,69%
50×5	0,13%	0,08%	0,13%	0,89%
50×10	1,87%	1,31%	0,92%	4,53%
50×20	2,75%	2,32%	2,29%	5,24%
100×5	0,0011%	0,0003%	0,0003%	0,46%
generally	**1,41%**	**1,15%**	**1,07%**	**3,20%**

Table 2. Improvement to NEH solution.

$n \times m$	1 processor	4 processors independent	4 processors with broadcasting
20×5	1,94%	2,17%	2,19%
20×10	2,34%	2,79%	2,90%
20×20	1,69%	1,72%	1,80%
50×5	0,75%	0,80%	0,75%
50×10	2,54%	3,08%	3,45%
50×20	2,37%	2,77%	2,80%
100 ×5	0,46%	0,46%	0,46%
generally	**1,74%**	**1,98%**	**2,07%**

As we can see in Table 1. and Table 2., results of the parallel algorithm are 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, improvement of solution value for parallel algorithm compared to sequential one was at the level of 18%. Parallel algorithm with broadcasting of the upper bound value was better than the sequential one at the level of 24%, all parallel algorithms with the same number of iterations (as the sum of iterations on each processor) like sequential algorithm.

6 Conclusions

We have discussed a new approach to the permutation of flow shop scheduling based on a randomization version of iterative improvement. The probabilistic element of the algorithm makes simulated annealing much better than the iterative improvement approach. The advantage is especially visible for large problems. As compared to the sequential algorithm parallelization increases the quality of obtained solutions.

References

1. Černý V., Thermodynamical approach to travelling salesman problem: An efficient simulation algorithm. J. Optim. Theory Appl. **45** (1985) 41-51.
2. Garey M.R., D.S. Johnson, R. Seti, The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research **1** (1976) 117-129.
3. Grabowski J., A new algorithm of solving the flow-shop problem, Operations Research in Progress. D. Reidel Publishing Company (1982) 57-75.
4. Grabowski J., J. Pempera, New block properties for the permutation flow-shop problem with application in TS. Journal of Oper. Res. Soc. **52** (2001) 210-220.
5. Ignall E., L.E. Schrage, Application of the branch-and-bound technique to some flow-shop scheduling problems. Operations Research **13/3** (1965) 400-412.
6. Ishibuchi H., S. Misaki, H. Tanaka, Modified Simulated Annealing Algorithms for the Flow Shop Sequencing Problem. Eur. Jour. of Oper. Res. **81** (1995) 388-398.
7. Johnson S.M., Optimal two and three-stage production schedules with setup times included. Naval Research Logistic Quertely **1** (1954) 61-68.
8. Kirkpatrick S., C.D. Gellat, M.P. Vecchi, Optimization by simulated annealing. Science **220** (1983) 671-680.
9. Lageweg B.J., J.K., Lenstra, A.H.G. Rinnooy Kan, A General Bouding Scheme for the Permutation Flow-Schop Problem, Opns. Res. **26** (1978) 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) 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) 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) 243-253.
13. Osman I., C. Potts, Simulated Annealing for Permutation Flow-Shop Scheduling. OMEGA **17(6)** (1989) 551-557.
14. Reeves C., Improving the Efficiency of Tabu Search for Machine Sequencing Problems. Journal of Operational Research Society **44(4)** (1993) 375-382.
15. Reeves C., A Genetic Algorithm for Flowshop Sequencing. Computers & Operations Research **22(1)** (1995) 5-13.
16. Taillard E., Some efficient heuristic methods for the flow shop sequencing problem. European Journal of Operational Research **47(1)** (1990) 65-74.
17. Taillard E., Benchmarks for basic scheduling problems. European Journal of Operational Research **64** (1993) 278-285.