Parallel tabu search method approach for a very difficult permutation scheduling problems

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

There are very many issues, where scheduling can be applied, in computer systems (single and multiprocessors) as well as in production systems. Scheduling problems belongs in most cases to NP-hard class. For most of classical scheduling problems, published in last 10 years (for example benchmarks of Taillard [14] for the flow shop problem), there are still no optimal solutions. In this paper we propose very effective method of construct parallel algorithms based on tabu search metaheuristic. We apply block properties, which enable parallel algorithm to distribute calculations and reduce communication between processors. Algorithms are implemented in Ada95 and MPI.[*]

1. Introduction

We can see the process of flow jobs through machines (processors) in many practical problems of scheduling, in computer systems as well as in production systems. Optimization of this process is based on finding some optimal permutation in the set of all permutations of jobs. Many methods of algorithms construction consist in looking through (directly or indirectly) all or a part of set of feasible solutions. Such a mechanism is based on generating from current (base) solution next solution, or a set of solutions (so called neighborhood), from which the best solution is chosen. This solution is the base solution in the next iteration. Such a mechanism can be met (among others) in Branch and Bound (B&B) method and in many other algorithms which consist of improving the solution, as well as in the best (nowadays) approximate algorithms, metaheuristics: tabu search and simulated annealing methods. Quality of these algorithm's solutions depends on: number of iteration, the method of neighborhood describing and its reviewing. The time of computations can be shorten by its realization in multiprocessor environment. Unfortunately parallelization of the sequential algorithms directly (for example by using parallel compiler) does not give satisfactory speedup. In this paper we propose some new elements of parallel local search algorithm. Partitioning solution (permutation) into blocks (subpermutations) enables to decrease neighborhood size and its division into separated subsets. It makes possible its generating and reviewing independently.

Computer simulations for some classical scheduling problems (for example flow shop, single machine total weighted tardiness) allow getting very good solutions and superlinear speedup effect. This work is continuation of [2], [3] and [18] .

[*]The work was supported by KBN Poland, within the grant No. T11A01624

2. Blocks in permutation

Let $N = \{1, 2, \ldots, n\}$ be a set of n elements (enumerated by numbers from 1 to n). By Φ_n we mean a set of all permutations of set N.

Permutational optimization problem (POP) consists in determine such a optimal element π^*, that

$$F(\pi^*) = \min\{F(\pi) : \pi \in \Pi\},$$

where $\Pi \subseteq \Phi_n$ is a set of feasible solutions and $F : \Pi \to R^+ \cup 0$ is a goal function. For every permutation $\pi \in \Pi$, we call a ***move*** transformation

$$r : \pi \to r(\pi) \in \Pi.$$

Example 1. Let $i, j \in \{1, 2, \ldots, n\}$ be a par of positions in the permutation π. Among many types of moves considered in literature, two of them appear prominently:

1. Insert move, in which the job $\pi(i)$ is removed from the position i, and next insert on the position j in π. Thus

move generates the following permutation:

$$(\pi(1), \pi(2), \ldots, \pi(i-1), \pi(i+1), \ldots, \pi(j),$$

$$\pi(i), \pi(j+1), \ldots, \pi(n)), \quad \text{ if } i < j, \text{ and}$$

$$(\pi(1), \pi(2), \ldots, \pi(j-1), \pi(i), \pi(j), \ldots, \pi(i-1),$$

$$\pi(i+1), \ldots, \pi(n)), \quad \text{ if } i > j.$$

2. Swap move, in which the job $\pi(i)$ and $\pi(j)$ are swapped between some positions i and j in π. Thus move generates the permutation:

$$(\pi(1), \pi(2), \ldots, \pi(i-1), \pi(j), \pi(i+1), \ldots, \pi(j-1),$$

$$\pi(i), \pi(j+1), \ldots, \pi(n)).$$

Let $M(\pi)$ be a fixed set of moves in permutation $\pi \in \Pi$, and

$$M(\Pi) = \bigcup_{\pi \in \Pi} M(\pi)$$

be a set of all moves. We call a ***neighborhood*** $N(\pi)$ of permutation π as a set of permutations generated by moves from $M(\pi)$, i.e.

$$N(\pi) = \{r(\pi) : \ r \in M(\pi)\}.$$

Of course $N(\pi) \subseteq \Pi$.
Let

$$B(\pi) = [\pi^1, \pi^2, \ \ldots \ , \pi^k]$$

be a partition of permutation $\pi \in \Pi$ into k subpermutations. By $S(\pi^i)$ we represent a set of elements of the subpermutation $\pi^i \in B(\pi)$.

From partition properties we now, that π^i, $i = 1, 2, \ \ldots \ , k$ is subpermutation of π, i.e. exists positions $1 \leq b^i \leq e^i \leq n$ such, that $\pi^i(j) = \pi(j)$, $j = b^i, b^i + 1, \ \ldots \ , e^i$ and

$$S(\pi^i) \cap S(\pi^j) = \emptyset, \quad i, j = 1, 2, \ \ldots \ , n, \ \ i \neq j.$$

Partition $B(\pi)$ of permutation π generates also partition of the set of moves $M(\pi)$.

We call ***internal move*** such a move $r \in M(\pi)$ for subpermutation $\pi^i \in B(\pi)$, which changes order of elements only in subpermutation π^i. Precisely, if $\delta = r(\pi)$, there exists partition $B(\delta) = [\delta^1, \delta^2, \ \ldots \ , \delta^k]$ such, that

$$\pi^j = \delta^j, \ j = 1, 2, \ \ldots \ , k, \ j \neq i \text{ and } S(\pi^i) = S(\delta^i).$$

By $MI(\pi^i)$ we represent a set of internal moves for subpermutation π^i.

We call a move $r \in M(\pi)$ ***external move***, if at least one element of subpermutation $\pi^i \in B(\pi)$ is moved by r before the first or after the last element of π^i. By $MO(\pi^i)$ we represent a set of all external moves of subpermutation $\pi^i \in B(\pi)$. It is simple to proof, that $MI(\pi^i) \cap MO(\pi^i) = \emptyset$, $i = 1, 2, \ \ldots \ , k$. Moreover, let for permutation π

$$MI(\pi) = \bigcup\nolimits_{\pi^i \in B} MI(\pi^i) \text{ and}$$

$$MO(\pi) = \bigcup\nolimits_{\pi^i \in B} MO(\pi^i).$$

Theorem 1. If $M(\pi)$ is a set of moves of the permutation $\pi \in \Pi$, for any partition $B(\pi) = [\pi^1, \pi^2, \ \ldots \ , \pi^k]$ we have:

$$M(\pi) = \bigcup_{\pi^i \in B(\pi)} (MI(\pi^i) \cup MO(\pi^i)).$$

Proof. From the definition and properties of moves.

We can describe the neighborhood of permutation π by

$$N(\pi) = \{r(\pi) : \ r \in M(\pi)\} =$$

$$\{r(\pi) : \ \ r \in \bigcup_{\pi^i \in B(\pi)} (MI(\pi^i) \cup MO(\pi^i))\} \qquad (1)$$

Let Π be a set of feasible solutions, $M(\Pi)$ a set of moves and F goal function, for some POP. Further, we take under consideration any partition $B(\pi) = [\pi^1, \pi^2, \ \ldots \ , \pi^k]$ of permutation $\pi \in \Pi$. Subpermutation $\pi^i \in B(\pi)$ is called ***block***, if

$$\forall r \in MI(\pi^i), \quad F(r(\pi)) \geq F(\pi).$$

Therefore every internal move generates the permutation with goal function not less than $F(\pi)$. If every subpermutation from partition $B(\pi)$ is block we say that $B(\pi)$ is ***partition of permutation*** π ***into blocks.***

Example 2. Samples of problems with blocking partitioning property:

1. Flow shop problem – subsets of jobs from the same machine on critical path are blocks, see. Nowicki and Smutnicki [11], Grabowski and Pempera [6], Grabowski and Wodecki [7].

2. Job shop problem – subsets of jobs from the same machine on critical path are blocks, see. Nowicki and Smutnicki [12], Grabowski and Wodecki [8] .

3. Single machine total weighted tardiness problem – subsets of jobs made on time and made after deadline are blocks, see. Bożejko and Wodecki [4].

Theorem 2. Let $B(\pi)$ be some partition of permutation $\pi \in \Pi$ into blocks. If

$$F(r(\pi)) = \min\{F(\upsilon(\pi)) : \ \upsilon \in M(\pi)\} \text{ and}$$

$$F(r(\pi)) < F(\pi^*), \text{ then } r \in MO(\pi).$$

Proof. From the definition and properties of the moves. Therefore, every element of neighborhood $N(\pi)$ of permutation π, which goal function is less than $F(\pi)$, is generated by some external move from the set $MO(\pi)$. Solving POP we can skip internal moves, i.e. moves from the set $MI(\pi)$. If

$$NO(\pi^i) = \{r(\pi) : \ \ r \in MO(\pi^i)\},$$

then (1) has form

$$N(\pi) = \bigcup_{\pi^i \in B(\pi)} NO(\pi^i). \qquad (2)$$

Formula (2) considerably decreases number of elements of the neighborhood and divides it into sum of separated subsets. In parallel implementation it enables to use, in natural way, many processors and speed up computations.

So, there is natural possibility to use $|B(\pi)|$ independently working processors to generate the neighborhood of solution π. Then every processor finds the best solutions of its own subset of solutions.

Example 3. For neighborhood generated by popular moves: insert or swap (Example 1, also see [6], [7], [11]) and partition into blocks $B(\pi) = [\pi^1, \pi^2, \ \dots \ , \pi^k]$, number of elements from neighborhood (2) is

$$|N(\pi)| = \sum_{i=1}^{k} (n - |\pi^i|) \cdot |\pi^i| =$$

$$= n \cdot \sum_{i=1}^{k} |\pi^i| - \sum_{i=1}^{k} |\pi^i|^2 = n^2 - \sum_{i=1}^{k} |\pi^i|^2.$$

Its minimal value is achieved, when there is only one block. Then, neighborhood is the empty set. Generally, partitions with small number of blocks are good from computational point of view.

3. Parallel tabu search method

The tabu search (TS) method is a metaheuristic approach designed to find a near-optimal solution of combinatorial optimization problems. The basic version of TS starts from an initial solution x^0. The elementary step of the method performs, for a given solution x^i, a search through the neighborhood $N(x^i)$ of x^i. The neighborhood $N(x^i)$ is defined by a move performed from x^i. The move transforms a solution into another solution. The aim of this elementary search is to find in $N(x^i)$ a solution x^{i+1} with the lowest cost functions. Then the search repeats from the best solutions found, as a new starting solution, and the process is continued. In order to avoid cycling, becoming trapped to a local optimum, and more general to conduct the search in "good regions" of the solution space, a memory of the search history is introduced. Among many classes of the memory introduced for tabu search (see. Glover [9]), the most frequently used is the short term memory, called the tabu list. This list memorizes, for a chosen span of time, selected attributes of these solutions or moves.

There are two basic types of tabu search parallelization discussed in literature. The first one, called single-walk, is based on neighborhood decomposition onto concurrent working processors. Received solutions are exactly the same as in sequential algorithm, but computing time is shorter. Aarts and Verhoeven [1] make the distinction between single-step and multiple-step parallelism within this type. In the case of single-step implementations, neighbors are searched and evaluated in parallel after neighborhood partitioning. The algorithm subsequently selects and performs one single move. In multiple-step parallelizations, a sequence of consecutive moves in the neighborhood is made simultaneously.

The second type of parallelization, called multiple–walk type, is based on concurrent working tabu search threads, running on different processors. There are dwo sub-types of this parallelizm: independent search, where there is no communication between threads, and cooperative search, with exchanging e.g. the best known solution found of the thread. Classification of multiple-walk tabu search algorithm was created by Voss in [17]. The first parallel implementations of tabu search based on this type of strategy seem to concern the quadratic assignment problem and job shop scheduling Taillard [15, 16].

We propose a hybrid type of tabu search parallelism in this paper. We use blocks to partitioning the neighborhood of the solution. Such an algorithm PSTS (parallel synchronous tabu search) is classical parallel single-walk, single-step tabu search (Fig. 1). Proposed new algorithm PATS (parallel asynchronous tabu search) uses special list $\alpha = (\alpha_1, \alpha_2, ..., \alpha_t)$ of current solutions instead of one current solution in classical tabu. Length of list α is permanently equal t. Such a parallel algorithm is based on two-level parallelism. One level is based on concurrently explored solutions from the list α by parallel working processors. These solutions are explored concurrently by block partitioning to find the best representative (solution) of each block, and added to list α (Fig. 2), and this is the second level of parallelism, made by another group of parallel working processors. Of course we also use a tabu list, to prevent generating solutions on list α serially. The second level of parallelism does not have any influence on results of computations (only speed up), but the first level of parallelism does. Additionally, in most parallel computing systems, we don't know the order of solutions (representatives) come on list α from processors – so such an algorithm is not deterministic one.

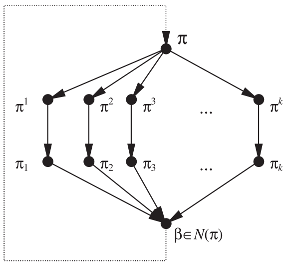

Figure 1. Skeleton of PSTS algorithm.

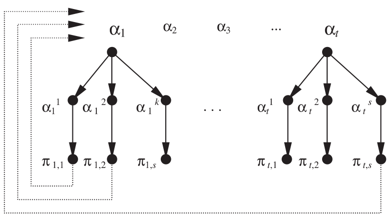

Figure 2. Skeleton of PATS algorithm.

A parallel tabu search algorithm written for the EREW PRAM model of parallel computations is given below:

Master processor

- Commission slave processors on level 1 to determine representatives for permutations of list α in parallel;
- get representatives from slave processors level 2 (asynchronously, when they come);
- reject representatives forbidden by tabu list, unless its goal value is less than the best known;
- add representatives to α list (instead the worst element of α, if the length of list is maximal fixed).

Slave processors on level 1

- Get permutation α_i to explore from list α of master processor;
- partition permutation α_i into blocks (compute critical path);
- commission slave processors on level 2 to determine representatives for each block of α_i in parallel.

Slave processors on level 2

- Get block α_i^j of permutation α_i from slave processor on level 1;
- determine representative (permutation $\pi_{t,j}$) of neighborhood generated for this block;
- send $\pi_{t,j}$ to master processor.

If the length of list $|\alpha|$ equals 1, we obtain synchronous PSTS algorithm out of foregoing scheme. For $|\alpha| > 1$ we obtain asynchronous PATS algorithm, because slave processors works independently and asynchronously.

4. Parallel tabu algorithm for the permutation flow shop scheduling problem

In this chapter we preset short description of the flow shop problem and implementation draft of the parallel algorithm which solve this problem.

4.1. Permutation flow shop problem

The flow shop problem can be defined as follows, using the notation of Nowicki and Smutnicki [11], and Grabowski and Wodecki [7].
There are a set of n jobs $J = \{1, 2, \ldots, n\}$ and 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 π =($\pi(1)$, $\pi(2)$, $\ldots$,$\pi(n)$) 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 $C_{max}(\pi)$=$C_{\pi(n)m}$.

Permutational flow shop problem amount to determine optimal permutation (with minimal goal function value) in the set of all permutations Π.

In [6], [7], [11] we can find, that for the insert and swap moves blocks are sequences of the jobs made on the same machine. So it is possible to apply the method of parallelization presented in previous chapter.

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.

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 in π is decomposed into subsequences $\pi^1, \pi^2, \ldots, \pi^m$, where:

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

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

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

In other words, π^k is a maximal subsequence of the critical path, which contains operations processed on the same machine. For insert and swap moves subsequences $\pi^1, \pi^2, \ldots, \pi^m$ are blocks. Operations $\pi(f_k)$ and $\pi(l_k)$ in π^k are called the *first* and *last* ones, respectively.

Theorem 3. [Grabowski [5]]
Let $G(\pi)$ be a graph with blocks π^k, $k = 1, 2, \ldots, m$. 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 $j \in \pi^k$ precedes job $\pi(f_k)$, for some $k = 2, \ldots, m$, or

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

Theorem 3 gives the necessary condition to obtain a permutation such that $C_{max}(\omega) < C_{max}(\pi)$.

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

4.2. Computer simulations

The algorithm was implemented in Ada95 language and run on 4 – processors Sun Enterprise 4x400 MHz under Solaris 7 operating system. Tasks of Ada95 language were executed in parallel as system threads. Tests ware based on 70 instances with 100,...,1000 operations (n×m=20×5, 20×10, 20×20, 50×5, 50×10, 50×20, 100×5) due to Taillard [14], taken from the OR-Library [13]. The results ware compared to reference Taillard ones (optimal or the best known), taken from this library. Starting solutions were taken from quick approximate algorithm NEH (Navaz, Enscore and Ham, [10]). For all tested algorithms, the number of iterations was counted as a sum of iterations on processors. For example, for 1000 iterations, 4 - processor implementations makes 250 iterations on every of 4 processors.

Table 1. Distance of parallel synchronous taboo search (PSTS) solutions and NEH compared to the best Taillard [14] solution (1000 iterations).

$n \times m$	PSTS algorithm	NEH algorithm
20×5	0,92%	2,87%
20×10	1,98%	4,74%
20×20	2,68%	3,69%
50×5	0,39%	0,89%
50×10	1,86%	4,53%
50×20	2,57%	5,24%
100×5	0,14%	0,46%
average	**1,51%**	**3,20%**

As we can see in Table 1 and Table 2, results of the algorithms 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 1000 iterations, improvement of results for parallel PATS algorithm compared to the PSTS (which has the same results as sequential tabu search) was at the level of 14% for 1-processors implementation (by the advantage of the list α), at the level of 58% for 4 and 8-processors implementation and even at the level of 64% for 12-processors implementation, all algorithms with the same number of iterations (as the sum of iterations on each processor) and comparable costs (product of number of processors and computation time).

Table 2. Distances of parallel asynchronous taboo search (PATS) solutions compared to the best Taillard [14] solution (1000 iterations).

$n \times m$	1*	4	8	12
20×5	0,84%	0,19%	0,24%	0,05%
20×10	1,59%	0,89%	0,83%	0,77%
20×20	1,10%	0,62%	0,62%	0,63%
50×5	0,25%	0,01%	0,02%	0,03%
50×10	2,34%	0,87%	0,76%	0,63%
50×20	2,81%	1,81%	1,89%	1,67%
100×5	0,15%	0,01%	0,04%	0,04%
average	**1,30%**	**0,63%**	**0,63%**	**0,55%**

* number of processors

5. Conclusions

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

References

[1] Aarts E.H.L., Verhoeven M., *Local search, in Annotated bibliographies in Combinatorial Optimization* (Dell'Amico M., Maffioli F., Martello S., eds.), Wiley and Sons, Chichester, 1997.

[2] Bożejko W., Wodecki M., "Solving the flow shop problem by parallel tabu search", *IEEE Computer Society*, PR01730 ISBN 0-7695-1730-7, 2002, pp. 189-194.

[3] Bożejko W., Wodecki M., "Parallel algorithm for some single machine scheduling problems", *Automatics*, vol. 134, 2002, pp. 81-90.

[4] Bożejko W. Wodecki M.: "A block approach for single machine total weighted tardiness problem", (to appear).

[5] Grabowski J., "Generalized problems of operations sequencing in the discrete production systems", (Polish), *Monographs*, 9, Scientific Papers of the Institute of Technical Cybernetics of Wrcaw Technical Univercity, 1979.

[6] Grabowski J., Pempera J., "New block properties for the permutation flow-shop problem with application in TS", *Journal of the Operational Research Society*, 52, 2001, pp. 210-220.

[7] Grabowski J., Wodecki M., "A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion", *Computers and Operations Research*, 31, 2004, pp. 1891-1909.

[8] Grabowski J., Wodecki M., "A very fast tabu search algorithm for job shop problem", In: Rego C., Alidaee B., eds, *Adaptive memory and evolution: tabu search and scatter search*, Kluwer Academic Publishers, Dordrecht, 2004 (in print).

[9] Glover F., Laguna M., *Tabu Search*, Kluwer Academic Publishers, Boston, 1997.

[10] Nawaz M., Enscore E.E.Jr, Ham I., "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem", *OMEGA International Journal of Management Science*, 11, 1983, pp. 91-95.

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

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

[13] OR-Library: http://mscmga.ms.ic.ac.uk/info.html

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

[15] Taillard E., "Parallel taboo search techniques for the job shopscheduling problem problem", *ORSA Journal on Computing*, 6, 1994, pp. 108-117.

[16] Taillard E., "Robust taboo search for the quadratic assignment problem", *Parallel Computing*, 7, 1991, pp. 443-455.

[17] Voss S., *Tabu search: Applications and prospects, in Network Optimization Problems* (Du D.Z., Pardalos P.M., eds.), World Scientific, 1993.

[18] Wodecki M., Bożejko W., "Solving the flow shop problem by parallel simulated annealing", *LNCS*, No. 2328, Springer Verlag, 2002, pp. 236-247.