A Hybrid Evolutionary Algorithm for some Discrete Optimization 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

Discrete optimization methods are applied in time-dependent systems where there are problems of production management and job's scheduling. One can encounter such problems in preparing travel itineraries for tourists, in optimal ways (e.g. traveling salesman's way), schedule planning and in expert systems connected with taking optimal decisions. Many of these problems amount to determining optimal scheduling (permutation of some objects) and usually they are NP-hard. They have also irregular goal functions and very many local minima. Classic heuristic algorithms (tabu search, simulated annealing and genetic algorithm) quickly converge to some local minimum and diversification of the search process is difficult. In this paper we present a hybrid evolutionary algorithm for solving permutation optimization problems. It consists in testing feasible solutions, which are local minima. *

1 Introduction

Most of the problems connected with recommending systems and optima planning are also pure discrete optimization problems. We present a general evolution method approach that can be used to find approximate solutions of the hard optimization problems (*OP* problems). Let Π be a set of all permutations of elements $1, 2, \ldots, n$, and $F : \pi \rightarrow \mathrm{R}^{+},\ \pi \in \Pi$. The problem consists in determining optimal permutation (with minimal or maximal value of the goal function F) in the solution space Π. Some representative examples of permutation problems are: the traveling salesman problem, the assignment problem, and the single and multi-machine scheduling problems. Although these problems have simple formulations, they are very troublesome, because in most cases they belong to the NP-hard problems class. By their very nature, the *OP* have a huge number of various local optima. Therefore, to solve these problems approximate methods are mainly used. Construction algorithms or classic local optimization algorithms do not always allow obtaining good results. These algorithms usually finish calculations after finding a few local optima. So nowadays many approaches, not so "sensitive" to detecting local optima – especially artificial intelligence methods, are applied to solve *OP*.

In this paper we present a method of the algorithm's construction for solving *OP*, consisting in determining and researching of local minima. This method is based on the following observation. If there are the same elements in some positions in several permutations, which are local minima, then these elements are in the same position in the optimal solution.

The basic idea is to start with an initial population (any subset of the solution space). Next, for every element of the population, a local optimization algorithm is applied (e.g. descending search algorithm) to determine a local minimum. In this way we obtain a set of permutations – local minima. If there is an element which is in the same position in several permutations, then it is fixed in this position in the permutation, and other positions and elements of permutations are still free. A new population (a set of permutations) is generated by drawing free elements in free positions (because there are fixed elements in fixed positions). After determining a set of local minima (for the new population) we can increase the number of fixed elements. To prevent finishing of the algorithm's work after executing some number of iterations (when all positions are fixed and there is nothing left to draw), in every iteration "the oldest" fixed elements are set as free.

The proposed method is especially helpful in solving big instances of very hard problems with irregular goal functions. One can encounter such problems, among others, in very efficient strategies of control of the discrete production Just-In-Time systems which are often elements of production's recommending systems.

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

The paper is organized as follows. In the next section we introduce the notation, elements of evolutionary algorithm and local search algorithms. There are results of computational experiments for two classical strong NP-hard problems of scheduling in Section 3. Obtained results are compared with the best known from literature. There are conclusions in Section 4.

2 Hybrid Evolutionary Algorithm

Let Π be a set of all permutations of elements from the set $N = \{1, 2, \ldots, n\}$ and the function $F : \pi \rightarrow \mathrm{R}^+,\ \pi \in \Pi$. We consider the permutation optimization problem (*POP*) consisting in determining optimal permutation $\delta \in \Pi$ such, that

$$F(\delta) = \min\{F(\pi) :\ \pi \in \Pi\}.$$

To solve this problem we propose the evolutionary algorithm which examines local minima of function F. To determine local minimum a local search algorithm is used. We apply the following notation:

π^* : sub-optimal permutation determined by the algorithm,
η : number of elements in the population,
P^i : population in the iteration i of the algorithm, $P^i = \{\pi_1, \pi_2, \ \ldots\ , \pi_\eta\}$,
$LocalOpt(\pi)$: local optimization algorithm to determine local minimum, where π is a starting solution,
LM^i : a set of local minima in iteration i, $LM^i = \{\hat{\pi}_1, \hat{\pi}_2, \ \ldots\ , \hat{\pi}_\eta\}$, $\hat{\pi}_j = LocalOpt(\pi_j)$, $\pi_j \in P^i,\ j = 1, 2, \ \ldots\ , \eta$.
FS^i : a set of fixed elements and position in permutations of population P^i,
$FixSet(LM^i, FS^i)$: a procedure which determines a set of fixed elements and positions in the next iteration of evolutionary algorithm, $FS^{i+1} = = FixSet(LM^i, FS^i)$,
$NewPopul(FS^i)$: a procedure which generates a new population in the next iteration of algorithm, $P^{i+1} = NewPopul(FS^i)$.

In any permutation $\pi \in P^i$ positions and elements which belong to the set FS^i (in iteration i) we call *fixed*, and other elements and positions we call *free*.

Working of the algorithm begins with creating an initial population P^0 (and it can be created randomly). We set a sub-optimal solution π^* as the best element of the population P^0. A new population of iteration $i+1$ (a set P^{i+1}) is generated as follows. For current population P^{i+1} a set of local minima LM^i is determined (for each element $\pi \in P^i$ executing procedure $LocalOpt(\pi)$). Elements which are in the same positions in local minima are established (procedure $FixSet(LM^i, FS^i)$), and a set of fixed elements and positions FS^{i+1} is generated. Each permutation of a new population P^{i+1} has fixed elements (in fixed positions) from the set FS^{i+1}. Free elements are randomly drawn in remaining, free positions of permutation. If permutation $\beta \in LM^i$ exists and $F(\beta) < F(\pi^*)$, then permutation π^* is set to β. The algorithm finishes its work after generating a fixed number of generations.

The general structure of the hybrid evolutionary algorithm for the permutation optimization problem is given below.

Hybrid Evolutionary Algorithm (HEA)

Initialization:
randomly create an initial population
  $P^0 \leftarrow \{\pi_1, \pi_2, \ldots, \pi_\eta\}$;
$\pi^* \leftarrow$ the best element of the population P^0;
the number of iteration $i \leftarrow 0$;
a set of fixed elements and positions $FS^0 \leftarrow \emptyset$;
repeat
  Determine a set of local minima
    $LM^i \leftarrow \{\hat{\pi}_1, \hat{\pi}_2, \ldots, \hat{\pi}_\eta\}$, where
  $\hat{\pi}_j \leftarrow$*LocalOpt*(π_j), $\pi_j \in P^i$;
  for $j \leftarrow 1$ **to** η **do**
    if $F(\hat{\pi}_j) < F(\pi^*)$ **then**
      $\pi^* \leftarrow \hat{\pi}_j$;
    end if;
  end for;
  Determine a set
    $FS^{i+1} \leftarrow FixSet(LM^i, FS^i)$
  and generate a new population
    $P^{i+1} \leftarrow NewPopul(FS^i)$;
  $i \leftarrow i + 1$;
until not Stop Criterion;

The algorithm stops (*Stop condition*) after execution *Max_iter* iterations or after exceeding a fixed time. Procedures *LocalOpt*, *FixSet* and *NewPopul* are described in further parts of the paper.

2.1 Local optimization (*LocalOpt* procedure)

A fast algorithm based on local improvement is applied to determine local minima. The method begins with an initial solution x^0. In every iteration for the current solution x^i the neighborhood $N(x^i)$ is determined. $N(x^i)$ is a subset

of the set of feasible solutions. The neighborhood is generated by moves that are fixed transformations of solution x^i. Next, from the neighborhood the best element x^{i+1} is chosen, which is the current solution in the next iteration.

2.2 A set of fixed elements and position (*FixSet* procedure)

The set FS^i (in iteration i) includes foursomes (a, l, α, φ), where a is an element of the set N ($a \in N$), l is a position in permutation ($1 \leq l \leq n$) and $\alpha,\ \varphi$ are attributes of a pair (a, l). A parameter α means "*adaptation*" and decides on inserting to the set, and φ – "*age*" – decides on deleting from the set. Both of these parameters are described in a further part of this chapter. The maximal number of elements in the set FS^i is n. If a foursome (a, l, α, φ) belongs to the set FS^i, therefore there is an element a in the position l in each permutation from the population P^i.

In every iteration of the algorithm, after determining local minima (*LocalOpt* procedure), a new set $FS^{i+1} = FS^i$ is established. Next, a $FixSet(LM^i, FS^i)$ procedure is called, in which the following operations are executed: (a) changing of the age of each element (φ parameter), (b) deleting the oldest elements, (c) inserting the new elements.

There are two functions of acceptance $\Gamma(i)$ and $\Phi(i)$ connected with insert and delete operations. Both of them can be determined experimentally.

2.2.1 Modification of element's age

In every iteration of the algorithm, the age of each element which belongs to FS^i is increased by 1, that is

$$\forall (a, l, \alpha, \varphi) \in FS^{i+1},\ FS^{i+1} \leftarrow FS^{i+1} \backslash \{(a, l, \alpha, \varphi)\} \cup$$

$$\cup \{(a, l, \alpha, \varphi + 1)\}.$$

The age parameter makes it possible to delete an element from the set FS^i. Each fixed element is free after some number of iterations and can be fixed again in any free position.

2.2.2 Inserting elements

Let $P^i = \{\pi_1, \pi_2, \ldots, \pi_\eta\}$ be a population of η elements in iteration i. For each permutation $\pi_j \in P^i$, applying the local search algorithm (*LocalOpt*(π_j) procedure), a set of local minima $LM^i = \{\hat{\pi}_1, \hat{\pi}_2, \ldots, \hat{\pi}_\eta\}$ is determined. Each permutation

$$\hat{\pi}_j = (\hat{\pi}_j(1), \hat{\pi}_j(2),\ \ldots, \hat{\pi}_j(n)), \quad j = 1, 2,\ \ldots \eta.$$

Let

$$nr(a, l) \geq \left| \{\hat{\pi}_j \in LM^i : \quad \hat{\pi}_j(l) = a\} \right|.$$

It is a number of permutations from the set LM^i, in which element a is in the position l.

If $a \in N$ is a free element and $\alpha = \frac{nr(a,l)}{\eta} \geq \Phi(i)$, then the element a is fixed in the position l; $\varphi = 1$ and the foursome (a, l, α, φ) is inserted to the set of fixed element and positions, that is

$$FS^{i+1} \leftarrow FS^{i+1} \cup \{(a, l, \alpha, \varphi)\}.$$

Acceptance function Φ should be defined so that

$$\forall i, \ \ 0 < \Phi(i) \leq 1.$$

2.2.3 Deleting elements

To test many local minima, each fixed element is released after executing some number of iterations. Let $ES = \{(a, l, \alpha, \varphi) \in FS^{i+1} : \quad \frac{\alpha}{\varphi} \leq \Gamma(i)\}$. If $ES \neq \emptyset$, then elements of this set are deleted from FS^{i+1}, that is $FS^{i+1} \leftarrow FS^{i+1} \backslash ES$, otherwise (when $ES = \emptyset$), let $\Delta = \max\{\frac{\alpha}{\varphi} : \quad (a, l, \alpha, \varphi) \in FS^{i+1}\}$. The element Δ is deleted from the set FS^{i+1}, that is

$$FS^{i+1} \leftarrow FS^{i+1} \backslash \Delta.$$

Function $\Gamma(i)$ should be defined in such a way that each element of the set FS^i is deleted after executing some number of iterations.

2.3 A new population (*NewPopul* procedure)

If a foursome $(a, l, \alpha, \varphi) \in FS^{i+1}$, then in each permutation of a new population P^{i+1} there is an element a in a position l. Randomly drawn free elements will be inserted in remaining (free) positions. Population P^{i+1} is generated as follows.

Algorithm *NewPopul*(FL_i)
$P^{i+1} := \emptyset$;
Determine a set of free elements
$FE := \{a \in N : \ \exists\, (a, l, \alpha, \varphi) \in FS^{i+1}\}$
and a set of free positions
$FP := \{l : \ \exists\, (a, l, \alpha, \varphi) \in FS^{i+1}\}$;
for j:=1 **to** η **do** {*Inserting fixed elements*}
  for every $(a, l, \alpha, \varphi) \in FS^{i+1}$ **do**
    $\pi_j(l) := a$;
  end for;
  $W := FE$;
  {*Inserting free elements*}
  for s:=1 **to** n **do**
    if $s \notin FP$ **then** $\pi_j(s) := w$, **where**
      $w := random(W)$ **and** $W := W \backslash \{w\}$;
    end if;
  $P_{i+1} := P_{i+1} \cup \{\pi_j\}$.
  end for;

Function *random* generates an element of the set W from the uniform distribution. Computational complexity of the algorithm is $O(\eta \cdot n)$.

3 Implementations of the method

In this section a realization of the hybrid evolutionary algorithm for two classical scheduling problems is presented. In a *LocalOpt*(π_j) $\pi_j \in P^i$ procedure there is applied a very quick *descent search* algorithm. The neighborhood is generated by insert moves which consist in taking an element from some position in the permutation and inserting it to another position and moving elements between these positions. Such a neighborhood has $n(n-1)$ elements, where n is the length of permutation. The algorithm starts with a feasible solution $\pi_j \in P^i$, and it tries to improve this solution making small changes in it. Values of functions Γ and Φ were set to $\Gamma(i) = 0.8$ and $\Phi(i) = 0.6$ (after preliminary experiments).

3.1 Single machine scheduling problem

In the single machine total weighted tardiness problem, denoted as $1\|\sum w_i T_i$, a *set of jobs* $N = \{1, 2, \ldots, n\}$ has to be processed without interruption on a single machine that can handle only one job at a time. All jobs become available for processing at time zero. Each job $i \in N$ has an integer *processing time* p_i, a *due date* d_i, and a positive *weight* w_i. For a given sequence of the jobs and (the earliest) completion time C_i, the *tardiness* T_i= max$\{0, C_i - d_i\}$ and the cost $f_i(C_i) = w_i \cdot T_i$ of job $i \in N$ can be computed. The goal is to find a job sequence (permutation) that minimizes the sum of the costs given by $\sum_{i=1}^{n} f_i(C_i) = \sum_{i=1}^{n} w_i \cdot T_i$.

The problem is NP-hard (Lenstra et all [14]). A large number of studies has been devoted to the problem. Emmons [6] derives several dominance rules that restrict the search process for an optimal solution. These rules are used in many algorithms. Enumerative algorithms that use dynamic programming and branch and bound approaches to the problem are described by Fischer [7], Potts and Van Wassenhove [19]. These and other algorithms are discussed and tested in a review paper by Abdul-Razaq et al. [1]. Algorithms are a significant improvement to exhaustive search, but they remain laborious and are applicable only to relatively small problems (with the number of jobs not exceeding 50). The enumerative algorithms require considerable computer resources both in terms of computation times and core storage. Therefore, many algorithms have been proposed to find near optimal schedules in reasonable time. These algorithms can be broadly classified into construction and interchange methods.

The construction methods use dispatching rules to build a solution by fixing a job in a position at each step. Several constructive heuristics are described by Fischer [7] and in a review paper by Potts and Van Wassenhove [20]. They are very fast, but the quality of the solution is not good.

Interchange methods start from an initial solution and repeatedly try to improve the current solution by local changes. The interchanges are continued until a solution that cannot be improved is obtained which is a local minimum. To increase the performance of local search algorithms, there are used metaheuristics like Tabu Search (Crauwels et al. [4]), Simulated Annealing (Potts and Van Wassenhove [20]), Genetic Algorithms (Crauwels et al. [4]), Ant Colony Optimization (Den Basten et al. [5]). A very effective local-search method has been proposed by Congram et al. [3], and next improved by Grosso et al. [10]. The key aspect of the method is its ability to explore an exponential-size neighborhood in polynomial time, using a dynamic programming technique.

An implementation of the hybrid evolutionary algorithm HEA was tested on problems with n=40, 50 and 100 jobs of benchmark instances taken from the OR-library [2]. The benchmark set contains 125 instances for each size of n value. There are results obtained by HEA compared with the best known results from literature in Table 1 and in Figure 1. For comparison, also the results of the constructive algorithm META (composition of SWPT, EDD, AU and COVERT algorithms) from [19] and parallel simulated annealing (SA) algorithm based on [22] are presented. The chosen measure was relative distance (in percent) to the best known solution's goal function.

n	HEA	SA	META
40	0,00%	1,20%	15,92%
50	0,02%	0,86%	14,69%
100	0,26%	1,78%	16,64%
average	**0,09%**	**1,28%**	**15,75%**

Table 1. Results for the single machine total tardiness problem

As we can see in Table 1, the results of the HEA algorithm are much better than results of parallel SA and constructive META algorithms. Replacing simple descent search algorithm in HEA by some advanced local search algorithm additionally improves its results.

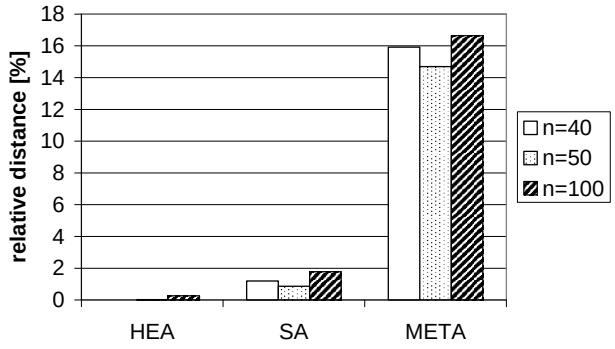

Figure 1. Relative distance to the best known solutions for the single machine scheduling

3.2 Flow shop problem

The classic flow shop problem, denoted as $F||C_{max}$, can be described as follows. There is a set of enumerated jobs J=$\{1,2,\ldots,n\}$ and a set of m machines M=$\{1,2,\ldots,m\}$. A job $j \in J$ is 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 $\pi(0) = 0$, $C_{0k} = 0$, $k = 1, 2, ..., m$, $C_{j0} = 0$, $j = 1, 2, ..., n$. It is well known that $C_{max}(\pi) = C_{\pi(n)m}$.

Johnson [12] gives an O(nlogn) algorithm for two machines ($F|2|C_{\max}$), Garey, Johnson and Seti [8] show that $F|3|C_{\max}$ is strongly NP-hard. The best available branch and bound algorithms are those of Lageweg, Lenstra and Rinnooy Kan [13]. 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 Nowicki, Smutnicki [16] and Grabowski, Wodecki [9]. Sequential simulated annealing algorithms are proposed by Osman, Potts [18], Ogbu, Smith [17], Ishibuchi, Misaki and Tanaka [11]. A parallel simulated annealing algorithm is proposed by Wodecki and Bożejko [22].

Similarly as in a previous problem, the implementation of the hybrid evolutionary algorithm HEA was tested on benchmark instances taken from the OR-library [2], proposed by Taillard [21] and compared with the best known results from the literature. For comparison, also the results of constructing the approximate algorithm NEH [15] and the parallel simulated annealing (SA) algorithm from [22] are presented in Table 2 and in Figure 2.

$n \times m$	HEA	SA	NEH
20×5	0,03%	0,62%	2,87%
20×10	0,51%	1,70%	4,74%
20×20	0,41%	1,82%	3,69%
50×5	0,03%	0,13%	0,89%
50×10	1,17%	0,92%	4,53%
average	**0,43%**	**1,04%**	**3,34%**

Table 2. Results for the flow shop problem

The results presented in Table 2 show, that the HEA algorithm gains considerably better results than other compared algorithms, especially for the large problems.

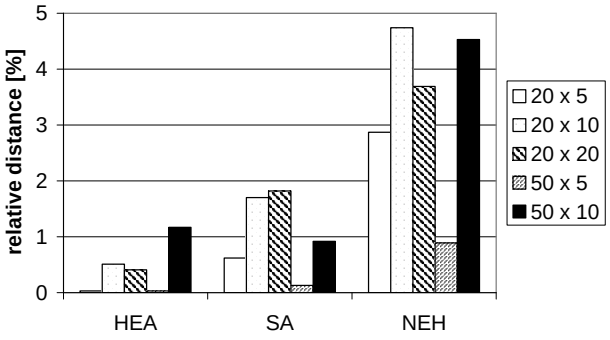

Figure 2. Relative distance to the best known solutions for the flow shop scheduling

4 Conclusions

We have discussed a new approach to the permutation optimization problems based on the hybrid evolutionary algorithm. Using a population with fixed features of local optima makes the performance of the method much better than the iterative improvement approaches, such as in tabu search and simulated annealing methods. The advantage is especially visible for large problems.

References

[1] Abdul-Razaq T.S., C.N. Potts, L.N. Van Wassenhove, A survey of algorithms for the single machine total weighted tardiness scheduling problem, Discrete Applied Mathematics, 26, 1990, 235-253.

[2] Beasley J.E., OR-Library: distributing test problems by electronic mail, Journal of the Operational Research Society 41, 1990, 1069-1072.

[3] Congram, R.K., C.N. Potts, S.L. Van de Velde, An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem, INFORMS Journal on Computing, Vol. 14, No. 1, 2002, 52-67.

[4] Crauwels H.A.J., C.N. Potts, L.N. Van Wassenhove, Local Search Heuristics for the Single machine Total Weighted Tardiness Scheduling Problem, INFORMS Journal on Computing, Vol. 10, No. 3, 1998, 341-350.

[5] Den Basten, M., T. Stützle, M. Dorigo, Design of Iterated Local Search Algorithms An Example Application to the Single Machine Total Weighted Tardiness Problem, J.W. Boers et al. (eds.) Evo Worskshop 2001, LNCS 2037, 441-451.

[6] Emmons H., One machine sequencing to Minimize Certain Functions of Job Tardiness, Opns. Res. 17, 1969, 701-715.

[7] Fisher M.L., A Dual Algorithm for the One Machine Scheduling Problem, Mathematical Programming, 11, 1976, 229-252.

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

[9] Grabowski J., M. Wodecki, A very fast search algorithm for the permutation flow shop problem with makespan criterion, Computers & Operations Research, 31,2004, 1891-1909.

[10] Grosso A., F. Della Croce, R. Tadei, An enhanced dynasearch neighborhood for single-machine total weighted tardiness scheduling problem, Operation Research Letters, 32, 2004, 68-72.

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

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

[13] 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, 53-67.

[14] Lenstra J.K., A.G.H. Rinnoy Kan, P.Brucker, Complexity of Machine Scheduling Problems, Annals of Discrete Mathematics, 1, 1977, 343-362.

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

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

[17] Ogbu F., D. Smith, The Application of the Simulated Annealing Algorithm to the Solution of the $n|m|C_{max}$ Flowshop Problem, Computers and Operations Research, 17(3), 1990, 243-253.

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

[19] Potts C.N., L.N. Van Wassenhove, A Branch and Bound Algorithm for the Total Weighted Tardiness Problem, Operations Research, 33, 1985, 177-181.

[20] Potts C.N., L.N. Van Wassenhove, Single Machine Tardiness Sequencing Heuristics, IIE Transactions, 23, 1991, 346-354.

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

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