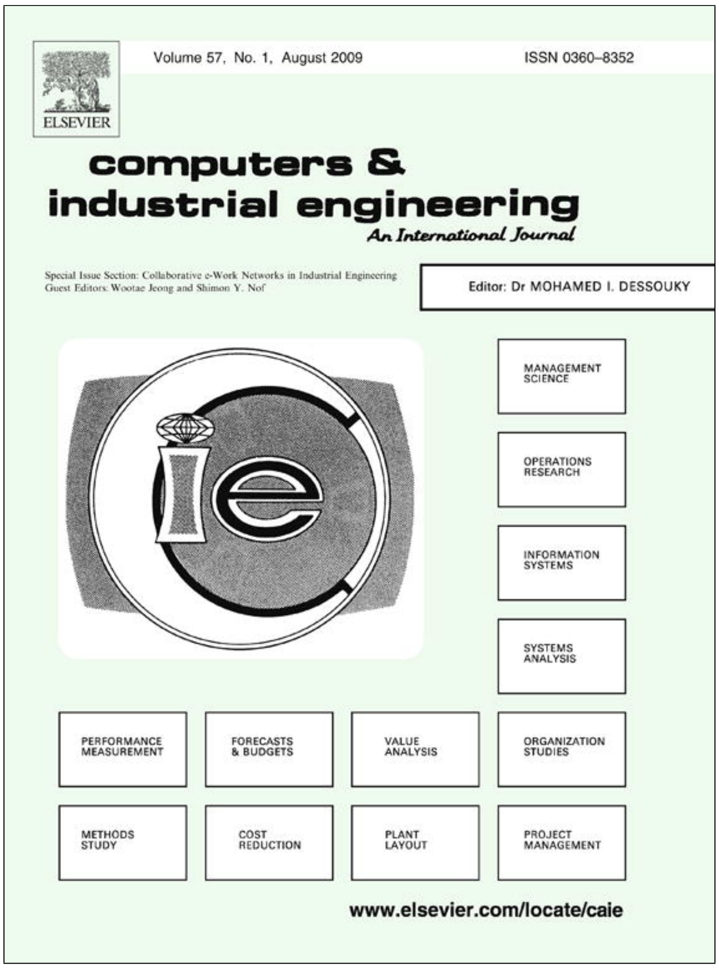
Volume 57, No. 1, August 2009
ISSN 0360–8352
ELSEVIER
computers & industrial engineering
An International Journal
Special Issue Section: Collaborative e-Work Networks in Industrial Engineering
Guest Editors: Wootae Jeong and Shimon Y. Nof
Editor: Dr MOHAMED I. DESSOUKY
MANAGEMENT SCIENCE
OPERATIONS RESEARCH
INFORMATION SYSTEMS
SYSTEMS ANALYSIS
PERFORMANCE MEASUREMENT
FORECASTS & BUDGETS
VALUE ANALYSIS
ORGANIZATION STUDIES
METHODS STUDY
COST REDUCTION
PLANT LAYOUT
PROJECT MANAGEMENT
www.elsevier.com/locate/caie

Computers & Industrial Engineering 57 (2009) 269–276

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

Solving permutational routing problems by population-based metaheuristics

Wojciech Bożejko [a,*], Mieczysław Wodecki [b]

[a] Wrocław University of Technology, Institute of Computer Engineering, Control and Robotics, Janiszewskiego 11-17, 50-372 Wrocław, Poland
[b] University of Wrocław, Institute of Computer Science, Joliot-Curie 15, 50-383 Wrocław, Poland

ARTICLE INFO

Article history:
Received 10 February 2008
Received in revised form 22 November 2008
Accepted 24 November 2008
Available online 6 December 2008

Keywords:
Metaheuristics
TSP
Single machine problem
TSP with due dates

ABSTRACT

In this paper we consider two routing problems: traveling salesman problem (TSP, fundamental problem of the combinatorial optimization) and the TSP with times of traveling, times of processing and due dates where the objective is to minimize the total weighted tardiness (TSPTWT). Since problems are NP-hard in the strong sense, we propose a metaheuristic algorithm to determine a good sub-optimal solution.

We present an algorithm based on the idea of researching and analyzing local optima.

TSP with times of traveling, times of processing and due dates, and with the total weighted tardiness cost function is identical with the single machine total weighted tardiness problem with sequence-dependent setup times. It was possible to find the new best known solutions for 81 of 120 benchmark instances of this scheduling problem using the method proposed here.

© 2008 Elsevier Ltd. All rights reserved.

1. Introduction

Discrete optimization methods are applied to time-dependent systems if there are problems with production management and job scheduling. One can encounter such problems while preparing the travel itineraries for tourists, the optimal ways (e.g., traveling salesman's way), the schedule planning and the expert systems connected with taking optimal decisions. Many of these deal with determining optimal scheduling (permutation of some objects) and usually they are NP-hard. They also have irregular goal functions and very many local minima. Classic heuristic algorithms (tabu search, simulated annealing and genetic algorithm) quickly converge to a local minimum and the diversification of its search process is difficult. We present a general population-based method approach that can be used to find the approximate solutions of the hard combinatorial optimization problems. These problems can be described as follows: for a given finite set of feasible solutions, $\mathcal{X}$, the goal function F is defined as a mapping $F : x \rightarrow \mathrm{R}^{+}$. The optimization problem aims at finding the optimal solution $x^{*} \in \mathcal{X}$ with

$$F(x^{*}) = \min\{F(x) : x \in \mathcal{X}\}.$$

Some representative examples of the permutation problems include the TSP and TSP with times of traveling, times of processing and due dates, and with the total weighted tardiness cost function (TSPTWT). For both these problems feasible solutions can be represented as permutations.

This kind of problems has both theoretical and practical significance. They are NP-hard, so optimal solutions can be obtained only in limited sizes of the problem. So metaheuristic algorithm are commonly applied to obtain near-optimal solutions. These problems are the most difficult interesting from theoretical and important from the practical point of view.

As we said we have adopted and tested PBM for these two NP-hard permutational scheduling problems:

1. TSP,
2. TSP with total weighted tardiness cost function, with processing times and due dates (TSPTWT).

We have used the benchmark tests taken form the TSP-Library (TSPLIB), and from the newest literature for the TSPTWT (Cicirello & Smith, 2005). We have compared the obtained solutions with the optimal ones or the best known. It was possible to obtain very good solutions (with a very small percentage deviation to the best known) by executing a considerably smaller number of iterations and shortening the total calculation time. It was also possible to improve the best solutions for the TSPTWT, which is identified with the single machine total weighted tardiness problem with sequence-dependent setup times.

This paper is organized as follows: in the next section we introduce problems. Next, in the Section 3, the elements of PBM are presented. The Section 4 includes the results of the computational experiments for two strongly NP-hard problems. The final conclusions are presented in the Section 5.

* Corresponding author. Tel.: +48 71 320 2961.
E-mail addresses: wojciech.bozejko@pwr.wroc.pl (W. Bożejko), mwd@ii.uni.wroc.pl (M. Wode).

0360-8352/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cie.2008.11.022

2. The problems

In this section we present three NP-hard optimization problems of which solutions can be represented by a permutation.

2.1. Traveling salesman problem

The classical TSP is defined on an undirected graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$ is a vertex (cites) set and $E = \{\{i, j\} : i \neq j, i, j \in V\}$ is an edge set. A non-negative cost (distance) matrix $C = (c_{i,j})$ is defined on E. The matrix C is symmetric $(c_{i,j} = c_{j,i},\ i, j \in V)$ and satisfies the triangle inequality $(c_{i,j} + c_{j,k} \geqslant c_{i,k},$ for all $i, j, k \in V)$. The problem deals with finding a minimum length Hamiltonian cycle (a tour that passes through each city exactly once, and returns to the starting city) on a G. Each feasible solution of the TSP (a cycle including all the nodes of G) is a permutation of elements of the set V. Let

$$L(\delta) = \sum_{j=1}^{n-1} c_{\delta(j),\delta(j+1)} + c_{\delta(n),\delta(1)},$$

be a length of the traveling salesman's tour

$$\delta = (\delta(1), \delta(2), \ldots, \delta(n-1), \delta(n), \delta(1)), \quad \delta \in \Pi$$

where Π is a set of all permutations of elements of the set V.

The TSP belongs to a class of the NP-hard problems (Sahni & Teogilo, 1976), however a solution of such a problem is usually made using heuristic approach that converges to a locally optimal solution (see Reinelt, 1994). A very popular and effective approach to leaving such local optimum are the following methods: tabu search (Knox, 1994), simulated annealing (Lo & Hus, 1998), neural networks (Leung, Jin, & Xu, 2004) and evolutionary strategies (Tsai, Tsai, & Tseng, 2004). Recent TSP studies using the exact methods (branch and bound approach) are Fischetti and Toth (1997), and Lysgaard (1999). *TSP with processing times and due dates.* Let us assume that $p_i, i \in v$ is the time of processing of the salesman in the city $i \in v$. Also, let the distance between cities be identified with the time of travel. Let call by d_i a *due date* which is the latest time to finish processing in the city $i \in v$. If in some sequence of the cities visiting C_i is the moment of time of finishing processing in the city $i \in v$, than $T_i = \max\{0, C_i - d_i\}$ we call lateness, and $w_i T_i$ is the penalty for lateness, where $w_i \geqslant 0$ is a weight connected with the city $i \in v$. The goal is to minimize the total weighted tardiness.

In the further part of this paper we will consider the problem of routing which can be modelled as a scheduling problem. It is important to point out that it is exactly the same problem, but described in the language of scheduling theory.

2.2. Single machine total weighted tardiness problem with sequence-dependent setup times

The problem can be formulated as follows. Let $N = \{1, 2, \ldots, n\}$ be a set of n jobs which have to be processed, without an interruption, on one machine. This machine can process the most one job in any time. For a job $i (i = 1, 2, \ldots, n)$, let p_i, w_i, d_i be: a *time of executing*, a *weight of the cost function* and a *deadline*. Let s_{ij} be a setup time which represents a time which is needed to prepare the machine for executing a job j after finishing executing a job i. Additionally, s_{0i} is a time which is needed to prepare a machine for executing the first job i (at the beginning of the machine work). If a sequence of job's executing is determined and $C_i (i = 1, 2, \ldots, n)$ is a time of finishing executing a job i, then $T_i = \max\{0, C_i - d_i\}$ we call a *tardiness*, and $f_i(C_i) = w_i T_i$ *a cost of tardiness* of a job i. The considered problem consists in determining such a sequence of jobs'execution of jobs which minimizes a *sum of costs of tardiness*, i.e., $\sum w_i T_i$.

Let Π be a set of permutations of elements from the set N. For a permutation $\pi \in \Pi$ by

$$F(\pi) = \sum_{i=1}^{n} f_{\pi(i)}(C_{\pi(i)})$$

we represent a *cost of permutation* π (i.e., a sum of costs of tardiness when jobs are executed in a sequence determined by a permutation π), where $C_{\pi(i)} = \sum_{j=1}^{i} (s_{\pi(j-1)\pi(j)} + p_{\pi(j)})$ and $\pi(0) = 0$. This problem consists in determining a permutation $\pi \in \Pi$ which has a minimal sum of costs of tardiness.

This scheduling problem is denoted in literature as $1|s_{ij}|\sum w_i T_i$ and it is strongly NP-hard, because $1\|\sum w_i T_i$ (with $s_{ij} = 0$) is strongly NP-hard (see Lenstra, Rinnoy Kan, & Brucker, 1977). To date the best construction heuristics for this problem has been the apparent tardiness cost with setups (ATCS – Lee, Bhaskaran, & Pinedo, 1997). Many metaheuristics have also been proposed. Tan, Narasimban, Rubin, and Ragatz (2000) presented a comparison of four methods of solving the considered problem: branch and bound, genetic search, random-start pair-wise interchange and simulated anneling. Gagné, Price, and Gravel, 2002 compared an ant colony optimization algorithm with other heuristics. Cicirello and Smith (2005) proposed benchmarks for the single machine total tardiness problem with sequence-dependent setups by generated 120 instances and applied stochastic sampling approaches: limited discrepancy search (LDS), heuristic-biased stochastic sampling (HBSS), value biased stochastic sampling (VBSS), value biased stochastic sampling seeded hill-climber (VBSS-HS) and Simulated Annealing. The best goal function value obtained by their approaches was published in literature and presented at http://www.ozone.ri.cmu.edu/benchmarks.html as the upper bounds of the benchmark problems. These upper bounds were next improved by Cicirello (2006) by genetic algorithm, Lin and Ying (2006) by Tabu search, simulated annealing and genetic algorithm, and Liao and Juan (2007) by an ant optimization.

In this paper we propose a method by which we have obtained the new better upper bound values.

3. Population-based metaheuristics

We present a method which belongs to the population-based approaches to solving combinatorial optimization problems (COP), and which consists in determining and researching the local minima. This (heuristic) method is based on the following observation. If there are the same elements in some positions in several solutions, which are local minima, then these elements can be in the same position in the optimal solution.

As we propose this method for solving problems in which a solution is a permutation, that's why in the next part of the paper we identify these two terms.

The basic idea is to start with an initial population (any subset of the solution space). Next, for each element of the population a local optimization algorithm is applied (e.g., descending search algorithm or a metaheuristics, see Potts & VanWassenhove, 1991) 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 from finishing the algorithm's work after executing some number of iterations (when all positions are fixed and there is nothing left to draw), in each iteration"the oldest" fixed elements are set as free. The skeleton of PBM is presented on the Fig. 1.

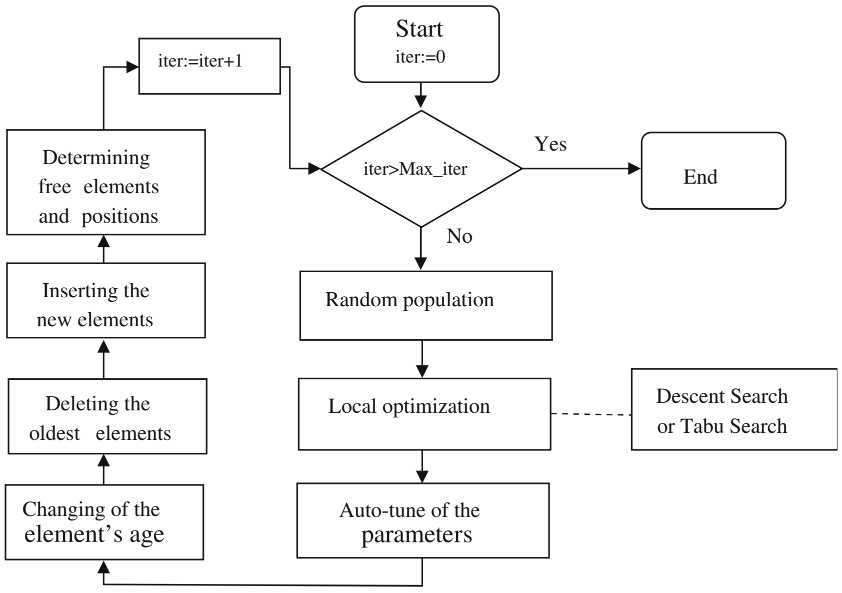
Start
iter:=0
iter>Max_iter
Yes
End
No
Random population
Local optimization
Descent Search or Tabu Search
Auto-tune of the parameters
Changing of the element's age
Deleting the oldest elements
Inserting the new elements
Determining free elements and positions
iter:=iter+1

Fig. 1. Population-based metaheuristics PBM.

Let Π be a set of all permutations of elements from the set $N = \{1, 2, \ldots, n\}$ and the function:

$$F : \Pi \rightarrow R^+ \cup \{0\}.$$

We consider a problem which consists in determining optimal permutation $\hat{\pi} \in \Pi$. We use 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 procedure 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 positions 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 the PBM, $FS^{i+1} = FixSet(LM^i, FS^i)$,
$NewPopul(FS^i)$	:	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*, other elements and positions we call *free*.

The algorithm begins by 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,

$$F(\pi^*) = \min\{F(\beta) : \beta \in P^0\}$$

A new population of iteration $i + 1$ (a set P^{i+1}) is generated as follows: for current population P^i 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 the new population P^{i+1} contains the fixed elements (in fixed positions) from the set FS^{i+1}. Free elements are randomly drawn in the remaining free positions of permutation.

If permutation $\beta \in LM^i$ exists and $F(\beta) < F(\pi^*)$, then we update π^* ($\pi^* \leftarrow \beta$). The algorithm finishes after a fixed number of generations.

The general structure of the PBM for the permutation optimization problem is given below.

Algorithm 1. Population-based metaheuristics (PBM)

```
Initialization:
  $P^0 \leftarrow \{\pi_1, \pi_2, \ldots, \pi_\eta\}$;            random creation of the initial population
  $F(\pi^*) = \min\{F(\beta) : \beta \in P^0\}$   the best element of the population $P^0$
  $i \leftarrow 0$;                                  the number of iteration
  $FS^0 \leftarrow \emptyset$;                              a set of fixed elements and positions
repeat
  Determine a set of local minima
    $LM^i \leftarrow \{\hat{\pi}_1, \hat{\pi}_2, \ldots, \hat{\pi}_\eta\}$,
    where
    $\hat{\pi}_j \leftarrow LocalOpt(\pi_j)$, $\pi_j \in P^i$;
  for $j \leftarrow 1$ to $\eta$ 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 Criterion*) after executing the *Max_iter* iterations or exceeding a fixed time. The complexity of the algorithm depends on time-consuming of the local optimization algorithm. Each other elements of the algorithm (*FixSet*, *NewPopul*, etc.) possesses complexity $O(n)$.

3.1. Local optimization (LocalOpt procedure)

A fast method based on the local improvement is applied to determine the local minima. The method begins with an initial solution π^0. In each iteration for the current solution π^i the neighborhood $\mathcal{N}(\pi^i)$ is determined. Next, from the neighborhood the best element π^{i+1} is chosen constituting the current solution in the next iteration. The method is exhaustive.

Algorithm 2. Descent Search (DS)

```
Select a starting point $\pi^0$;
$\pi_{best} \leftarrow \pi^0$ $i \leftarrow 0$;
repeat
  choose the best element $\beta$ from the neighborhood $\mathcal{N}(\pi^i)$
  according to a given criterion based on the
  goal function's value $F(\beta)$;
  $\pi^i \leftarrow \beta$ $i \leftarrow i+1$;
  if $F(\beta) < F(\pi_{best})$ then
  $\pi_{best} \leftarrow \beta$;
until $F(\beta) \neq F(\pi_{best})$.
```

A crucial ingredient of the local search algorithm is the definition of the neighborhood function in combination with the solution representation. It is obvious that the choice of a good neighborhood is one of the key elements of the neighborhood search method's efficiency.

Traditionally a neighborhood of the solution π is a search space which can be defined as a set of new solutions obtained from π by exactly one move (a single perturbation of π). During the iterative process, the current solution of the algorithm "moves" through the solution space Π from neighbor to neighbor. A move is evaluated by comparing the goal function's value of the current solution to each single one of its neighbor.

The evolution of the solution $\pi^i, i = 1, 2, \ldots, \eta$ draws a trajectory in the search space Π. There exist many criteria for selecting the next solution π^{i+1} in the neighborhood of π^i. If the current solution is not worse that π^i, i.e., $F(\pi^{i+1}) \leqslant F(\pi^i)$, then this strategy is usually called a steepest descent strategy. The main weakness of the descent algorithm is its inability to escape from local minima (all elements in the neighborhood $\mathcal{N}(\pi^i)$ are worse than π^i).

For any iteration of the local search algorithm a subset of moves applicable to it is defined. This subset of moves generates a subset of solutions – the neighborhood. Each move transforms a permutation (current solution) into another permutation from Π.

Let k and l $(k \neq l)$ be a pair of positions in a permutation

$$\pi = (\pi(1), \pi(2), \ldots, \pi(k-1), \pi(k), \pi(k+1), \ldots, \pi(l-1), \pi(l), \pi(l+1), \ldots, \pi(n))$$

Among many types of moves considered in the literature three of them appear prominently:

1. Insert move (*i-move*) consists in removing the job $\pi(k)$ from the position k and next insert it in a position l. Let us assume $k \leqslant l$. Thus, the move generates a new permutation π_l^k in the following way:

$$\pi_l^k = \begin{cases} \pi(i) & \text{if} \quad i < k \text{ or } i > l, \\ \pi(i+1) & \text{if} \quad k < i \leqslant l, \\ \pi(k) & \text{if} \quad i = l. \end{cases}$$

For $k \geqslant l$ the permutation π_l^k can be defined in a similar way as the permutation obtained by moving element $\pi(k)$ to the position before the element $\pi(l)$.

2. Swap move (*s-move*) in which the jobs if $\pi(k)$ and $\pi(l)$ are swapped among some positions k and l. The move generates the following permutation:

$$\pi_l^k = \begin{cases} \pi(i) & \text{if} \quad i \neq k \text{ or } i \neq l, \\ \pi(l) & \text{if} \quad i = k, \\ \pi(k) & \text{if} \quad i = l. \end{cases}$$

3. 2-*opt* in which the jobs from $\pi(k)$ to $\pi(l)$ are removed and inserted in reverse order among some positions k and l. Let us assume $k \leqslant l$. The move generates the following permutation:

$$\pi_l^k = \begin{cases} \pi(i) & \text{if} \quad i < k \text{ or } i > l, \\ \pi(l-i+k) & \text{if} \quad k \leqslant i \leqslant l. \end{cases}$$

Computational complexity of executing *i*-move and 2-*opt* is $O(n)$ and $O(1)$ of executing *s*-move.

In the implementation of the $(LocalOpt\pi_j)$ $\pi_j \in P^i$ procedure a very quick *descent search* algorithm, described above, is applied. The neighborhood is generated by swap moves (*s*-moves) for the QAP and single machine problem and by 2-*opt* moves for the TSP problem.

3.2. A set of fixed elements and position (FixSet procedure)

The set FS^i (in iteration i) includes quadruples (a, l, α, φ), where a is an element of the set $N = \{1, 2, \ldots, n\}$, l is a position in the permutation $(1 \leqslant l \leqslant n)$ and α, φ 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. Parameter φ enables to set free a fixed element after making a number of iterations of the algorithm. However, a parameter α determines such a fraction of local minima in which an element a is in position l.

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 the quadruple (a, l, α, φ) belongs to the set FS^i, then there is an element a in the position l in each permutation from the population P^i.

In each iteration of the algorithm, after determining local minima (*LocalOpt* procedure), a new set $FS^{i+1} = FS^i$ is established. Next,

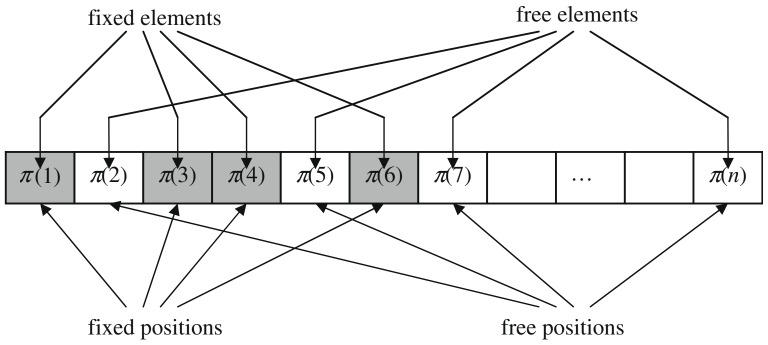

Fig. 2. Fixed and free positions and elements.

a $FixSet(LM^i, FS^i)$ procedure is invoked in which the following operations are executed:

(1) modifying the age of each element,
(2) erasing the oldest elements,
(3) fixing the new elements.

There are two functions of acceptance Φ and Γ connected with the operations of inserting and deleting. Function Φ is determined by an auto-tune function. Function Γ is fixed experimentally as a constant. The schema of the permutation with fixed and free elements is presented on the Fig. 2.

3.3. Fixing elements

Let $P^i = \{\pi_1, \pi_2, \ldots, \pi_\eta\}$ be a population of η elements in the iteration *i*. For each permutation $\pi_j \in P^i$, applying the local search algorithm (($LocalOpt\pi_j$) procedure), a set of local minima $LM^i = \{\hat{\pi}_1, \hat{\pi}_2, \ldots, \hat{\pi}_\eta\}$ is determined. For any permutation $\hat{\pi}_j = (\hat{\pi}_j(1), \hat{\pi}_j(2), \ldots, \hat{\pi}_j(n)), j = 1, 2, \ldots \eta$, let be $nr(a,l) = |\{\hat{\pi}_j \in LM^i : \hat{\pi}_j(l) = a\}|$. It is a number of permutations from the set LM^i in which the element *a* is in the position *l*.

If $a \in N$ is a free element and

$$\alpha = \frac{nr(a,l)}{\eta} \geqslant \Phi(i)$$

then the element *a* is fixed in the position *l*; $\varphi = 1$ and the quadruple (a, l, α, φ) is inserted to the set of fixed elements and positions, that is

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

3.4. Auto-tune of the acceptance level Φ

Acceptance function Φ is defined so that

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

It is possible that no element is acceptable to be fixed in an iteration. To prevent from it, an auto-tune procedure for Φ value is proposed. In each iteration *i*, if

$$\max_{a,l \in \{1,2,\ldots,n\}} \frac{nr(a,l)}{\eta} < \Phi(i)$$

therefore, $\Phi(i)$ value is fixed as

$$\Phi(i) \leftarrow \max_{a,l \in \{1,2,\ldots,n\}} \frac{nr(a,l)}{\eta} - \epsilon,$$

where ϵ is a small constant, e.g., $\epsilon = 0.05$. In this way the value of $\Phi(i)$ is decreased. Similarly, it is possible to increase this value when it is too small (and too many elements are fixed in one iteration) such that at least one element is fixed in each iteration.

3.5. Deleting elements

Each fixed element is released after executing some number of iterations to makes testing a plenty of local minima possible. In this implementation function $\Gamma(i)$ is defined as a constant equals 2, so each element of the set FS^i is deleted after executing two iterations.

3.6. Procedure NewPopul

Let a quadruple $(a, l, \alpha, \varphi) \in FS^{i+1}$. Therefore in each permutation of a new population P^{i+1} there exists 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 3. New Population (***NewPopul***(FS_{i+1}))

```
$P^{i+1} \leftarrow \emptyset$;
Determine a set of free elements
$FE \leftarrow \{a \in N : \not\exists(a, l, \alpha, \varphi) \in FS^{i+1}\}$
and a set of free positions
$FP \leftarrow \{l : \not\exists(a, l, \alpha, \varphi) \in FS^{i+1}\}$;
for $j \leftarrow$ to $\eta$ do
  {Inserting fixed elements}
  for every $(a, l, \alpha, \varphi) \in FS^{i+1}$ do
    $\pi_j(l) \leftarrow a$;
  end for;
  $W \leftarrow FE$;
  {Inserting free elements}
  for $s \leftarrow 1$ to $n$ do
    if $s \in FP$ then
      $\pi_j(s) \leftarrow w$, where
      $w \leftarrow random(W)$ and $W \leftarrow W \setminus \{w\}$;
  end for;
  $P_{i+1} \leftarrow P_{i+1} \cup \{\pi_j\}$.
end for.
```

A function *random* generates an element of the set *W* from the uniform distribution. Computational complexity of the *NewPopul* algorithm is linear.

3.7. Convergence of the method

We can select two levels in the proposed PBM method: (1) non-deterministic, based on random search (random drawing of individuals (permutations) in the population), and (2) deterministic one, based on a local improvement method. The matter of convergence of both of these methods has been considered in the literature.

1. In the paper (Chia & Glynn, 2007) the authors prove the convergence of the random search method showing asymptotic convergence rate for this method in addition.

Table 1
Relative percentage deviations from the optimal or best known solutions and time for the TSP – tests taken from the TSPLIB (1995). Percentage standard deviation σ^{PBM} over 10 runs.

Problem	Meta-RaPS_TSP		PBM_TSP			
	δ	$t^{MetaRaPS}$	δ^{PBM}_{min}	δ^{PBM}_{aver}	t^{PBM}	σ^{PBM}
lin105	0.00	20	0.00	0.05	7	0.334
pr107	0.00	139	0.00	0.26	8	0.289
pr124	0.00	49	0.07	0.31	10	0.303
bier127	0.90	48	0.58	0.58	12	0.383
pr136	0.39	73	0.47	0.78	15	0.332
pr152	0.00	178	0.04	0.14	19	0.128
KroA200	1.07	190	0.78	0.78	32	0.172
KroB200	1.26	134	0.91	1.51	31	0.322
pr226	0.23	357	0.27	0.62	39	0.248
pr264	1.58	824	0.05	0.05	51	0.082
pr299	2.01	766	1.35	2.05	123	0.307
pr439	3.29	2265	1.17	1.17	314	0.142
pr1002	6.04	7032	4.13	5.84	897	0.306
Average	1.29	944.8	0.76	1.09	119.8	0.256

$t^{MetaRaPS}$ – times of computation on the AMD 900 Mhz processor.
t^{PBM}– times of computation on the Celeron 2.4 GHz processor.

2. In the paper (Hanafi, 2000) the author shows the method of creating a convergent local search algorithm (based on the tabu search idea), showing convergence in a finite number of iterations.

The methods which are applied on both levels (even independently) are theoretically convergent. However, specific restrictions requested for theoretical convergence of algorithms (included in works Chia & Glynn, 2007; Hanafi, 2000) makes the method practically unimplementable (i.e., the memory of all visited solutions). It shows the manner of building of theoretically convergent PBM, but such an approach does not address issues of implementation.

4. Computational experiments

PBM was implemented in the Microsoft Visual C++ 6.0 language and executed on the Celeron 2.4 GHz personal computer (for TSP problem instances) and Pentium IV 3.0 GHz personal computer (for TSPTWT problem instances), both with 512 MB RAM memory. Obtained results were compared to the best known solutions and with results of other algorithms from the literature.

Value of functions Γ was set as a constant to $\Gamma(i) = 0.15$. Size of the population was set to 100 individuals. The algorithm stopped after 50 iterations. For each version of the algorithm the following metrics were calculated:

- δ – percentage relative deviation to the benchmark's cost function value where

$$\delta = \frac{F_{alg} - F_{ref}}{F_{ref}} \cdot 100\%$$

where F_{ref} is reference criterion function value (optimal or the best known) taken from the literature and F_{alg} is the result obtained by PBM. There were no situations where $F_{ref} = 0$ for the benchmark tests.

- APRD – average percentage relative deviation.
- σ^{PBM} – standard deviation of the obtained results. For 10 runs of PBM (for each tested instance) we have obtained 10 results δ_i, $i = 1, 2, \ldots, 10$. Then

Table 2
Average improvement rates (%) of the SA, GA and TS approaches from Lin and Ying (2006) compared to the proposed PBM approach for the benchmark instances of Cicirello and Smith (2005) for the single machine total weighted tardiness problem with setup times (TSP with processing times and due dates), $n = 60$. Standard deviation of the PBM results σ^{PBM} over 10 runs.

Problem set	δ^{SA}	δ^{GA}	δ^{TS}	δ^{PBM}	t^{PBM}	σ^{PBM}
1–10	20.00	22.83	19.12	24.38	5.14	5.19
11–20	20.89	27.60	18.46	40.89	5.05	25.92
21–30	30.39	30.93	29.18	34.13	2.97	34.08
31–40	6.86	6.42	5.81	8.19	1.02	3.63
41–50	5.21	5.65	5.33	6.59	6.37	2.57
51–60	5.29	5.65	4.44	9.20	5.99	6.64
61–70	7.25	6.56	7.25	7.77	7.26	4.68
71–80	15.39	15.02	16.32	20.19	6.76	7.42
81–90	0.66	0.56	0.56	0.96	6.51	0.34
91–100	−0.47	−0.50	−0.11	1.11	5.90	1.31
101–110	0.60	0.24	0.64	3.06	6.60	0.53
111–120	−0.23	−0.44	−0.23	1.00	6.15	1.28
Average	9.32	9.97	8.90	13.12	5.48	7.80

$$\sigma^{PBM} = \left(\frac{1}{10} \sum_{i=1}^{10} (\delta_i - APRD)^2 \right)^{\frac{1}{2}}.$$

- t^{Alg} (in seconds) – real time of the algorithm execution.

4.1. TSP

The algorithm was tested on the benchmark instances taken from TSPLIB (1995) with number of cities from 105 to 1002. In the paper DePuy, Morga, and Whitehouse (2005) there are comparative results of algorithm *Meta-RaPS* and other 23 algorithms well known in literature. The fastest of these algorithms is *Meta-RaPS*. We have used the same benchmark instances as in the paper DePuy et al. (2005). Table 1 includes comparisons of the calculations between *PBM_TSP* and *Meta-RaPS* algorithms. *Meta-RaPS* algorithm was executed on the AMD 900 MHz Athlon PC, which has 1397 MFLOPS (WINTUNE98, 1998). Celeron 2.4 GHz, on which our algorithms

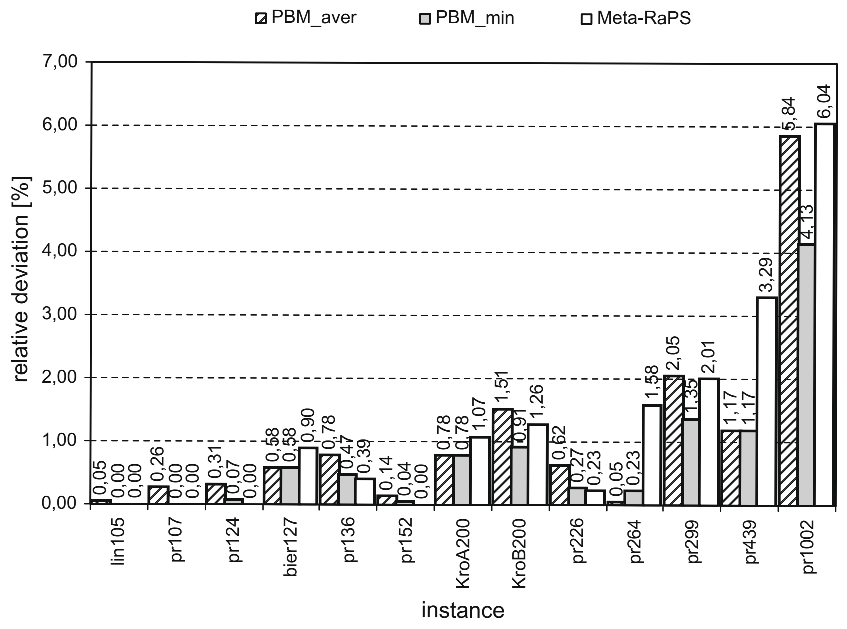

Fig. 3. Relative percentage deviations from the optimal or best known solutions for the TSP (instances taken from the TSPLIB).

were executed, has 2927 MFLOPS (WINTUNE98) so it is 2.1 times faster.

For each algorithm the percentage relative deviation σ to the optimal (or the best known) solution from literature has been calculated. For the TSP these values are presented on TSPLIB web page.

As we can see the *PBM_TSP* is much more faster (on average 3.75 times faster after converting the speed of AMD 900 and Celeron 2.4 processors).

The average percentage deviations to optimal (or the best known solutions) of the *PBM_TSP* is 1.09% and it is 15.5% lower than in *Meta-RaPS* (DePuy, Morga, & Whitehouse, 2005). Comparing execution time of both algorithms we are obtaining in average 119.8 s for *PBM_TSP* and 944.8 s for *Meta-RaPS*. Taking into consideration that a computer with Celeron 2.4 GHz processor is 2.1 times faster than computer with AMD 900 MHz processor we can say that *PBM_TSP* is also 3.75 times faster. Results of comparison are given on the Fig. 3.

Table 3
Results of computational experiments for the problem $1|s_{ij}|\sum w_i T_i$. The new 81 upper bounds are marked by a bold font. The δ is counting based on the benchmark instances values from Cicirello and Smith (2005).

No.	F_{ParPTM}	$\delta^{PBM}(\%)$	t^{PBM}	No.	F_{ParPTM}	$\delta^{PBM}(\%)$	t^{PBM}
1	**618**	−36.81	4.84	61	**76105**	−4.73	7.09
2	**5061**	−22.01	5.05	62	44769	−6.46	7.41
3	**1769**	−24.66	5.30	63	75317	−4.45	7.00
4	**6389**	−23.13	5.28	64	92591	−3.93	7.20
5	4662	−16.84	5.13	65	**126696**	−6.07	6.74
6	7207	−12.58	5.34	66	**59685**	−6.82	7.55
7	**3647**	−16.10	5.23	67	29390	−15.79	7.42
8	143	−56.27	4.97	68	**22120**	−16.22	7.53
9	6482	−14.69	4.89	69	**71118**	−5.70	7.39
10	**1943**	−20.73	5.39	70	75102	−7.51	7.27
11	3898	−25.94	4.67	71	**148230**	−8.06	6.69
12	0	0.00	5.13	72	**45543**	−20.01	6.44
13	**4929**	−19.81	5.20	73	**29045**	−20.35	6.97
14	**2967**	−24.71	5.19	74	**30503**	−20.34	7.00
15	**1321**	−54.68	5.36	75	**21602**	−30.27	6.55
16	**4326**	−35.54	5.34	76	55378	−18.02	6.59
17	**127**	−72.51	4.61	77	33404	−17.64	6.88
18	**1337**	−46.82	5.16	78	**20294**	−19.16	7.06
19	**0**	−100.00	4.88	79	**117596**	−6.54	6.86
20	2983	−28.86	5.00	80	**18620**	−41.53	6.53
21	0	0.00	0.45	81	**384383**	−0.71	6.36
22	0	0.00	0.49	82	**410257**	−0.78	6.58
23	0	0.00	0.27	83	**458844**	−1.55	6.39
24	**1051**	−41.32	5.81	84	**330022**	−0.49	6.66
25	0	0.00	5.14	85	**555065**	−0.63	6.22
26	0	0.00	0.47	86	**362677**	−0.85	6.86
27	0	−100.00	5.47	87	**398551**	−1.11	6.78
28	0	−100.00	5.61	88	**433244**	−0.83	6.39
29	0	0.00	0.39	89	**410739**	−1.48	6.42
30	**0**	−100.00	5.67	90	**402078**	−1.19	6.47
31	0	0.00	0.42	91	**342096**	−1.46	6.33
32	0	0.00	0.50	92	**361921**	−1.05	5.75
33	0	0.00	0.45	93	**407915**	−0.62	6.02
34	0	0.00	0.44	94	**333588**	−0.81	5.59
35	0	0.00	0.44	95	521836	−1.15	5.38
36	0	0.00	0.52	96	462757	−0.35	5.94
37	**436**	−81.89	5.42	97	**413089**	−1.71	5.74
38	0	0.00	0.47	98	**527603**	−0.92	6.27
39	0	0.00	1.09	99	**368353**	−1.72	5.74
40	0	0.00	0.44	100	**436004**	−1.33	6.22
41	**69252**	−5.36	6.59	101	**353018**	−0.79	6.38
42	**58111**	−6.06	6.72	102	**493072**	−0.62	6.41
43	**146510**	−2.32	6.22	103	**378864**	−0.34	6.61
44	**35462**	−8.43	6.14	104	**358033**	−1.10	6.98
45	**59085**	−5.86	6.31	105	**350806**	−23.13	6.78
46	**35080**	−7.66	6.19	106	**454769**	−1.12	6.47
47	**73412**	−4.89	6.36	107	**352766**	−1.09	6.56
48	**65011**	−5.67	6.33	108	**461828**	−1.34	6.84
49	**78005**	−7.29	6.61	109	**413004**	−0.68	6.64
50	**31764**	−12.34	6.23	110	**419437**	−0.44	6.34
51	**50459**	−13.85	6.22	111	**344532**	−1.77	6.31
52	**97052**	−7.89	6.00	112	**372287**	−1.36	6.61
53	**88952**	−6.81	5.99	113	**260093**	−1.18	6.34
54	124229	0.54	5.63	114	**469611**	−0.76	5.77
55	**67969**	−11.00	5.69	115	463002	0.60	5.84
56	**77051**	−12.86	6.56	116	535967	−0.79	6.28
57	**67339**	−4.37	5.89	117	**505416**	−2.54	6.13
58	**47184**	−15.02	6.45	118	354576	−0.84	6.23
59	**53409**	−9.57	5.80	119	**577318**	−1.14	5.94
60	**65111**	−11.21	5.69	120	398723	−0.24	6.05
				Average		−13.12	5.48

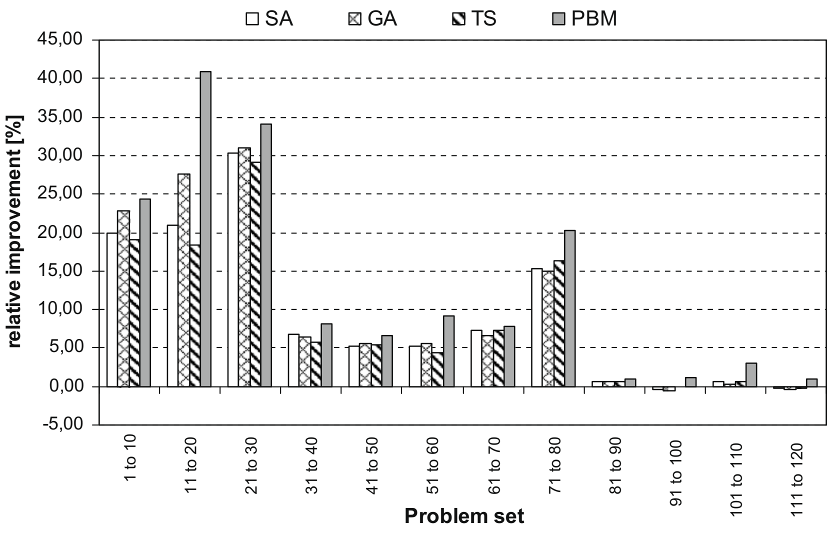

Fig. 4. Relative improvement rates of the optimal or best known solutions for the TSPTWT (instances from Cicirello & Smith, 2005).

4.2. TSP with processing times and due dates

As this problem is identified with the single machine total weighted tardiness problem with sequence-dependent setup times, computational experiments were done to compare the obtained results with the benchmarks from literature (Cicirello & Smith, 2005) and the newest obtained results for this single machine problem (Cicirello, 2006; Lin & Ying, 2006; Liao & Juan, 2007).

The problem instances from the benchmark set of Cicirello are generated according to Lee et al. procedure (Lee et al., 1997). Each problem instance is characterized by three parameters: the due-date tightness factor τ; the due-date range factor R; and the setup time severity factor η. We consider problem sets characterized by the following parameter values: $\tau = \{0.3, 0.6, 0.9\}$, $R = \{0.25, 0.75\}$, $\eta = \{0.25, 0.75\}$. For each of the twelve combinations of parameter values we are obtaining 10 problem instances with 60 jobs each.

As we can see in Table 3 it was possible to find 81 (per 120) new upper bounds of the optimal cost function for the single machine total weighted tardiness problem with sequence-dependent setup times (TSP with processing times and due dates) for the benchmark instances from Cicirello and Smith (2005) and upper bounds from Cicirello (2006), Lin and Ying (2006) and Liao and Juan (2007). In the paper of Lin and Ying (2006), in which there is a comparison of TS, GA and SA approaches, the average running time of SA, GA, and TS was 21 s for each problem on the Pentium IV 1.4 GHz machine (which has 170 MFLOPS in Java Linpack benchmark; Dongarra, 1994).

For the PBM the average percentage deviation to the best known solutions was on the level of $-13,12\%$ and the average time of execution of one instance took 5.48 s on Pentium IV 3.0 GHz machine (262 MFLOPS in Java Linpack benchmark Dongarra, 1994, so it is 1.54 times faster than Pentium IV 1.4 GHz). So we can say PBM algorithm is in average 48% times faster than algorithms of Lin and Ying (2006), obtaining better results. Results of comparison are given in the Table 2 and on the Fig. 4.

Two new optimal solution has been found. For the benchmark's instance number 19 and 32 the optimal value of the cost function is 0 (optimality is proved because of non-negativeness of the cost function).

5. Conclusions

We have discussed a new approach to the permutation optimization problems based on the population-based metaheuristics. The usage of the 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, simulated annealing as well as classical genetic algorithms. Especially good results are obtained for the problems in which significance is a position of an element in a solution (i.e., TSP with due dates). The advantage is especially visible for large problems.

References

Chia, Y. L., Glynn, P. W. (2007). Optimal convergence rate for random search. In *Proceedings of the 2007 INFORMS simulation society research workshop*, <http://www.informs-sim.org/2007informs-csworkshop/11.pdf>.

Cicirello, V. A. (2006). Non-wrapping order crossover: An order preserving crossover operator that respect absolute position. In *Proceedings of the 8th annual genetic and evolutionary computation conference GECCO 2006*. ACM Press.

Cicirello, V. A., & Smith, S. F. (2005). Enhancing stochastic search performance by value-based randomization of heuristics. *Journal of Heuristics, 11*, 5–34.

DePuy, G. W., Morga, R. J., & Whitehouse, G. E. (2005). Meta-RaPS: A simple and effective approach for solving the traveling salesman problem. *Transportation Research Part E, 41*, 115–130.

Dongarra, J. J. (1994). Performance of various computers using standard linear equations software, technical report CS-89-85, University of Tennessee.

Fischetti, M., & Toth, P. (1997). A polyhedral approach to the asymmetric traveling salesman problem. *Management Science, 43*(11), 1520–1536.

Gagné, C., Price, W. L., & Gravel, M. (2002). Comparing an ACO algorithm with other heuristics for the single machine scheduling problem with sequence-dependent setup times. *Journal of the Operational Research Society, 53*, 895–906.

Hanafi, S. (2000). On the convergence of tabu search. *Journal of Heuristics, 7*, 47–58.

Knox, J. (1994). Tabu search performance on the symmetric traveling salesman problem. *Computers & Operations Research*, 867–876.

Lee, Y. H., Bhaskaran, K., & Pinedo, M. (1997). A heuristic to minimize the total weighted tardiness with sequence-dependent setups. *IIE Transactions, 29*, 45–52.

Lenstra, J. K., Rinnoy Kan, A. G. H., & Brucker, P. (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics, 1*, 343–362.

Leung, K.-S., Jin, H.-D., & Xu, Z.-B. (2004). An expanding self-organizing neural network for the traveling salesman problem. *Neurocomputing, 62*, 267–292.

Liao, C.-J., & Juan, H. C. (2007). An ant opimization for single-machine tardiness shceduling with sequence-dependent setups. *Computers & Operations Research, 34*, 1899–1909.

Lin, S., & Ying, K.-C. (2006). Solving single-machine total weighted tardiness problems with sequence-dependent setup times by meta-heuristics. *International Journal of Advanced Manufacturing Technology*. doi:10.1007/s00170-006-0693-1.

Lo, C. C., & Hus, C. C. (1998). Annealing framework with learning memory. *IEEE Transaction on System, Man, Cybernetics, Part A, 28*(5), 1–13.

Lysgaard, J. (1999). Cluster based branching for the asymmetric traveling salesman problem. *European Journal of Operational Research, 119*(2), 314325.

Potts, C. N., & Van Wassenhove, L. N. (1991). Single machine tardiness sequencing heuristics. *IIE Transactions, 23*, 346–354.

Reinelt, G. (1994). *The traveling salesman: Computational solutions for TSP applications*. Berlin: Springer.

Sahni, S., & Teogilo, G. (1976). P-complee approximation problems. *Journal of the Association for Computing Machinery, 23*(3), 555–565.