International Journal of Computational Intelligence Research.
ISSN 0973-1873 Vol.2, No.2 (2006), pp. 151–158

Population-Based Heuristics for Hard Permutational Optimization Problems

Wojciech Bożejko[1] **and Mieczysław Wodecki**[2]

[1]Wrocław University of Technology
Institute of Computer Engineering, Control and Robotics
Janiszewskiego 11-17, 50-372 Wrocław, Poland
wojciech.bozejko@pwr.wroc.pl

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

Abstract: **In this paper we present a population-based algorithm for solving permutational optimization problems. It consists in testing the feasible solutions which are the 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 one can suppose that these elements can be in the same positions in the optimal solution. The presented properties and ideas can be applied to two classical strongly NP-hard scheduling problems:**

1. **single machine total weighted tardiness problem**
2. **flow shop problem with goal function C_{max}.**

Computational experiments on the benchmark instances from the OR-Library [3] are presented and compared with the results yielded by the best algorithms discussed in the literature. These results show that the algorithm proposed allows us to obtain the best known results for the benchmarks in a short time.

Keywords: algorithm, metaheuristics, scheduling, optimization, population, local search

I. Introduction

Discrete optimization methods are applied to time-dependent systems if there are problems of production management and job's scheduling. One can encounter such problems with 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 are the Traveling Salesman Problem [24], the Quadratic Assignment Problem [12] and the single [1] and multi-machine [10] scheduling problems. Although these problems present simple formulations, they are very troublesome, because in most cases they belong to the NP-hard problems class. For many of these problems a natural solution representation constitutes a permutation. Because of their inherent nature, the permutation optimization problem (*POP*) has a huge number of various local optima. Therefore, to solve these problems the approximate methods mainly used are:

1. constructive methods,
2. improvement methods.

Constructive heuristics are essentially single pass methods which construct a permutation by fixing at each step the position of an element in the permutation. They are very fast, easily implementable, however the performance of the generated solutions is rather poor. The second group of the methods deals with improving a given solution. An important member of this group is the local search method. These algorithms usually finish calculations after finding a few local

optima. Thus nowadays many approaches, not so "sensitive" to detecting local optima – especially artificial intelligence methods, are applied to solve *POP*.
In this paper we present a method which belongs to the improvement approaches for solving *POP*, and which consists in determining and researching the local minima. This (heuristics) 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 each element of the population, a local optimization algorithm is applied (e.g. descending search algorithm or metaheuristics, see [22]) 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 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 proposed here method is especially helpful in solving large-sized instances of very difficult problems with irregular goal functions. One can encounter such problems, among others, in very efficient control strategies of the discrete production Just-In-Time systems which are often elements of production's recommending systems.
We have adopted and tested our approach for two NP-hard permutational scheduling problems:

1. single machine total weighted tardiness problem

2. flow shop problem with goal function C_{max}.

We use the benchmark tests taken form the OR-Library [3]. We compare the obtained solutions with the best known ones published in the literature. We obtain the same solutions by executing a considerably smaller number of iterations and shortening the total calculation time.
This paper is organized as follows: in the next section we introduce the notation, the elements of population-based algorithm and the local search algorithms. The Section 3 includes the results of the computational experiments for two classical strong NP-hard problems of scheduling. The obtained results are compared with the best known in the literature. The final conclusions are presented in Section 4.

II. Population-based algorithm

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

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

We consider the problem which consists in determining optimal permutation $\hat{\pi} \in \Pi$.
To solve this problem we propose the heuristic algorithm which examines local minima of the function F. To determine local minimum a local improvement algorithm is used. 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 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.

We also use the following functions connected with population's evolution:

$FixSet(LM^i, FS^i)$	:	a procedure which determines a set of fixed elements and positions in the next iteration of the 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*, 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+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 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 population-based heuristic algorithm for the permutation optimization problem is given below.

Algorithm 1. Testing of Feasible Local Minima (TFLM)

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*(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 Criterion*) after executing the *Max_iter* iterations or exceeding a fixed time.
Procedures *LocalOpt*, *FixSet* and *NewPopul* are described in further parts of the paper.

A. 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. The $\mathcal{N}(\pi^i)$ is a subset of the set of feasible solutions. The neighborhood is generated by moves that are fixed transformations of the solution π^i into another permutation from the set of the feasible solution Π. Next, from the neighborhood the best element π^{i+1} is chosen which is the current solution in the next iteration.

Algorithm 2. Neighborhood Search (NS)

Select a starting point x;
$x_{best} \leftarrow x$;
repeat
chose an element y from the neighborhood $\mathcal{N}(x)$
according to a given criterion based on the
goal function's value $F(y)$;
$x \leftarrow y$;
if $F(y) < F(x_{best})$ **then**
$x_{best} \leftarrow y$;
until some termination condition is satisfied.

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 that can be reached 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 π^i, $i = 1, 2, ...$, 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. $F(\pi^{i+1}) \leq F(\pi^i)$, 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, two 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. Thus the move generates a new permutation π_l^k in the following way:

$$\pi_l^k = (\pi(1), \ldots, \pi(k-1), \pi(k+1), \ldots, \pi(l-1),$$
$$\pi(l), \underline{\pi(k)}, \pi(l+1), \ldots, \pi(n)),$$

if $k < l$, and

$$\pi_l^k = (\pi(1), \ldots, \pi(l-1), \underline{\pi(k)}, \pi(l), \pi(l+1), \ldots,$$

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

if $k > l$.

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:

$$\pi_l^k = (\pi(1), \pi(2), \ldots, \pi(k-1), \underline{\pi(l)},$$
$$\pi(k+1), \ldots, \pi(l-1), \underline{\pi(k)}, \pi(l+1), \ldots, \pi(n)).$$

Computational complexity of executing $i-$move is $O(n)$ and $O(1)$ of executing $s-$move.
In an implementation of the *LocalOpt*(π_j) $\pi_j \in P^i$ procedure there is applied a very quick *improvement search* algorithm. The neighborhood is generated by swap and insert moves (both *i*-moves and *s*-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 $(n-1)(3n/2-2)$ elements, where n is the length of permutation.

B. 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 ($1 \leq l \leq n$) 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 determine 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, a $FixSet(LM^i, FS^i)$ procedure is invoked 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 the operations of inserting and deleting. Both of them can be determined experimentally.

1) Modification of element's age

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

$$\forall (a, l, \alpha, \varphi) \in FS^i,$$

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

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) Inserting 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*(π_j) procedure), a set of local minima $LM^i = \{\hat{\pi}_1, \hat{\pi}_2, \ldots, \hat{\pi}_\eta\}$ is determined. For any permutation

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

let be

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

It is a number of permutations from the set LM^i in which the 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 quadruple (a, l, α, φ) is inserted to the set of fixed elements and positions, that is

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

Acceptance function Φ should be defined so that

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

3) Deleting elements

To test many local minima each fixed element is released after executing some number of iterations.
Let be

$$ES = \{(a, l, \alpha, \varphi) \in FS^{i+1} : \ \ \frac{\alpha}{\varphi} \leq \Gamma(i)\}.$$

It is a set of some elements and positions which are fixed in all permutations of the population P^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

$$\delta = (a', l', \alpha', \varphi') \in FS^{i+1}$$

be such that

$$\frac{\alpha'}{\varphi'} = \min\{\frac{\alpha}{\varphi} : \ \ (a, l, \alpha, \varphi) \in FS^{i+1}\}.$$

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.

C. A new population procedure

If a quadruple $(a, l, \alpha, \varphi) \in FS^{i+1}$, then 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
$FE \leftarrow \{a \in N : \ \neg \exists\, (a, l, \alpha, \varphi) \in FS^{i+1}\}$
and a set of free positions
$FP \leftarrow \{l : \ \neg \exists\, (a, l, \alpha, \varphi) \in FS^{i+1}\}$;
for $j \leftarrow$ **to** η **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** $W \leftarrow W \backslash \{w\}$;
  end for;
  $P_{i+1} \leftarrow P_{i+1} \cup \{\pi_j\}$.
end for.

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

III. Implementations of the method

In this section an application of the TFLM algorithm for two classical scheduling problems:

- single machine total weighted tardiness problem,
- flow shop problem,

is presented. We have compared the results of TFLM algorithm (for the test instances from OR-Library [3]) with other algorithms from the literature.
The algorithm was coded in C++ and implemented on a Sun Enterprise 4x400MHz computer. 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 $\Gamma(i) = 0.15$ and $\Phi(i) = 0.6$ (after preliminary experiments).
Let $\mathcal{AH}$ be an heuristic algorithm used to solve the considered problem. For each group of test instances we have collected the following values:

$F^{\mathcal{AH}}$ – the cost found by the $\mathcal{AH}$ algorithm,
$\delta^{\mathcal{AH}}$ – percentage relative deviation of the cost function $F^{\mathcal{AH}}$ found by algorithm $\mathcal{AH}$ from the optimal (or best known) solution value OPT,

$$\delta^{\mathcal{AH}} = \frac{F^{\mathcal{AH}} - OPT}{OPT} * 100\%,$$

$\delta^{\mathcal{AH}}_{APRD}$ – the average (for group of test instances) percentage relative deviation ($\delta^{\mathcal{AH}}$) of the cost function $F^{\mathcal{AH}}$ found by algorithm $\mathcal{AH}$,
$t^{\mathcal{AH}}$ – the time of execution of the $\mathcal{AH}$ algorithm (in seconds).

A. 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 an 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

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

where $\pi \in \Pi$, and Π is a set of all permutation
The problem is NP-hard (Lenstra et all [16]). A large number of studies has been devoted to the problem. Emmons [7] proposes 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 [8], Potts and Van Wassenhove [21]. These and other algorithms are discussed and tested in a review paper by Abdul-Razaq et al. [1]. Algorithms consitute a significant improvement to the 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 the computation times and the core storage. Therefore, many algorithms have been proposed to find near optimal schedules in a reasonable time. These algorithms can be broadly classified into construction and improvement methods.
The construction methods use dispatching rules to come up with a solution by fixing a job in a position at each step. Several constructive heuristics are described by Fischer [8] and in a review paper by Potts and Van Wassenhove [22]. They are very fast, but their quality is not good.
The improvement 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. Such a solution is a local minimum. To increase the performance of local search algorithms, there are used metaheuristics like Tabu Search (Crauwels et al. [5]), Simulated Annealing (Potts and Van Wassenhove [22]), Genetic Algorithms (Crauwels et al. [5]), Ant Colony Optimization (Den Basten et al. [6]). A very effective local-search method was proposed by Congram et al. [4] and next improved by Grosso et al. [11]. The key aspect of the method is its ability to explore an exponential-size neighborhood in polynomial time, using a dynamic programming technique.

1) Experimental results

An implementation of the TFLM algorithm was tested on problems with n=40, 50 and 100 jobs of benchmark instances taken from the OR-library [3]. The benchmark set contains 125 instances for each size of the n value. There are results obtained by TFLM which are compared with the best known results from the literature in the Table 1 and in the Figure 1. For comparison, also the results of the constructive algorithm META (composition of SWPT, EDD, AU and COVERT algorithms, the best constructive ones, see [2]) and parallel simulated annealing (SA) algorithm based on [25] are presented. The selected measure constituted relative distance (in percent) to the best known solution's goal function.

Table 1: Results for the single machine total tardiness problem (the average percentage relative deviation $\delta^{\mathcal{AH}}_{APRD}$ and average times of execution $t^{\mathcal{AH}}$).

n	TFLM		SA		META
	$\delta^{\mathcal{TFLM}}_{APRD}$	$t^{\mathcal{TFLM}}$	$\delta^{\mathcal{SA}}_{APRD}$	$t^{\mathcal{SA}}$	$\delta^{\mathcal{METAH}}_{APRD}$
40	0,03%	0.17	1.20%	0.89	15.92%
50	0.06%	0.44	0.86%	1.68	14.69%
100	0.26%	1.31	1.78%	2.90	16.64%
average	**0.09%**	**0.64**	**1.28%**	**1.82**	**15.75%**

As we can see in the Table 1 and in the Figure 1, the average results of the TFLM algorithm are much better than the results of the parallel SA and the constructive META algorithms. We can additionally improves TFLM results by the replacement of the simple descent search algorithm by an advanced local search algorithm.

B. 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,...,n} and a set of m machines M={1,2,...,m}. A job $j \in J$ is a sequence of m operations O_{j1}, O_{j2},...,O_{jm}. Operation O_{jk} corresponds to the processing of job j on a machine k during an uninterrupted processing time p_{jk}. We

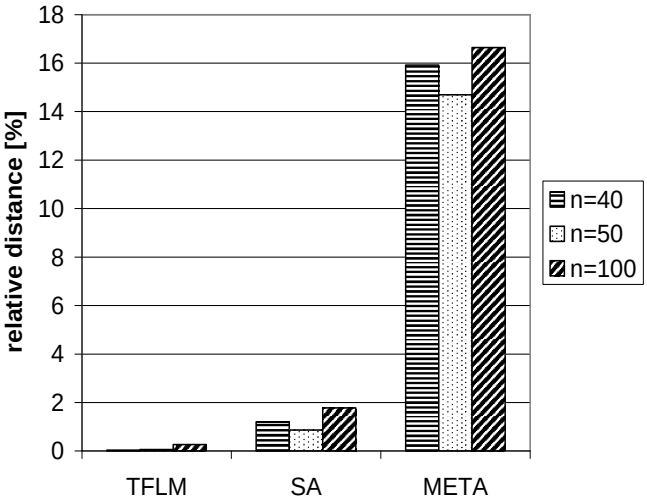

Figure. 1: Relative distance to the best known solutions for the single machine scheduling.

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 want 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 a 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

$$\pi(0) = 0,\ C_{0k} = 0,\ k = 1, 2, \ldots, m,$$

and

$$C_{j0} = 0,\ j = 1, 2, \ldots, n.$$

It is well known that $C_{max}(\pi) = C_{\pi(n)m}$ (see [10]).

Johnson [14] provides with an $O(nlogn)$ algorithm for two machines ($F|2|C_{\max}$), Garey, Johnson and Seti [9] 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 [15]. Their performance is not entirely satisfactory, 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 [18] and Grabowski, Wodecki [10]. Sequential simulated annealing algorithms are proposed by Osman, Potts [20], Ogbu, Smith [19], Ishibuchi, Misaki and Tanaka [13]. A parallel simulated annealing algorithm is proposed by Wodecki and Bożejko [25].

1) Experimental results

Similarly as in the previous problem, the implementation of the TFLM algorithm was tested on benchmark instances taken from the OR-library [3] proposed by Taillard [23] and compared with the best known results from the literature. To make a comparison, also the results of the best constructive approximate algorithm NEH [17] and the parallel simulated annealing (SA) algorithm from [25] are presented in the Table 2 and in the Figure 2.

Table 2: Results for the flow shop problem (the average percentage relative deviation $\delta_{APRD}^{\mathcal{AH}}$ and average times of execution $t^{\mathcal{AH}}$).

$n \times m$	TFLM		SA		NEH
	$\delta_{APRD}^{\mathcal{TFLM}}$	$t^{\mathcal{TFLM}}$	$\delta_{APRD}^{\mathcal{SA}}$	$t^{\mathcal{SA}}$	$\delta_{APRD}^{\mathcal{NEH}}$
20×5	0,04%	0.22	1.39%	1.10	2,87%
20×10	0.89%	0.44	2,32%	1.50	4,74%
20×20	0.76%	1.19	2.06%	2.75	3,69%
50×5	0,12%	4.57	0,15%	11.20	0,89%
50×10	1.26%	17.02	1.68%	19.10	4,53%
average	**0,61%**	**4.69**	**1,62%**	**7.13**	**3,34%**

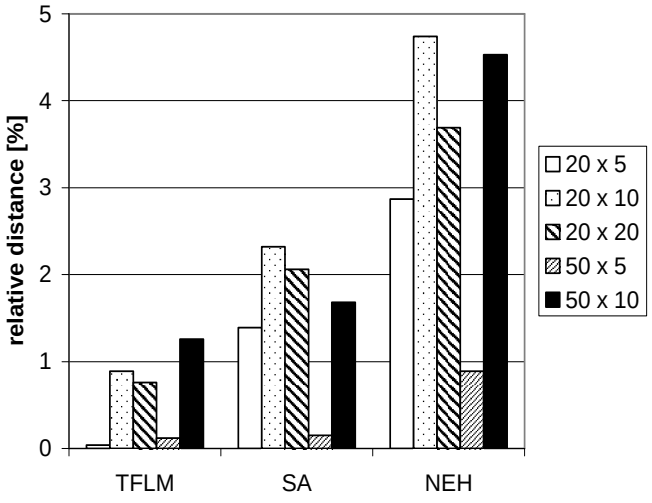

Figure. 2: Relative distance to the best known solutions for the flow shop scheduling

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

IV. Conclusions

We have discussed a new approach to the permutation optimization problems based on the population-based heuristic algorithm. 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 and simulated annealing methods. The advantage is especially visible for large problems.
In the future work we want to research an influence the parameters of the algorithm (Φ and Γ) to values of obtaining solutions, especially variable values modified similarly like temperature parameter in the simulated annealing method.

References

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

[2] S.M. Akturm, B.M. Yildirim, A new dominance rule for the total weighted tardiness problem, Production Planning & Control, Vol. 10, No. 2, pp. 138-149, 1999.

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

[4] R.K. Congram, 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, pp. 52-67, 2002.

[5] H.A.J. Crauwels, 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, pp. 341-350, 1998.

[6] M. Den Basten, 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, LNCS 2037, pp. 441-451, 2001.

[7] H. Emmons, One machine sequencing to Minimize Certain Functions of Job Tardiness, Operations Research, 17, pp. 701-715, 1969.

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

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

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

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

[12] M. Hasegawa, T. Ikeguchi, K. Aihara, K. Itoh, A novel chaotic search for quadratic assignment problems, European Journal of Operational Research, 139, pp. 543-556, 2002, .

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

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

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

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

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

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

[19] F. Ogbu, 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), pp. 243-253, 1990.

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

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

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

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

[24] Ch.F. Tsai, C.W. Tsai, Ch.Ch. Tseng, A new hybrid heuristic approach for solving large traveling salesman problem, Information Sciences, 166, pp. 67-81, 2004.

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