Parallel Population Training Algorithm for the Single Machine Total Tardiness Problem

Wojciech Bożejko[1] and Mieczysław Wodecki[2]

[1] Institute of Engineering Cybernetics, Wrocław University of Technology
Janiszewskiego 11-17, 50-372 Wrocław, Poland
email: wbo@ict.pwr.wroc.pl
[2] Institute of Computer Science, University of Wrocław
Przesmyckiego 20, 51-151 Wrocław, Poland
email: mwd@ii.uni.wroc.pl

Abstract. In this paper we consider a single machine scheduling problem which is a representative of the wide group of combinatorial optimization problems. They are usually NP-hard and they have irregular goal functions with very many local minima. We present two parallel population training (PT) algorithms, based on a cooperative and independent model of the search process, for solving such a class of problems. It consists in generating and testing the population of feasible solutions, which are local minima. By computer simulations on benchmarks taken from the OR-Library we have obtained very promising results using a cluster of personal computers and the MPI library. ⋆

1 Introduction

We present a general parallel population training approach that can be used to find approximate solutions of the hard optimization problems (OP). 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 of determining optimal permutation (with minimal value of the goal function F) in the solution space Π. Some representative examples of permutation problems are: the Traveling Salesman Problem, the Quadratic 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 us to obtain good results. These algorithms usually finish calculations after finding a few local optima. So, nowadays many approaches, not so "sensitive" to local optima – especially artificial intelligence methods, are applied to solve OP.

We propose a method of the parallel algorithm's construction for solving OP consisting in determining and researching of local minima. This method

⋆ The work was supported by KBN Poland, within the grant No. 4T11A01624

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. So we consider two models here: independent – with no communication and independent subpopulations, and cooperative model where all subpopulations have the same fixed elements in positions in each permutations. 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.

2 Population Training Method

To solve the *OP* problem we propose the population training method which examines local minima of function F. To determine local minimum a local search algorithm is used. We apply the following notation: π^* - suboptimal permutation algorithm determined, η - number of population elements, P^i - population in the iteration i, $P^i = \{\pi_1, \pi_2, \ldots, \pi_\eta\}$, $LocalOpt(\pi)$ - local optimization algorithm determining the local minimum where π is a starting solution, LM^i - set of local minima in iteration i, $LM^i = \{\hat{\pi}_1, \hat{\pi}_2, \ldots, \hat{\pi}_\eta\}$, where $\hat{\pi}_j = LocalOpt(\pi_j)$, $\pi_j \in P^i$, $j = 1, 2, \ldots, \eta$, FS^i - set of fixed elements and position in permutations of population P^i, $FixSet(LM^i, FS^i)$ - procedure which determines a set of fixed elements and positions in the next iteration of evolutionary algorithm, $FS^{i+1} \leftarrow FixSet(LM^i, FS^i)$, $NewPopul(FS^i)$ - procedure which generates a new population in the next iteration of the algorithm, $P^{i+1} \leftarrow NewPopul(FS^i)$.

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 the permutation. If permutation $\beta \in LM^i$ exists and $F(\beta) < F(\pi^*)$, therefore permutation π^* is set to β. The algorithm finishes its work after generating a fixed number of generations.
The general structure of the PT sequential algorithm for the permutation optimization problem is present in the paper of Bożejko and Wodecki [1].

3 Parallel Population Training Algorithm

For the parallel version of the PT algorithm, two models of parallelization have been proposed.

Independent model. In this model processes are executing independent algorithms (working on different subpopulations) with different parameters of fixing elements in positions. At the end, the best solution of each subpopulation is collected and the best solution of the whole algorithm is chosen. The advantage of this model is diversification of the solutions space search process.
Cooperative model. This model works in the same way as a sequential algorithm with a big population composed of subpopulations of each process. Processes are connected with independent algorithms and different subpopulations. The same parameters of fixing elements in positions are used in every process. In every iteration the average number $nr(a, l)$ of permutations (for all subpopulations) in which there is an element a in a position l is computed.
The general structure of the parallel PT algorithm can be described as follows.

Parallel Population Training Algorithm (ParPTA)
Initialization:
$P_p^0 \leftarrow \{\pi_1, \pi_2, \ldots, \pi_\eta\}$; $\pi_p^* \leftarrow$ the best element of the population P_p^0; $i \leftarrow 0$;
$FS_p^0 \leftarrow \emptyset$; p is the number of process $p = 1, 2, ..., ntasks$;
parfor $p = 1..nrtasks$
  For each process p determine sets of local minima
  $LM_p^i \leftarrow \{\hat{\pi}_1, \hat{\pi}_2, \ldots, \hat{\pi}_\eta\}$, where $\hat{\pi}_j \leftarrow LocalOpt(\pi_j), \pi_j \in P^i$;
  for $j \leftarrow 1$ **to** η **do**
    if $F(\hat{\pi}_j) < F(\pi_p^*)$ **then** $\pi_p^* \leftarrow \hat{\pi}_j$ **end if**;
  end for;
  Determine sets $FS_p^{i+1} \leftarrow FixSet(LM_p^i, FS_p^i)$;
  Compute $nr_p(a, l)$ – numbers of permutation from the set LM_p^i, in which there is an element a in the position l;
  if *model* = *cooperative* **then**
    Exchange values of nr_p between processes, compute the average values and store them in every process instead of original nr_p values;
  end if;
  Generate new populations $P_p^{i+1} \leftarrow NewPopul(FS_p^i)$;
  $i \leftarrow i + 1$;
until not Stop Criterion;

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

A fast algorithm based on local improvement is applied to determine local minima.
The fixed elements and positions

A set FS^i includes foursomes (a, l, α, φ), where $a \in N$, l is a position in permutation ($1 \leq l \leq n$) and α, φ are attributes of a pair (a, l). The parameter α means "*adaptation*" and decides about inserting to the set, and φ – "*age*" – decides about deleting from the set. Maximal number of elements in the set FS^i is n. If a foursome (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 every iteration of the algorithm, after determining the local minima (procedure *LocalOpt*), a new set $FS^{i+1} = FS^i$ is established. Next, a $FixSet(LM^i, FS^i)$ procedure is called, in which there the following operations are executed: (a) changing of the age of each element (φ parameter), (b) inserting the new elements, (c) deleting the oldest elements.

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

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

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

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.

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 teh 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 $\hat{\pi}_j = (\hat{\pi}_j(1), \hat{\pi}_j(2), \ldots, \hat{\pi}_j(n))$, $j = 1, 2, \ldots \eta$. Let $nr(a,l) = \left|\{\hat{\pi}_j \in LM^i : \hat{\pi}_j(l) = a\}\right|$. It is a number of permutations from the set LM^i, in which there is an element a 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 $\forall i,\ 0 < \Phi(i) \leq 1$.

Deleting elements. To test many local minima, each fixed element is released after executing some number of iterations.

Let $ES = \{(a,l,\alpha,\varphi) \in FS^{i+1} : \frac{\alpha}{\varphi} \leq \Gamma(i)\}$. 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 $\Delta = \max\{\frac{\alpha}{\varphi} : \ (a,l,\alpha,\varphi) \in FS^{i+1}\}$. An 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 so that each element of the set FS^i is deleted after executing some number of iterations.

A new population

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. Therefore to generate a new population P^{i+1}, randomly drawn free elements are inserted in remaining free positions of the elements of population P^i.

4 Computational experiments

In this section, a realization of the parallel population training algorithm for the classic scheduling problem is presented.

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. 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 completion time C_i, the *tardiness* $T_i = \max\{0, C_i - d_i\}$. The goal is to find a job sequence (permutation) that minimizes the sum of the costs given by $\sum_{i=1}^{n} w_i \cdot T_i$. The problem is NP-hard. Many sequential metaheuristics have been proposed for this problem. A very effective local search method has been proposed by Congram et al. [2], and next improved by Grosso et al. [3]. The key aspect of the method is its ability to explore an exponential size neighborhood in polynomial time, using a dynamic programming technique.

Parallel version of the PT Algorithm was implemented in C++ language with the MPI library (MPICH) and it was tested on the cluster of 15 computers with Intel Celeron 2.4GHz processors and 240MB RAM memory, connected by Ethernet 100mbit/s.

In a $LocalOpt(\pi_j)$, $\pi_j \in P^i$ procedure there is applied a very fast *descent search* algorithm. The neighborhood is generated by insert moves which consists in taking an element from some position in the permutation and inserting it to another position and moving elements between these positions. The algorithm starts with a feasible solution $\pi_j \in P^i$, and it tries to improve this solution making small changes to it. Values of functions Γ and Φ were set to $\Gamma(i) = 0.8$ and $\Phi(i) = 0.6$. Size of the population was set to 100 individuals per processor. The algorithm stopped after 10 descents (where descent means finding fully fixed solution, with all elements fixed in some positions).

Table 1. Relative deviation to the best known solutions for the algorithm ParPTA

n	SeqPTA	ParPTA			META
	1 proc	4 proc	8 proc	15 proc	
Independent Model					
40	0,003%	0,001%	0,000%	0,000%	15,920%
50	0,015%	0,004%	0,004%	0,004%	14,690%
100	0,358%	0,041%	0,030%	0,030%	16,640%
average	0,125%	0,015%	0,011%	0,011%	15,750%
Cooperative Model					
40	0,003%	0,000%	0,000%	0,000%	15,920%
50	0,015%	0,007%	0,002%	0,000%	14,690%
100	0,358%	0,093%	0,086%	0,074%	16,640%
average	0,125%	0,033%	0,029%	0,025%	15,750%

An implementation of a PT algorithm (PTA) was tested on problems with n=40, 50 and 100 jobs of benchmark instances taken from the OR-library: http://mscmga.ms.ic.ac.uk/info.html. The benchmark set contains 125 instances for each size of n value. There are results obtained by ParPTA compared with the best known results from the literature in Table 1,2,3. For comparison, also the results of constructive algorithm META (composition of SWPT, EDD, AU and COVERT algorithms) from [4] are presented. The chosen measure was relative distance (in percent) to the best known solution's goal function[1].

As it turned out, the strategy of independent searches (independent ParPAT) was significantly better than others. The average distance of the independent ParPTA for 15 processors to the reference solutions were over 10 times smaller

[1] optimal solutions for n=40, 50 and the best known solutions for n=100

then the results of SeqPTA (results of cooperative ParPTA were in average 5 times smaller) and over 1000 times smaller then the constructive META algorithm. Independent ParPTA has also the highest number of optima found (Tab. 3.). Execution times of the parallel algorithms (cooperative and independent) were comparable (a few second per one instance of the problem), but independent algorithm was a little faster (cause of rare communication).

Table 2. Execution times (in seconds) for all 125 instances

n	SeqPTA	ParPTA					
	1 proc	4 proc		8 proc		15 proc	
		Ind	Coo	Ind	Coo	Ind	Coo
40	11	9	5	7	7	9	10
50	44	43	42	35	37	36	34
100	3025	3425	3350	3500	3500	3400	3900
average	1027	1159	1132	1181	1181	1148	1314

Table 3. Number of optima[1] found per 125 instances

n	SeqPTA	ParPTA					
	1 proc	4 proc		8 proc		15 proc	
		Ind	Coo	Ind	Coo	Ind	Coo
40	121	124	125	125	125	125	125
50	111	122	120	124	121	124	123
100	71	85	80	88	87	90	88
average	101	110	108	112	111	113	112

5 Conclusions

We have discussed a new approach to the permutation optimization problems based on the PT 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. As compared to the sequential algorithm, parallelization very significantly increases the quality of solutions obtained. Two models of the algorithm's cooperativity have been proposed. In most cases the results of tests (after a small number of iterations) are equal optimal or the best known.

References

1. Bożejko W., Wodecki M., A Hybrid Evolutionary Algorithm for some Discrete Optimization Problems, ISDA'05, IEEE Computer Society, 2005, 326-331.
2. 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.
3. 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.
4. Potts C.N., Van Wassenhove L.N., A Branch and Bound Algorithm for the Total Weighted Tardiness Problem, Operations Research, 33 (1985), 177-181.