4 Conclusions

We have proposed a new parallel approaches based on metaheuristics: tabu search, simulated annealing, genetic algorithm and scatter search, designed for solving permutation scheduling problems. Multiple-thread search parallelization increased the quality of obtained solutions keeping comparable costs of computations. Superlinear speedup is observed in cooperative model of parallelism. Computer experiments show, that the parallel algorithm is considerably more efficient and stable in comparison to sequential algorithms. Especially, parallel metaheuristics are the modern and efficient tool to solve strongly NP-hard scheduling problems, thus can be recommended for the use in the scheduling practice and theory.

References


Parallel Population Training Metaheuristics for the Routing Problem

Wojciech Bożejko¹ and Mieczysław Wodecki²

1 Institute of Computer Engineering, Control and Robotics, Wroclaw University of Technology
2 Janiszewskiego 11-17, 50-372 Wroclaw, Poland
email: wojciech.bozejko@pwr.wroc.pl
email: mwd@i.uni.wroc.pl

Abstract. In the paper we propose a parallel population training metaheuristics for solving TSP with times of traveling, times of processing and due dates, which can be formulated as a single machine scheduling problem with total weighted tardiness criterion and sequence-dependent setup times. Since the problems is NP-hard in the strong sense, we propose a metaheuristic algorithm to determine a good suboptimal solution. Calculations on the representative group of benchmark instances were done and results were compared with the best known from literature. Obtained solutions were better than benchmark ones for almost all instances.

1 Introduction

The problem which is considered here can be formulated both as a routing problem and single machine scheduling problem. Let us assume, that \( p_i \in \mathcal{V} \) is the time of processing of the salesman in the city \( i \in \mathcal{V} \). Also let the distance between cities is identified with the time of travel. Let call by \( d_i \) a due date, the latest time to finishing traveling in the city \( i \in \mathcal{V} \). If in some sequence of the cities visiting \( C_t \) is the moment of time of finishing processing in the city \( i \in \mathcal{V} \), than \( T_t = max\{C_t - d_i\} \) we call lateness, and \( w_t \) is the penalty for lateness, where \( w_t \geq 0 \) is a weight connected with the city \( i \in \mathcal{V} \). The goal is to minimize the total weighted tardiness.

In the further part of the paper we will consider this problem of routing which can be modelled as a scheduling problem. It is denoted in literature as [4] \( \sum_{i} w_i T_i \) and \( x \) is strongly NP-hard. To date the best construction heuristics for this problem has been the Apparent Tardiness Cost with Setups (ATCS – Lee, Bhaskaran and Pinedo [5]). Many metaheuristics have also been proposed. Tan et al. [8] presented a comparison of four methods of solving the considered problem: Branch and Bound, Genetic Search, random-start pair-wise interchange and Simulated Annealing. Gagné, Price and Gravel [4] compared the Ant Colony Optimization algorithm with other heuristics. Civirelli and Smith [2] 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 [3] by Genetic Algorithm, Lin and Ying [7] by Tabu Search, Simulated Annealing and Genetic Algorithm, and Liao and Juan [6] by the Ant Optimization.

In this paper we propose a method by which we have obtained the new better upper bound values. It is based on the idea which we introduced in the paper [1].

2 Definition of the Problem

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 at most one job in any time. For a job \( i \) \((i = 1, 2, \ldots, n)\), let \( p_i, u_i, d_i \) be: a time of executing, a weight of the cost function and a deadline. Let \( s_{ij} \) be a setup time representing 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 \( T = \max \{0, C_i - d_i\} \) we call it a tardiness, and \( f_i(C_i) = u_i T_i \) a cost of tardiness of a job \( i \). The considered problem consists in determining such a sequence of executing of jobs which minimizes a sum of costs of tardiness, i.e. \( \sum C_i T_i \).

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

\[
F(\pi) = \sum_{i=1}^{n} f_{i+1}(C_{\pi(i)})
\]

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

3 Population Training Metaheuristics

We present a method belonging to the population training approaches 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.

Because 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) 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, than 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.

Let \( \Pi \) 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: \( \pi^* \) - sub-optimal permutation determined by the algorithm, \( K \) - number of elements in the population, \( P^i \) - population in the iteration \( i \) of the algorithm, \( \pi_i = [\pi_{i1}, \pi_{i2}, \ldots, \pi_{iK}] \), LocalOpt(\( \pi_i \)) - local optimization procedure to determine local minimum, where \( \pi \) is a starting solution, \( LM^i \) - a set of local minima in iteration \( i \), \( FS^i \) - a set of fixed elements and position in permutations of population \( P^i \). NewPopul(\( FS^i \)) - a procedure which determines a new population in the next iteration of the population training heuristics, \( FS^{i+1} \) - a set of fixed elements and position in permutations of population \( P^{i+1} \). 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 \( \pi^* \) (\( \pi^* \leftarrow \beta \)). The algorithm finishes (Stop Criterion) after executing the MaxIter iterations.

4 Parallel Population Training Metaheuristics (ParPTM)

For the parallel version of the PTM two models of parallelization have been proposed.

Single-thread model. This model executes multiple population training heuristics which synchronize populations in each iteration, i.e. common global table of the fixed
elements and positions is used for each processor. In every iteration the average number count(a, l) of permutations (for all subpopulations) in which there is an element a in a position l is computed. This model is called cooperative.

Multiple-thread model. In this model processes execute 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. We will call this model independent.

The general structure of the parallel population training metaheuristic using MPI library is given below.

Algorithm 1. Parallel Population Training Metaheuristics

```
procedure ParPTM(int n, int benchm_opt, bool stops, bool communication)
    n - number of jobs to schedule;
    benchm_opt - value of the benchmark's near optimal solution, taken from [2];
    stops - if it is true, the algorithm stops after achieving benchm_opt;
    communication - if it is true the algorithm has got a common count table;
parfor p ← 1..n_tasks do
    best_cost_p ← ∞;
    a_p ← 0.7;
    fixed_p ← 0;
    int count_p[N.MAX][N.MAX];
    int ϕ_p[N.MAX]; for i ← 1..N.MAX do ϕ_p[i] ← 0; end for;
    perm P_p[K.MAX];
for iter ← 1..R do
    for t ← 0..K - 1 do
        P_p[t] ← Random_Perms();
        int f ← Descent_Search(P_p[t]);
        if f < best_cost_p then best_cost_p ← f; end if;
        int f_min:
        if stops == true then
            MPL_Allreduce(&f, &f_min, 1, MPL_INT, MPL_MIN, MPL_COMM_WORLD);
        end if;
if f_min <= benchm_opt then return f_min;
end for;
end for;
for i ← 1..K - 1 do
    for j ← 1..n do
        count_p[i][j] ← 0;
    end for;
end for;
for t ← 1..K - 1 do
    for i ← fixed_p do
        count_p[i][P_p[t][i]] += 1;
    end for;
end for;
for i ← 1..K - 1 do
    if communication == true then
        int new_count[N.MAX][N.MAX];
        MPL_Allreduce(count_p, new_count, (n + 1) * (n + 1),
                        MPL_INT, MPL_SUM, MPL_COMM_WORLD);
    end if;
end for;
for i, j ← 1..n do
    count_p[i][j] ← new_count[i][j]/n_tasks;
end for;
end for;
end parfor;
```

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 one iteration for the current solution π the neighborhood N(π) is determined. Next, from the neighborhood the best element π+1, which is the current solution in the next iteration, is chosen.

In the implementation of the LocalOpt procedure a very quick descent search algorithm is applied. The neighborhood is generated by insert moves.

A set of fixed elements and position (FixSet procedure). The set FS (in iteration i) includes quadruples (a, l, α, ϕ), where a is an element of the set N = {1, 2, ..., n}, l is a position in the permutation (1 ≤ l ≤ n) and α, ϕ are attributes of a pair (a, l). A
parameter $\alpha$ means "adaptation" and decides on inserting to the set, and $\varphi$ - "age" - decides on deleting from the set. Parameter $\varphi$ enables to set free a fixed element after making a number of iterations of the algorithm. However, a parameter $\alpha$ 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 $FSI$ is $n$. If the quadruple $(a, l, \alpha, \varphi)$ belongs to the set $FSI$ then there is an element $a$ in the position $l$ in each permutation from the population $P$.

In each iteration of the algorithm, after determining local minima (LocalOpt procedure), a new set $FSI^{t+1} = FS^I$ is established. Next, 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 $\Phi$ and $I$ connected with the operations of inserting and deleting. Function $\Phi$ is determined by an auto-tune function. Function $I$ is fixed experimentally as a constant.

Fixing elements. Let $P^I = \{\pi_1, \pi_2, \ldots, \pi_K\}$ be a population of $K$ elements in the iteration $i$. For each permutation $\pi_j \in P^I$, applying the local search algorithm (LocalOpt procedure), a set of local minima $LM^I_j = \{\pi_{i_1}, \pi_{i_2}, \ldots, \pi_{i_F}\}$ is determined. For any permutation $\pi_j = \{\pi_j(1), \pi_j(2), \ldots, \pi_j(n)\}, j = 1, 2, \ldots, K$, let be $count(a, l) = \{\pi_j \in LM^I: \pi_j(l) = a\}$. 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 (i.e. $a \in LM^I$) and  
\[ \alpha = \frac{\text{count}(a, l)}{K} \geq \Phi(i), \]
then the element $a$ is fixed in the position $l$; we assume $\varphi = 1$ and the quadruple $(a, l, \alpha, \varphi)$ is inserted to the set of fixed elements and positions, that is  
\[ FSI^{t+1} \leftarrow FSI^{t+1} \cup \{(a, l, \alpha, \varphi)\}. \]

Auto-tune of the acceptance level $\Phi$. Function $\Phi$ is defined so that for each iteration $0 < \Phi(i) \leq 1$. It is possible that no element is acceptable to be fixed in an iteration. To prevent from this an auto-tune procedure for $\Phi$ value is proposed. In each iteration $i$, if  
\[ \max_{a, l \in (1, 2, \ldots, n)} \frac{\text{count}(a, l)}{K} < \Phi(i) \]
therefore, $\Phi(i)$ value is fixed as  
\[ \Phi(i) \leftarrow \max_{a, l \in (1, 2, \ldots, n)} \frac{\text{count}(a, l)}{K} - \epsilon, \]
where $\epsilon$ 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).

Deleting elements. Each fixed element is released after executing some number of iterations to make possible testing of a plenty of local minima. In this implementation function $I(t)$ is defined as a constant equals 2, so each element of the set $FSI$ is deleted after executing $2t$ iterations.

Procedure NewPopul. Let a quadruple $(a, l, \alpha, \varphi) \in FSI^{t+1}$. Therefore, in each permutation of a new population $P^{t+1}$ there exists an element $a$ in a position $l$. Randomly drawn free elements will be inserted in remaining (free) positions.

5 Computational Experiments

Parallel population training metaheuristics was implemented in C++ language with the MPI library and it was tested on the Silicon Graphics SGI Altix 3700 Bx2 with 128 Intel Itanium2 1.5 GHz processors and cache-coherent Non-Uniform Memory Access (cc-NUMA), craylinks NUMAflex4 in fat tree topology with the bandwidth 4.3 Gbps, installed in the Wrocław Center of Networking and Supercomputing. Up to 16 processors of the supercomputer were used. Computational experiments were done to check the speed of convergence of the parallel algorithm in two proposed models of communication and to compare the obtained results with the benchmarks from literature [2] and the newest obtained results for this single machine problem [3], [7], [6].

In the Table 1 results of computational experiments for the scheduling problem 1 | $p_j$ | $C_j$, \(W, T, R\) are presented with the new upper bounds marked. As we can see in Table 1 it was possible to find 81 new upper bounds of the optimal cost function for the 120 benchmark instances. The average percentage deviation to the solutions of Cicirello and Smith [2] was on the level of $-13.12\%$ and was better than earlier proposed approaches for this problem (Cicirello and Smith [2] and upper bounds from Cicirello [3]).

Two criteria of the algorithm termination were checked. First one stops the algorithm after achieving the benchmark value from [2] or exceeding $R = 10$ iterations. This criterion was helpful to determine the speedup of the parallel algorithm tested for two models: the independent model and the model with communication. The results of computations for this criterion of the algorithm termination are presented in Table 3. The second criterion of the algorithm termination determines the speed of the parallel algorithm convergence. Algorithms execute exactly $R = 10$ iterations. The results of computations for this criterion are presented in Table 4.

For each version of the ParPTM algorithm (cooperative or independent, stops after achieving benchmark's value or stops after executing a constant number of iterations) the following metrics were calculated:

- PRD - Percentage Relative Deviation to the benchmark cost function value from [2],
- APRD - Average Percentage Relative Deviation, for groups of instances,
- $t_{cpu}$ (in seconds) - real time of executing the algorithm for 120 benchmark instances from [2],
- $t_{cpu}$ (in seconds) - the sum of time's consuming on all processors.
Table 1. Results of computational experiments for the problem \( \sum x_i T_i \). The new \( R_1 \) upper bounds are marked by a bold font.

<table>
<thead>
<tr>
<th>Nr</th>
<th>( P_{\text{ParPTM}} ) (%)</th>
<th>( P_{\text{Prd}} ) (%)</th>
<th>( P_{\text{ParPTM}} ) (%)</th>
<th>( P_{\text{ParPTM}} ) (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>474.67</td>
<td>474.67</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>2</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>3</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>4</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>5</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>6</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>7</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>8</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>9</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>10</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>11</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>12</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>13</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>14</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>15</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>16</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>17</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>18</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>19</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>20</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>21</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>22</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>23</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>24</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>25</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>26</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>27</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>28</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>29</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>30</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>31</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>32</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>33</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>34</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>35</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>36</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>37</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>38</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>39</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>40</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>41</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>42</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>43</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>44</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>45</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>46</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>47</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>48</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>49</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>50</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>51</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>52</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>53</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>54</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>55</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>56</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>57</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>58</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>59</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>60</td>
<td>500.00</td>
<td>500.00</td>
<td>50.00</td>
<td>50.00</td>
</tr>
</tbody>
</table>

As we can see in Table 3 the cooperative model of the ParPTM has shorter real times of execution (\( t_{\text{total}} \)) than the independent model. Also the times consumed by all the processors (\( t_{\text{run}} \)) is shorter for the cooperative model of communication. It means that the cooperative model obtains faster the same solutions as the independent model does.

Table 4 presents ARPD values obtained after execution of \( R = 10 \) iterations. The average value is better for the cooperative model of communication.

## 6 Conclusion

We have discussed a new approach to the permutation optimization problems based on the parallel population training metaheuristic 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 taboo search, simulated annealing as well as classical genetic algorithms.

### References

On Transformation of Planning with Uncertainty to Linear Programming

Adam Galuszka
Institute of Automatic Control, Silesian University of Technology
44-100 Gliwice, Poland
Adam.Galuszka@polsl.pl

Abstract. Planning is a difficult computational problem. One way to increase efficiency of searching for a solution may be a transformation of a problem to another problem and then search for a solution of the transformed problem. In this work it is shown how to transform planning problem with uncertainty to linear programming. The transformation from planning to Linear Programming is based on mapping of conditions and operators in each plan step to variables. Truth-values of conditions are mapped to 0 and 1 values and the objective function reaches the maximum if the goal situation is true in last step of planning. It is proposed to represent the utility function (representing degree of desirability of the plan) using the objective function of linear programming problem. Exemplary simulation shows the efficiency of proposed approach.

1 Introduction

This paper deals with planning in the presence of uncertainty. Planning problem is a problem of finding a set of actions (often also called operators) which transform an initial state into a desired goal situation. It should be distinguished from scheduling - well-known and frequently used technique of improving the cost of a plan. Planning is understood as causal relations between actions, while scheduling is concerned with metric constraints on actions [17]. When all states of a planning problem (including an initial and a goal) are described by a given set of predicates, then the problem is called STRIPS planning problem [15]. There are many applications of the planning problems in industrial processes, production planning, logistics and robotics [1, 15]. The STRIPS system has been successfully applied in planning modules of Deep Space One spacecraft [17] and for elevators control in Rockefeller Center in New York [13]. In these applications it is shown that planning algorithm in practice has to be efficient in time and generate high quality plans (close to optimal).

1.1 Uncertainty

The world we live in is full of uncertainties: for example the tomorrow’s weather or the world’s peace. This is part of people’s life and while we grow up we learn, to certain extent, to perceive the uncertain phenomena. We also feel how to use it. As example such sentence can be cited: “the weather is warm today”, what can have different meanings whether we are in Africa or in Alaska or is it Summer or Winter in Poland. Such