

4 Conclusions

We have proposed a new parallel approaches based on metaheuristics: tabu search, simulated annealing, genetic algorithm and scatter search, designed for solving permutational 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 the 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

1. Aarts E.H.L., de Bont F.M.J., Habers J.H.A., van Laarhoven P.J.M., Parallel implementations of the statistical cooling algorithm, *Integration* 4 (1986), 209–238.
2. Bożejko W., Pempera J., Smutnicki A., Parallel single-thread strategies in scheduling. Proceedings of ICAISC 2008, Lecture Notes in Artificial Intelligence, Springer 2008 (accepted).
3. Bożejko W., Wodecki M., Parallel tabu search method approach for a very difficult permutation scheduling problems, Proceedings of PARELEC 2004, IEEE Computer Society P2080 (2004), 156–161.
4. Bożejko W., Parallel job scheduling algorithms (in Polish). Report PRE 29/2003, Ph.D. dissertation, Wrocław University of Technology (2003).
5. Cantu-Paz E., Efficient and Accurate Parallel Genetic Algorithms, Springer (2000).
6. Crainic T.G., Toulouse M., Parallel metaheuristics, in Fleet management and logistics (T.G. Crainic and G. Laporte, eds.), 205–251, Kluwer (1998).
7. Grabowski J., Pempera J., New block properties for the permutation flow shop problem with application in tabu search. *Journal of Operational Research Society* 52 (2000), 210–220.
8. Grama A, Kumar V., State of the Art in Parallel Search Techniques for Discrete Optimization Problems, *IEEE Transactions on Knowledge and Data Engineering* 11, (1999), 28–35
9. James T., Rego C., Glover F., Sequential and Parallel Path-Relinking Algorithms for the Quadratic Assignment Problem, *IEEE Intelligent Systems* Vol. 20 Issue 4 (2005), 58–65.
10. Miki M., Hiroyasu T., Kasai M., Application of the temperature parallel simulated annealing to continuous optimization problems, *IPSL Transaction* 41 (2000), 1607–1616.
11. Nawaz M., Enscore Jr. E.E. and Ham I. A heuristic algorithm for the m -machine, n -job flow-shop sequencing problem. *OMEGA International Journal of Management Science* 11 (1983), 91–95.
12. Nowicki E., Smutnicki C., A fast tabu search algorithm for the permutation flow shop problem. *European Journal of Operational Research* 91 (1996), 160–175.
13. Nowicki E., Smutnicki C., Some aspects of scatter search in the flow-shop problem, *European Journal of Operational Research* 169 (2006), 654–666.
14. Reeves C. R. and Yamada T., Genetic algorithms, path relinking and the flowshop sequencing problem. *Evolutionary Computation* 6 (1998), 45–60.
15. Smutnicki C., Scheduling algorithms (in Polish), EXIT, Warsaw (2002).
16. Taillard E.: Benchmarks for basic scheduling problems. *European Journal of Operational Research* 64 (1993) 278–285
17. Taillard E., Home page, <http://www.eivd.ch/ina/collaborateurs/etd/default.htm>
18. Voss S., Tabu search: Applications and prospects, in *Network Optimization Problems* (Du D.Z., Pardalos P.M., eds.), World Scientific (1993).

Parallel Population Training Metaheuristics for the Routing Problem

Wojciech Bożejko¹ and Mieczysław Wodecki²

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

² Institute of Computer Science, University of Wrocław
Joliot-Curie 15, 50-383 Wrocław, Poland
email: mwd@ii.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 , $i \in \mathcal{V}$ is the time of processing of the salesman in the city $i \in \mathcal{V}$. Also let the distance between cities is identify with the time of travel. Let call by d_i a *due date*, the latest time to finishing processing in the city $i \in \mathcal{V}$. If in some sequence of the cities visiting C_i is the moment of time of finishing processing in the city $i \in \mathcal{V}$, than $T_i = \max\{0, C_i - d_i\}$ we call lateness, and $w_i T_i$ is the penalty for lateness, where $w_i \geq 0$ 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 $1|s_{ij}|\sum w_i T_i$ and it 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. Cicirello 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 $N = \{1, 2, \dots, n\}$ be a set of n jobs which have to be processed, without an interruption, on one machine. This machine can process at the most one job in any time. For a job i ($i = 1, 2, \dots, n$), 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 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 ($i = 1, 2, \dots, n$) 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 executing 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 $\pi(0) = 0$ 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 Π be a set of all permutations of elements from the set $N = \{1, 2, \dots, n\}$ 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, K – number of elements in the population, P^i – population in the iteration i of the algorithm, $P^i = \{\pi_1, \pi_2, \dots, \pi_K\}$, $LocalOpt(\pi)$ – local optimization procedure to determine local minimum, where π is a starting solution, LM^i – a set of local minima in iteration i , $LM^i = \{\hat{\pi}_1, \hat{\pi}_2, \dots, \hat{\pi}_j\}$, $\hat{\pi}_j = LocalOpt(\pi_j)$, $\pi_j \in P^i$, $j = 1, 2, \dots, K$, FS^i – a set of fixed elements and position in permutations of population P^i , $FixSet(LM^i, FS^i)$ – a procedure which determines a set of fixed elements and positions in the next iteration of the population training metaheuristics, $FS^{i+1} = FixSet(LM^i, FS^i)$, $NewPopul(FS^i)$ – 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 a 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 (*Stop Criterion*) after executing the *Max.iter* 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 metaheuristics 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  $p \leftarrow 1..nrtasks$  do
     $best\_cost_p \leftarrow \infty$ ;
     $\alpha_p \leftarrow 0.7$ ;
     $fixed_p \leftarrow 0$ ;
    int  $count_p[N\_MAX][N\_MAX]$ ;
    int  $\varphi_p[N\_MAX]$ ; for  $i \leftarrow 1..N\_MAX$  do  $\varphi_p[i] \leftarrow 0$ ; end for;
    perm  $PP[K\_MAX]$ ;
    for  $iter \leftarrow 1..R$  do
      for  $t \leftarrow 0..K - 1$  do
         $PP[t] \leftarrow Random.Perm()$ ;
        int  $f \leftarrow Descent\_Search(PP[t])$ ;
        if  $f < best\_cost_p$  then  $best\_cost_p \leftarrow f$ ; end if;
        int  $f_{min}$ ;
        if  $stops == true$  then
          MPI.Allreduce(& $f$ , & $f_{min}$ , 1, MPI_INT, MPI_MIN,
            MPI_COMM_WORLD);
          if  $f_{min} \leq benchm\_opt$  then return  $f_{min}$ ;
        end if;
  
```

```

    for  $i, j \leftarrow 1..n$  do
       $count_p[i][j] \leftarrow 0$ ;
    end for;
    for  $t \leftarrow 1..K - 1$  do
      for  $i \leftarrow 1..fixed_p$  do
         $count_p[i][PP[i]] ++$ ;
      end for;
    end for;
    if  $communication == true$  then
      int  $new\_count[N\_MAX][N\_MAX]$ ;
      MPI.Allreduce( $count_p$ ,  $new\_count$ ,  $(n + 1) * (n + 1)$ ,
        MPI_INT, MPI_SUM, MPI_COMM_WORLD);
      for  $i, pos \leftarrow 1..n$  do
         $count_p[i][pos] \leftarrow new\_count[i][pos] / nrtasks$ ;
      end for;
    end if;
     $\alpha$  change  $\alpha$  if it is too big or too small, i.e. no elements is fixed or too
    many are fixed
    AutoTune(& $\alpha$ );
    for  $pos, i \leftarrow 1..n$  do
      if  $count_p[i][pos] / K > pe$  then
         $fixed ++$ ;
         $\varphi_p[i] ++$ ;
      end if;
    end for;
    for  $i \leftarrow 1..n$  do
      if  $\varphi_p[i] > MAX\_AGE$  then
         $\varphi_p[i] \leftarrow 0$ ;
         $fixed --$ ;
      end if;
    end for;
  end for;
  return  $f$ ;
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^0 . In each iteration for the current solution π^t the neighborhood $\mathcal{N}(\pi^t)$ is determined. Next, from the neighborhood the best element π^{t+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^t (in iteration i) includes quadruples (a, l, α, φ) , where a is an element of the set $N = \{1, 2, \dots, n\}$, 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 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, 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 F connected with the operations of inserting and deleting. Function Φ is determined by an auto-tune function. Function F is fixed experimentally as a constant.

Fixing elements. Let $P^i = \{\pi_1, \pi_2, \dots, \pi_K\}$ be a population of K elements in the iteration i . For each permutation $\pi_j \in P^i$, applying the local search algorithm (*LocalOpt*(π_j) procedure), a set of local minima $LM^i = \{\hat{\pi}_1, \hat{\pi}_2, \dots, \hat{\pi}_K\}$ is determined. For any permutation $\hat{\pi}_j = (\hat{\pi}_j(1), \hat{\pi}_j(2), \dots, \hat{\pi}_j(n))$, $j = 1, 2, \dots, K$, let be $count(a, l) = |\{\hat{\pi}_j \in LM^i : \hat{\pi}_j(l) = a\}|$. 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 (i.e. $a \in LM^i$) and

$$\alpha = \frac{count(a, l)}{K} \geq \Phi(i),$$

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

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

Auto-tune of the acceptance level Φ . Function Φ is defined so that for each iteration i $0 < \Phi(i) \leq 1$. It is possible that no element is acceptable to be fixed in an iteration. To prevent from this an auto-tune procedure for Φ value is proposed. In each iteration i , if

$$\max_{a, l \in \{1, 2, \dots, n\}} \frac{count(a, l)}{K} < \Phi(i)$$

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

$$\Phi(i) \leftarrow \max_{a, l \in \{1, 2, \dots, n\}} \frac{count(a, l)}{K} - \epsilon,$$

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).

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

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.

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 $||s_{ij}|| \sum w_i T_i$ 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], Lin and Ying [7] and Liao and Juan [6], see Table 2).

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_{total} (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 $1|s_{ij}|\sum w_i T_i$. The new 81 upper bounds are marked by a bold font.

Nr	F_{ParPTM}	PRD	Nr	F_{ParPTM}	PRD
1	618	-36,81%	61	76108	-4,73%
2	5061	-22,01%	62	44769	-6,46%
3	1769	-24,66%	63	75317	-4,45%
4	6389	-23,13%	64	92591	-3,93%
5	4662	-16,84%	65	126696	-6,07%
6	7207	-12,58%	66	59685	-6,82%
7	3647	-16,10%	67	29390	-15,79%
8	143	-56,27%	68	22120	-16,22%
9	6482	-14,69%	69	71118	-5,70%
10	1943	-20,73%	70	75102	-7,51%
11	3898	-25,94%	71	148230	-8,06%
12	0	0,00%	72	45543	-20,01%
13	4929	-19,81%	73	29045	-20,35%
14	2967	-24,71%	74	30503	-20,34%
15	1321	-54,68%	75	21602	-30,27%
16	4326	-35,54%	76	55378	-18,02%
17	127	-72,51%	77	33404	-17,64%
18	1337	-46,82%	78	20294	-19,16%
19	0	-100,00%	79	117596	-6,54%
20	2983	-28,86%	80	18620	-41,53%
21	0	0,00%	81	384383	-0,71%
22	0	0,00%	82	410257	-0,78%
23	0	0,00%	83	458844	-1,55%
24	1051	-41,32%	84	330022	-0,49%
25	0	0,00%	85	555065	-0,63%
26	0	0,00%	86	362677	-0,85%
27	0	-100,00%	87	398551	-1,11%
28	0	-100,00%	88	433244	-0,83%
29	0	0,00%	89	410739	-1,48%
30	0	-100,00%	90	402078	-1,19%
31	0	0,00%	91	342096	-1,46%
32	0	0,00%	92	361921	-1,05%
33	0	0,00%	93	407915	-0,62%
34	0	0,00%	94	333588	-0,81%
35	0	0,00%	95	521836	-1,15%
36	0	0,00%	96	462757	-0,35%
37	436	-81,89%	97	413089	-1,71%
38	0	0,00%	98	527603	-0,92%
39	0	0,00%	99	368353	-1,72%
40	0	0,00%	100	436004	-1,33%
41	69252	-5,36%	101	353018	-0,79%
42	58111	-6,06%	102	493072	-0,62%
43	146510	-2,32%	103	378864	-0,34%
44	35462	-8,43%	104	358033	-1,10%
45	59085	-5,86%	105	350806	-23,13%
46	35080	-7,66%	106	454769	-1,12%
47	73412	-4,89%	107	352766	-1,09%
48	65011	-5,67%	108	461828	-1,34%
49	78005	-7,29%	109	413004	-0,68%
50	31764	-12,34%	110	419437	-0,44%
51	50459	-13,85%	111	344532	-1,77%
52	97052	-7,89%	112	372287	-1,36%
53	88952	-6,81%	113	260093	-1,18%
54	124229	0,54%	114	469611	-0,76%
55	67969	-11,00%	115	463002	0,60%
56	77051	-12,86%	116	535967	-0,79%
57	67339	-4,37%	117	505416	-2,54%
58	47184	-15,02%	118	354576	-0,84%
59	53409	-9,57%	119	577318	-1,14%
60	65111	-11,21%	120	398723	-0,24%
			Average		-13,12

Table 2. Average improvement rates (%) of the SA, GA and TS approaches from Lin and Ying [7] compared to the proposed ParPTM approach. Standard deviation of the ParPTM results σ^{ParPTM} is determined over 10 runs.

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

As we can see in Table 3 the cooperative model of the ParPTM has shorter real times of execution (t_{total}) than the independent model. Also the time consumed by all the processors (t_{cpu}) 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 tabu search, simulated annealing as well as classical genetic algorithms.

References

- Bożejko W. and M. Wodecki, Evolutionary Heuristics for Hard Permutational Optimization Problems, International Journal of Computational Intelligence Research, Vol. 2, Issue 2, Research India Publications (2006), 151-158.
- Cicirello V.A. and S.F. Smith, Enhancing stochastic search performance by value-based randomization of heuristics. Journal of Heuristics 11 (2005), 5-34.
- Cicirello V.A., Non-Wrapping Order Crossover: An Order Preserving Crossover Operator that Respect Absolute Position, 8th Annual Genetic and Evolutionary Computation Conference GECCO 2006, ACM Press (2006), 1125-1131.

Table 3. Total time of the ParPTM, the algorithm stops when the benchmark is achieved. Times per 120 instances.

Processors	cooperative			independent		
	APRD	$t_{total}(s)$	$t_{cpu}(s)$	APRD	$t_{total}(s)$	$t_{cpu}(s)$
1	1.48%	5658	5655	1.48%	5647	5645
2	0.65%	5383	10765	0.60%	5643	11287
4	-0.26%	5580	22323	-0.17%	17836	53516
6	-0.74%	5400	32283	-0.73%	8548	52516
8	-0.32%	5218	41753	-0.97%	12129	83196
12	-1.13%	5065	60722	-1.25%	5980	67995
16	-1.78%	6865	105670	-0.80%	20124	238615
average	-0.23%	5595.6	39881.6	-0.26%	10843.9	73252.9

Table 4. Convergence of the ParPTM, constant number of iterations R=10. Times per 120 instances.

Processors	cooperative			independent		
	APRD	$t_{total}(s)$	$t_{cpu}(s)$	APRD	$t_{total}(s)$	$t_{cpu}(s)$
1	-0.73%	9462	9459	-0.70%	8113	8113
2	-2.09%	9669	19334	-1.76%	8310	16621
4	-3.48%	10124	40166	-2.94%	11535	39998
6	-4.20%	11963	67905	-4.13%	16916	84066
8	-4.47%	10479	83108	-4.53%	12058	88930
12	-5.00%	10311	123602	-5.01%	8770	105131
16	-5.47%	10329	165150	-5.29%	8732	139760
average	-3.63%	10333.9	72674.9	-3.48%	10633.4	68945.6

- Gagné C., W.L. Price and M. Gravel, 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** (2002), 895-906.
- Lee Y.H., K. Bhaskaran and M. Pinedo, A heuristic to minimize the total weighted tardiness with sequence-dependent setups, *IIE Transactions* **29** (1997), 45-52.
- Liao C.-J. and H. C. Juan, An ant optimization for single-machine tardiness scheduling with sequence-dependent setups, *Computers & Operations Research* **34** (2007), 1899-1909.
- Lin S.-W. and K.-C. Ying, Solving single-machine total weighted tardiness problems with sequence-dependent setup times by meta-heuristics, *International Journal of Advanced Manufacturing Technology* (on line), DOI 10.1007/s00170-006-0693-1 (2006).
- Tan K.C., R. Narasimban, P.A. Rubin and G.L. Ragatz, A comparison on four methods for minimizing total tardiness on a single processor with sequence dependent setup times, *Omega* **28** (2000), 313-326.

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