# Implementing a Parallel Simulated Annealing Algorithm

**Conference Paper** · September 2009

DOI: 10.1007/978-3-642-14390-8_16 · Source: DBLP



**3 authors**, including:

Zbigniew Czech
Silesian University of Technology
**59** PUBLICATIONS **819** CITATIONS
SEE PROFILE

Rafał Skinderowicz
University of Silesia in Katowice
**25** PUBLICATIONS **118** CITATIONS
SEE PROFILE

# Implementing a parallel simulated annealing algorithm

Zbigniew J. Czech[1,2], Wojciech Mikanik[1], and Rafał Skinderowicz[2]

[1] Silesia University of Technology, Gliwice, Poland
{Zbigniew.Czech,Wojciech.Mikanik}@polsl.pl
[2] University of Silesia, Sosnowiec, Poland
Rafal.Skinderowicz@us.edu.pl

**Abstract.** The MPI and OpenMP implementations of the parallel simulated annealing algorithm solving the vehicle routing problem (VRPTW) are presented. The algorithm consists of a number of components which co-operate periodically by exchanging their best solutions found to date. The objective of the work is to explore speedups and scalability of the two implementations. For comparisons the selected VRPTW benchmarking tests are used.

**Key words.**
parallel simulated annealing, MPI library, OpenMP interface, vehicle routing problem with time windows

## 1 Introduction

This work presents two implementations of a parallel simulated annealing algorithm. The algorithm, consisting of a number of components which co-operate periodically by exchanging their best solutions found to date, is used to solve the vehicle routing problem with time windows (VRPTW). A solution to the VRPTW is the set of routes beginning and ending at the depot which serves a set of customers. For the purpose of goods delivery there is a fleet of vehicles. The set of routes should visit each customer exactly once, ensure that the service at any customer begins within the time window and preserve the vehicle capacity constraints. The VRPTW belong to a large family of the vehicle routing problems (VRP) which have numerous practical applications.

The MPI and OpenMP implementations of the parallel simulated annealing algorithm solving the VRPTW are presented. The objective of the work is to explore speedups and scalability of these two implementations. The results of the work extend our previous efforts concerning the application of parallel simulated annealing to solve the VRPTW. However in the earlier projects [1–6] the parallel execution of processes was simulated in a single processor, and the main goal of investigations was to achieve a high accuracy of solutions through co-operation of processes.

In section 2 the parallel simulated annealing algorithm is presented. Sections 3 and 4 describe the MPI and OpenMP implementations of the algorithm,

respectively. Section 5 contains the discussion of the experimental results. Section 6 concludes the work.

## 2 Parallel simulated annealing algorithm

The parallel simulated annealing algorithm comprises $p$ components which are executed as processes $\mathcal{P}_0$, $\mathcal{P}_1$, ..., $\mathcal{P}_{p-1}$ (Fig. 1). A process generates its own annealing chain divided into two phases (lines 6–18). A phase consists of a number of cooling stages, and a cooling stage consists of a number of annealing steps. The goal of phase 1 is to minimize the number of routes of the VRPTW solution, whereas phase 2 minimizes the total length of the routes.

```
1    parfor P_j, j = 0, 1, ..., p − 1 do
2      Set co-operation mode to regular or rare depending on a test set;
3      L := (5E)/p; {establish the length of a cooling stage}
4      Create the initial_solution using some heuristics;
5      current_solution_j := initial_solution; best_solution_j := initial_solution;
6      for f := 1 to 2 do {execute phase 1 and 2}
         {beginning of phase f}
7        T := T_{0,f}; {initial temperature of annealing}
8        repeat {a cooling stage}
9          for i := 1 to L do
10           annealing_step_f(current_solution_j, best_solution_j);
11         end for;
12         if (f = 1) or (co-operation mode is regular) then {ω = L}
13           co_operation;
14         else {rare co-operation: ω = E}
15           Call co_operation procedure every E annealing step
               counting from the beginning of the phase;
16         end if;
17         T := β_f T; {temperature reduction}
18       until a_f cooling stages are executed;
         {end of phase f}
19     end for;
20   end parfor;
21   Produce best_solution_{p−1} as the solution to the VRPTW;
```

**Fig. 1.** Parallel simulated annealing algorithm; constant $E = 10^5$ annealing steps

At every annealing step a neighbor solution is computed by migrating customers among the routes (line 2 in Fig. 2). In simulated annealing the neighbor solutions of lower costs are always accepted, where by accepting we mean that the neighbor solution becomes a current one in the annealing chain. The higher cost solutions are accepted with probability $e^{-\delta/T_k}$, where $T_k$, $k = 0, 1, \ldots, a_f$,

is a parameter called a temperature of annealing, and $a_f$ is a number of cooling stages in phase $f$, $f = 1$ or 2. The temperature drops from an initial value $T_{0,f}$ according to equation $T_{k+1} = \beta_f T_k$, where $\beta_f$ ($\beta_f < 1$) and $\delta$ denote some constants and an increase of the solution cost, respectively (line 17 in Fig. 1). A sequence of steps for which the temperature of annealing stays constant constitutes a cooling stage. The cost of solution $s$ to the VRPTW is defined in phase 1 of the parallel algorithm as: $cost_1(s) = c_1 N + c_2 D + c_3(r_1 - \bar{r})$, and in phase 2 as: $cost_2(s) = c_1 N + c_2 D$, where $N$ is the number of routes in solution $s$ (equal to the number of vehicles carrying out the service), $D$ is the total travel distance of the routes, $r_1$ is the number of customers in a route which is tried to be get rid of the current solution, $\bar{r}$ is an average number of customers in all routes, and $c_1$, $c_2$, $c_3$ are constants. The basic criterion of optimization is the number of routes, therefore $c_1 \gg c_2$ is fixed.

```
1    procedure annealing_step_f(current_solution, best_solution);
2      Create new_solution as a neighbor to current_solution
         (the way this step is executed depends on f);
3      δ := cost_f(new_solution) − cost_f(current_solution);
4      Generate random x uniformly in the range (0, 1);
5      if (δ < 0) or (x < e^{−δ/T}) then
6        current_solution := new_solution;
7        if cost_f(new_solution) < cost_f(best_solution) then
8          best_solution := new_solution;
9        end if;
10     end if;
11   end annealing_step_f;
```

**Fig. 2.** Annealing step procedure

```
1    procedure co_operation;
2      if j = 0 then Send best_solution_0 to process P_1;
3      else {j > 0}
4        receive best_solution_{j−1} from process P_{j−1};
5        if cost_f(best_solution_{j−1}) < cost_f(best_solution_j) then
6          best_solution_j := best_solution_{j−1};
7          current_solution_j := best_solution_{j−1};
8        end if;
9        if j < p − 1 then Send best_solution_j to process P_{j+1}; end if;
10     end if;
11   end co_operation;
```

**Fig. 3.** Procedure of co-operation of processes

The processes of the parallel algorithm co-operate with each other every $\omega$ annealing step passing their best solutions found to date (lines 12–16 in Fig. 1 and Fig. 3). The chain of annealing steps of process $\mathcal{P}_0$ is entirely independent (Fig. 4). The chain of process $\mathcal{P}_1$ is updated at steps $u\omega$, $u = 1, 2, \ldots, u_{\mathrm{m}}$, to the better solution between the best solutions found by processes $\mathcal{P}_0$ and $\mathcal{P}_1$ to date. Similarly, process $\mathcal{P}_2$ chooses as the next point in its chain the better solution between its own best and the one obtained from process $\mathcal{P}_1$. Thus the best solution found by process $\mathcal{P}_l$ is piped down for further enhancement to processes $\mathcal{P}_{l+1} \ldots \mathcal{P}_{p-1}$. Clearly, after step $u_m\omega$ process $\mathcal{P}_{p-1}$ holds the best solution, $X_b$, found by all the processes.

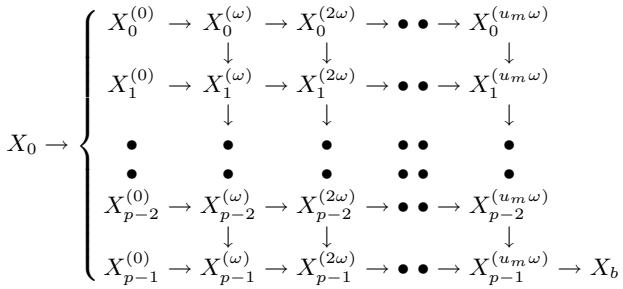


**Fig. 4.** Scheme of co-operation of processes ($X_0$ – initial solution; $X_b$ – best solution among the processes)

As mentioned before, the temperature of annealing decreases according to the equation $T_{k+1} = \beta_f T_k$ for $k = 0, 1, 2, \ldots, a_f$, where $a_f$ is the number of cooling stages. In this work we investigate two cases in establishing the points of process co-operation with respect to temperature drops. In the first case, of regular co-operation, processes interact at the end of each cooling stage ($\omega = L$) (lines 15–16 in Fig. 1). The number of annealing steps executed within a cooling stage is set to $L = (5E)/p$, where $E = 10^5$ is a constant and $p$, $p = 5$, 10, 15 and 20, is the number of processes (line 3 in Fig. 1). Such an arrangement keeps the parallel cost of the algorithms constant when different numbers of processes are used, provided the co-operation costs are neglected. Therefore in this case, as the number of processes becomes larger the length of cooling stages goes down, what means that the frequency of co-operation increases. In the second case, of rare co-operation, the frequency is constant and the processes exchange their solutions every $\omega = E$ annealing step (lines 17–18 in Fig. 1). For the number of processes $p = 10$, 15 and 20, the co-operation takes place after 2, 3 and 4 temperature drops, respectively.

## 3 The MPI implementation

All processes $\mathcal{P}_j$, $j = 0, 1, \ldots, p-1$, in the MPI implementation of the parallel simulated annealing algorithm, execute the code shown in Fig. 5. Like

```
      PROCESSES P_j, j = 0, 1, ..., p − 1;
1     Set co-operation mode to regular or rare depending on a test set;
2     L := (5E)/p; {establish the length of a cooling stage}
3     Create the initial_solution using some heuristics;
4     current_solution := initial_solution; best_solution := initial_solution;
5     for f := 1 to 2 do {execute phase 1 and 2}
        {beginning of phase f}
6       T := T_{0,f}; {initial temperature of annealing}
7       repeat {a cooling stage}
8         for i := 1 to L do
9           annealing_step_f(current_solution, best_solution);
10        end for;
11        if (f = 1) or (co-operation mode is regular) then {ω = L}
12          MPI_co_operation;
13        else {rare co-operation: ω = E}
14          Call MPI_co_operation procedure every E annealing step
              counting from the beginning of the phase;
15        end if;
16        T := β_f T; {temperature reduction}
17      until a_f cooling stages are executed;
        {end of phase f}
18    end for;
19    Produce best_solution of process P_{p−1} as the solution to the VRPTW;
```

**Fig. 5.** MPI implementation

in the general description of the algorithm (Fig. 1) the computation of processes is divided into phases 1 and 2. Depending on the co-operation mode, *regular* or *rare*, the processes interact every $L$ or $E$ annealing step, respectively. The MPI implementation of the co-operation between processes is presented in Fig. 6.

## 4 The OpenMP implementation

In the OpenMP implementation of the parallel simulated annealing algorithm, the current and best solutions found to date are kept in the shared memory in the arrays defined as follows

$$current\_solution,\ best\_solution\text{: } \textbf{array}[0\ ..\ p-1]\ \textbf{of}\ solution;$$

where $p$ is the number of threads, and *solution* is a data structure representing a VRPTW solution. The arrays are initialized in lines 5–9 in Fig. 7. The most

```
1    procedure MPI_co_operation;
2      if j = 0 then Send best_solution to process P_1 using MPI_Isend function;
3      else {j > 0}
4        Receive best_solution_r from process P_{j-1} using MPI_Recv function;
5        if cost_f(best_solution_r) < cost_f(best_solution) then
6          best_solution := best_solution_r;
7          current_solution := best_solution_r;
8        end if;
9        if j < p - 1 then
10         Send best_solution to process P_{j+1} using MPI_Isend function; end if;
11     end if;
12   end MPI_co_operation;
```

**Fig. 6.** Procedure of co-operation of MPI processes

time consuming part of the parallel simulated annealing algorithm in Fig. 1 is the loop in lines 9–11 where each process computes its own annealing chain. In the OpenMP implementation these chains are built by the threads within the parallel regions (lines 13–19 in Fig. 7). The co-operation of the threads is shown in Fig. 8. Summing up, only lines 5–9 and 13–19 of the OpenMP implementation shown in Fig. 7 are executed in parallel by the team of threads. The remaining parts of the implementation is executed sequentially by a single thread.

## 5 Experimental results

The MPI implementation was run on a cluster of 288 nodes which consisted of 1 or 2 Intel Xeon 2.33 GHz processors, each with 4 MB level 3 cache. Nodes were connected with the Infiniband DDR fat-tree full-cbb network (throughput 20 Gbps, delay 5 $\mu s$). The computer was executing a Scientific Linux. The source code was compiled using Intel 10.1 compiler and MVAPICH1 v. 0.9.9 MPI library. The OpenMP computations were performed on a ccNUMA parallel computer SGI Altix 3700 equipped with 128 Intel Itanium 2 processors clocked at 1.5 GHz, each with 6 MB level 3 cache. The computer was running Suse Linux Enterprise Server 10 operating system. The source code was compiled using Intel C compiler 11.0 with the following compilation flags: `-openmp -openmp-report2 -wd1572`.

The MPI and OpenMP implementations were compared on the selected VRPTW benchmarking tests proposed by Solomon [7]. The investigations reported in [6] indicated that Solomon's tests can be divided into 3 groups: I – tests which can be solved quickly (e.g. using $p = 20$ processes) to good accuracy with rare co-operation ($\omega = E$); II – tests which can be solved quickly ($p = 20$) but the frequency of co-operation should be relatively high (in this work we call it regular) (e.g. $\omega = E/4$ for $p = 20$) to achieve good accuracy of solutions; and III – tests whose solving cannot be accelerated as much as for the tests from groups I and II, i.e. the number of processes should be less than 20 to obtain

```
1   omp_set_num_threads(p); {set number of threads to p}
2   Set co-operation mode to regular or rare depending on a test set;
3   L := (5E)/p; {establish the length of a cooling stage}
4   Create the initial_solution using some heuristics;
5   #pragma omp parallel for
6   for j := 0 to p − 1 do
7     current_solution[j] := initial_solution;
8     best_solution[j] := initial_solution;
9   end for;
10  for f := 1 to 2 do {execute phase 1 and 2}
      {beginning of phase f}
11    T := T_{0,f}; {initial temperature of annealing}
12    repeat {a cooling stage}
13      #pragma omp parallel
14      {
15        j := omp_get_thread_num();
16        for i := 1 to L do
17          annealing_step_f(current_solution[j], best_solution[j]);
18        end for;
19      }
20      if (f = 1) or (co-operation mode is regular) then {ω = L}
21        OpenMP_co_operation;
22      else {rare co-operation: ω = E}
23        Call OpenMP_co_operation procedure every E annealing step
            counting from the beginning of the phase;
24      end if;
25      T := β_f T; {temperature reduction}
26    until a_f cooling stages are executed;
      {end of phase f}
27  end for;
28  Produce best_solution[p − 1] as the solution to the VRPTW;
```

**Fig. 7.** OpenMP implementation

```
1   procedure OpenMP_co_operation;
2     for j : = 1 to p − 1 do
3       if cost_f(best_solution[j − 1]) < cost_f(best_solution[j]) then
4         best_solution[j] := best_solution[j − 1];
5         current_solution[j] := best_solution[j − 1];
6       end if;
7     end for;
8   end OpenMP_co_operation;
```

**Fig. 8.** Procedure of co-operation of OpenMP threads

good accuracy of solutions. For comparisons of the implementations, two tests from each of these groups were chosen: tests R104, RC106 from group I, tests R107, R207 from group II, and tests RC104, RC202 from group III. The results of experiments are presented[3] in Tab. 1-2 and illustrated in Fig. 9-11.

**Table 1.** Results for tests R107 and R207; regular co-operation ($p$ – number of processes or threads, $\bar{\tau}$ – average execution time, $s$ – standard deviation of execution times, $\mathcal{S}$ – speedup, exp. – number of experiments)

| | | MPI implement. | | | | OpenMP implement. | | | |
|---|---|---|---|---|---|---|---|---|---|
| Test | $p$ | $\bar{\tau}$ | $s$ | $\mathcal{S}$ | exp. | $\bar{\tau}$ | $s$ | $\mathcal{S}$ | exp. |
| R107 | 1 | 730.8 | 9.9 | 1.00 | 100 | 1047.0 | 10.1 | 1.00 | 100 |
| | 5 | 149.4 | 1.7 | 4.89 | 100 | 263.5 | 3.9 | 3.97 | 100 |
| | 10 | 76.3 | 0.9 | 9.58 | 100 | 135.8 | 2.5 | 7.71 | 100 |
| | 15 | 50.8 | 0.6 | 14.38 | 100 | 91.0 | 0.9 | 11.51 | 100 |
| | 20 | 38.2 | 0.5 | 19.16 | 100 | 69.0 | 0.7 | 15.18 | 100 |
| R207 | 1 | 1113.7 | 36.9 | 1.00 | 100 | 1794.1 | 64.6 | 1.00 | 100 |
| | 5 | 231.5 | 8.6 | 4.81 | 100 | 434.0 | 21.6 | 4.13 | 100 |
| | 10 | 115.2 | 4.6 | 9.67 | 100 | 223.5 | 16.2 | 8.03 | 100 |
| | 15 | 77.5 | 2.4 | 14.38 | 100 | 144.5 | 3.6 | 12.41 | 100 |
| | 20 | 58.6 | 2.3 | 19.01 | 100 | 110.0 | 2.7 | 16.32 | 100 |

**Table 2.** Results for tests R104 and RC106; rare co-operation (meanings of columns as before)

| | | MPI implement. | | | | OpenMP implement. | | | |
|---|---|---|---|---|---|---|---|---|---|
| Test | $p$ | $\bar{\tau}$ | $s$ | $\mathcal{S}$ | exp. | $\bar{\tau}$ | $s$ | $\mathcal{S}$ | exp. |
| R104 | 1 | 755.6 | 7.8 | 1.00 | 1000 | 1028.2 | 10.3 | 1.00 | 234 |
| | 5 | 154.7 | 2.4 | 4.88 | 1000 | 262.6 | 8.0 | 3.92 | 200 |
| | 10 | 79.9 | 1.1 | 9.46 | 1000 | 134.8 | 5.3 | 7.63 | 200 |
| | 15 | 52.1 | 1.0 | 14.51 | 1000 | 90.5 | 3.2 | 11.36 | 200 |
| | 20 | 39.4 | 0.5 | 19.18 | 1000 | 68.7 | 1.7 | 14.97 | 200 |
| RC106 | 1 | 705.5 | 6.9 | 1.00 | 200 | 1072.0 | 20.5 | 1.00 | 100 |
| | 5 | 143.6 | 1.5 | 4.91 | 200 | 278.1 | 4.9 | 3.85 | 100 |
| | 10 | 73.1 | 0.7 | 9.65 | 200 | 139.9 | 29.8 | 7.66 | 100 |
| | 15 | 49.2 | 0.6 | 14.35 | 200 | 95.4 | 1.6 | 11.24 | 100 |
| | 20 | 35.5 | 0.6 | 19.33 | 200 | 73.8 | 1.0 | 14.54 | 100 |

It can be seen that the OpenMP implementation achieves slightly worse speedups than the MPI implementation. The experiments are still in progress

[3] Because lack of space we omit the results for tests RC104 and RC202.

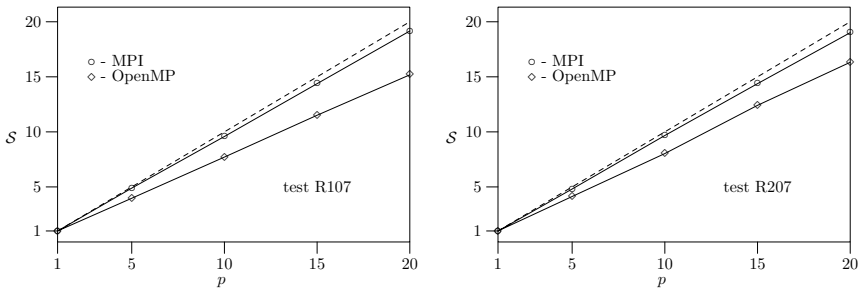


**Fig. 9.** Speedup $\mathcal{S}$ vs. number of processes (threads) $p$ for tests R107 and R207 (broken line shows the ideal speedup)

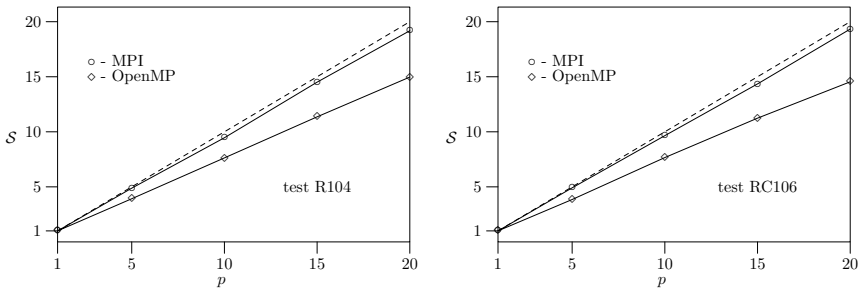


**Fig. 10.** Speedup $\mathcal{S}$ vs. number of processes (threads) $p$ for tests R104 and RC106

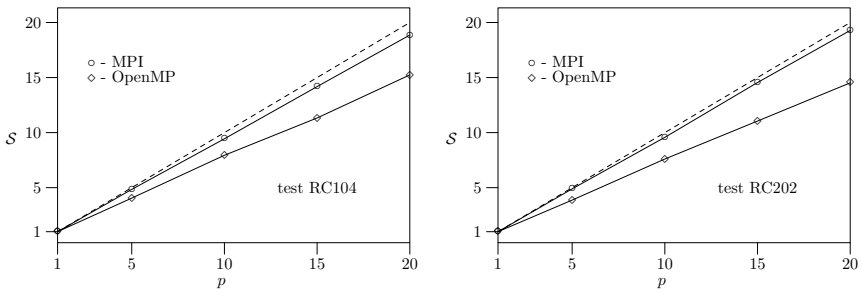


**Fig. 11.** Speedup $\mathcal{S}$ vs. number of processes (threads) $p$ for tests RC104 and RC202

and they concentrate now on the accuracy of solutions to the VRPTW. According to results presented in [6], the RC104 and RC202 tests are difficult to solve, i.e. for a larger number of processes the loss of accuracy of solutions to these tests is observed. The goal of the ongoing experiments is to clarify this issue using the MPI and OpenMP implementations.

## 6 Conclusions

The MPI and OpenMP implementations of the parallel simulated annealing algorithm to solve the VRPTW were presented. The experimental results obtained for the selected tests by Solomon indicated that the the MPI implementation scales better than the OpenMP implementation. The issue of the accuracy of the VRPTW solutions is to be cleared up.

## Acknowledgments

We thank the following computing centers where the computations of our project were carried out: Academic Computer Centre in Gdańsk TASK, Academic Computer Centre CYFRONET AGH, Kraków (computing grants 027/2004 and 069/2004), Poznań Supercomputing and Networking Center, Interdisciplinary Centre for Mathematical and Computational Modelling, Warsaw University (computing grant G27-9), Wrocław Centre for Networking and Supercomputing (computing grant 04/97). The research of this project was supported by the Minister of Science and Higher Education grant No 3177/B/T02/2008/35.

## References

1. Czarnas, P., Czech, Z.J., Gocyła, P., Parallel simulated annealing for bicriterion optimization problems, Proc. of the 5th International Conference on Parallel Processing and Applied Mathematics (PPAM 2003), (September 7–10, 2003), Czestochowa, Poland, Springer LNCS 3019/2004, 233–240.
2. Czech, Z.J., Czarnas, P., A parallel simulated annealing for the vehicle routing problem with time windows, Proc. 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing, Canary Islands, Spain, (January, 2002), 376–383.
3. Czech, Z.J., Wieczorek, B., Solving bicriterion optimization problems by parallel simulated annealing, Proc. of the 14th Euromicro Conference on Parallel, Distributed and Network-based Processing, (February 15–17, 2006), Montbéliard-Sochaux, France, 7–14 (IEEE Conference Publishing Services).
4. Czech, Z.J., Wieczorek, B., Frequency of co-operation of parallel simulated annealing processes, Proc. of the 6th Intern. Conf. on Parallel Processing and Applied Mathematics (PPAM 2005), (September 11–14, 2005), Poznań, Poland, Springer LNCS 3911/2006, 43–50.
5. Czech, Z.J., Speeding up sequential annealing by parallelization, Proc. of the International Conference on Parallel Computing in Electrical Engineering, PARELEC 2006, (September 13–17, 2006), Białystok, Poland, 349–354 (IEEE Conference Publishing Services).
6. Czech, Z.J., Co-operation of processes in parallel simulated annealing, Proc. of the 18th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2007), (November 13-15, 2006), Dallas, Texas, USA, 401–406.
7. Solomon, M.M., Algorithms for the vehicle routing and scheduling problems with time window constraints, Operations Research 35, (1987), 254–265, see also `http://w.cba.neu.edu /~msolomon /problems.htm`.