STUDIA INFORMATICA 2012
Volume 33 Number 1 (104)

Jakub NALEPA, Zbigniew J. CZECH
Silesian University of Technology, Institute of Informatics

A PARALLEL HEURISTIC ALGORITHM TO SOLVE THE VEHICLE ROUTING PROBLEM WITH TIME WINDOWS

Summary. The following article presents a parallel heuristic algorithm to solve the vehicle routing problem with time windows (VRPTW). The fleet size is minimized in the first phase and the traveled distance in the second one. The objective is to compare the accuracy of solutions obtained by the sequential and the parallel heuristics in the first phase. The influence of the population diversification and child generation on the accuracy is analyzed together with the speedups for the memetic algorithm in the second phase. The accuracy of solutions is defined as their proximity to the best known solutions of Gehring and Homberger's benchmarking tests.

Keywords: vehicle routing problem with time windows, heuristics, memetic algorithm, approximation algorithm

RÓWNOLEGŁY HEURYSTYCZNY ALGORYTM DLA ROZWIĄZYWANIA PROBLEMU TRASOWANIA POJAZDÓW Z OKNAMI CZASOWYMI

Streszczenie. W niniejszej pracy został przedstawiony równoległy heurystyczny algorytm dla rozwiązywania problemu trasowania pojazdów z oknami czasowymi. W pierwszej fazie jest minimalizowany rozmiar floty, a w drugiej fazie całkowita przebyta odległość. Celem pracy jest porównanie jakości rozwiązań otrzymanych za pomocą algorytmu sekwencyjnego oraz równoległego w pierwszej fazie. Przeanalizowany został wpływ zróżnicowania populacji i generowania rozwiązań potomnych na jakość rozwiązań wraz z przyspieszeniami dla algorytmu memetycznego drugiej fazy. Jakość rozwiązań jest oceniana na podstawie najlepszych obecnie znanych wyników dla problemów testowych Gehringa i Hombergera.

Słowa kluczowe: trasowanie pojazdów z oknami czasowymi, heurystyka, algorytm memetyczny, algorytm aproksymacyjny

1. Introduction

The vehicle routing problem with time windows (VRPTW) consists in finding a schedule for a fleet of homogenous vehicles servicing a set of geographically scattered customers. The capacities of the vehicles cannot be exceeded and the customers must be visited within their time windows. We consider the VRPTW as a hierarchical optimization problem. The primary objective is to minimize the total fleet size and the second one is to minimize the total distance traveled by the vehicles. This approach makes it possible to develop and optimize the algorithms for both phases independently.

In this paper we present a parallel heuristic algorithm to minimize the fleet size and a parallel memetic algorithm to minimize the traveled distance. A memetic algorithm is a hybridization of a genetic algorithm utilized for exhaustive exploration of the search space with a local optimization [5]. The heuristics is based on the improved sequential algorithm originally described by Nagata and Bräysy [3]. The sequential distance minimization was proposed by Nagata, Bräysy and Dullaert [4]. The objective of the work is to compare the accuracy of solutions obtained by the sequential and the parallel heuristics. The influence of the population diversification and child generation on the solutions is examined for the parallel memetic algorithm. We investigate the speedups of the OpenMP implementations.

Section 2 formulates the problem. The sequential and the parallel heuristics are discussed in Section 3. The memetic algorithm is described in Section 4. Section 5 describes the experimental results. Section 6 concludes the paper.

2. Formulation of the problem

Let $G = (V, E)$ be a directed graph with a set V of $N + 1$ vertices representing the customers and the depot, together with a set of edges $E = \{(v_i, v_j): v_i, v_j \in V, i \neq j\}$. The vertex v_0 represents the depot. The customers $v_i,\ i \in \{1,2, \dots, N\}$, are assigned their own non-negative service times s_i and demands d_i. The travel costs between each pair of travel points are given as $c_{i,j}$, where $i \neq j,\ i, j \in \{0,1, \dots, N\}$. Each customer and the depot have to be serviced within the time window $[e_i, l_i], i \in \{0,1, \dots, N\}$. The values e_i and l_i define the earliest and the latest time of starting the service respectively. A vehicle can arrive to a customer before its time window, but it has to wait until e_i.

The VRPTW is formulated as a problem of servicing N customers by a fleet of K homogenous vehicles of capacity Q. There is a single depot which is the start and the finish point of each route. A solution of the VRPTW, σ, is a set of routes in which all the customers are visited exactly once.

3. Minimizing the number of routes by the parallel heuristics

3.1. Description of a sequential algorithm

The route minimization heuristics reduces the number of routes in a feasible solution σ by one at a time. The process of removing the routes continues until the execution time reaches a defined maximum or the number of vehicles is minimal to ensure the feasibility [3]. The initial solution of the VRPTW consists of N routes. A randomly selected route (Fig. 1, line 1) is excluded from the current solution. The customers are inserted into the ejection pool (EP) (line 2). The penalty counters $p[i], i \in \{1,2,\dots,N\}$, are set to 1 (line 3) and indicate the reinsertion difficulties of the corresponding customers. The customers are taken from the EP applying the LIFO strategy (line 5).

function *RemoveRoute*(σ)
begin
1: Select and remove a random route from σ;
2: Initialize the *Ejection Pool* (EP) with a random permutation of V removed customers;
3: Initialize the penalty counters $p[i] := 1, i \in \{1,2,...,N\}$;
4: **while** EP ≠ Ø **and** (currIter < maxIter **or** epSize ≤ lastChanceSize) **and** epSize ≤ V + epAdd **and** epSteadyStateIter < maxIterFraction **and** currTime < maxTime **do**
5: Select and remove v_{ins} customer from EP using LIFO strategy;
6: **if** $N^{f}_{ins}(v_{ins}, \sigma) \neq \emptyset$ **then**
7: $\sigma := \sigma'$ selected randomly from $N^{f}_{ins}(v_{ins}, \sigma)$;
8: **else**
9: $\sigma := Squeeze(v_{ins}, \sigma)$;
10: **end if**
11: **if** v_{ins} is not in σ **then**
12: $p[v_{ins}] := p[v_{ins}] + 1$; {increasing the penalty counter}
13: Select $\sigma' \in N^{f}_{ej}(v_{ins}, \sigma)$ such that $P_{sum} = p[v^{(1)}_{out}] + ... + p[v^{(k)}_{out}]$ is minimal;
14: $\sigma := \sigma'$;
15: Insert the ejected customers $\{v^{(1)}_{out},..., v^{(k)}_{out}\}$ into EP;
16: $\sigma := Perturb(\sigma)$;
17: **end if**
18: **end while**
19: **if** EP ≠ Ø **then**
20: Restore σ to the initial solution;
21: **end if**
22: **return** σ;
End

Fig. 1. A sequential heuristic algorithm for minimizing the total number of routes
Rys. 1. Sekwencyjny heurystyczny algorytm minimalizacji liczby tras

All feasible insertion positions are determined (line 6). The change of the vehicle loads may be computed in constant time. Introducing forward and backward time window penalty

slacks allowed for constant-time verification of the time window penalties [3]. If the set of feasible insertion positions is not empty, then a random insertion is performed (line 7). The solution σ' with the customer v_{ins} becomes a new feasible (possibly partial) solution σ.

If the set $N_{ins}^{f}(v_{ins}, \sigma)$ is empty, then it is impossible to insert the customer v_{ins} without violating the constraints. The *Squeeze* method allows for creating the temporary infeasible solutions (line 9). The infeasible solution with the smallest value of the penalty function $F_p(\sigma)$ is chosen [4] and the attempts of restoring the feasibility are undertaken. If the squeezing fails, then the penalty counter of the customer is increased (line 12). The last approach of reinserting v_{ins} allows for ejecting other customers from the solution. A sum of the penalty counters P_{sum} is minimized to eject the customers that will be relatively easy to reinsert later. The customers that were inserted during the last l_{max} iterations are not considered for ejections [1]. The increasing number of the ejected customers k, $k \in \{1,2, \dots, k_{max}\}$, is tested. If at least one feasible ejection is found for a given k, then the other tests are skipped. If there are more ejections with the same P_{sum}, then one is chosen randomly [1]. A number of constant-time local moves are performed in *Perturb* method (line 16). The algorithm finishes if the EP is empty, the execution time exceeds the specified limit or the size of the EP is unacceptably large (line 4). The other breaking conditions are described in the next section.

3.2. Suggested modifications

The additional breaking condition of a main loop of the algorithm introduced in [1] addresses the maximal number of iterations *maxIter*. According to that, it should break even though the size of the EP is small and *currIter* > *maxIter*. A large number of iterations are usually performed if the number of routes is close to the optimum. Thus, it may be proficient to allow for additional loop executions if the EP is small and the probability of reinserting the customers is still high. The additional parameter *lastChanceSize* indicates the maximal number of customers allowed to reside in the EP for which the loop will continue despite of exceeding *maxIter*.

However, the size of the EP can stay constant during the execution for a long time. The maximal number of iterations in steady state *maxIterFraction* is introduced. It should vary with the maximal number of allowed iterations, thus *maxIterFraction* corresponds to a fraction of *maxIter*. If the EP size does not change during the *maxIterFraction* iterations, then the probability of a feasible customer reinsertion drops and the loop breaks.

The *Squeeze* function tries to restore the feasibility of an infeasible solution. The local search moves that may be calculated in constant time were used during the construction of $N_r(\sigma)$. If there are no feasible moves, then the linear-time moves are considered. This

approach is appropriate for smaller instances, since calculating the moves may become time-consuming for the larger number of customers. Additionally, it may be worth limiting the number of moves to test, e.g. search until n edge-exchanges are found.

A route to be removed is chosen randomly. The routes may be divided into two classes – the first containing the routes with the number of customers greater or equal to the average, and the second class with the others. Intuitively, it should be easier to reinsert the customers from the smaller route to the partial solution, thus choosing a random route from the first class gives a higher probability of feasible reinsertions. However, it is proficient to get rid of larger routes earlier and choose a random route from the second class, when the solution size is still far from the optimum.

If the squeezing fails, then the other ejections are tested. After a successful reinsertion of v_{ins} and ejections of other customers the solution is perturbed. In many cases perturbing is not necessary for efficient reinsertions. The perturbation may be omitted if the percentage of successful customer insertions without additional ejections is significant, e.g. 80%, for a given number of successive iterations (Fig. 1, lines 4-17). The main loop of the algorithm may fail due to e.g. exceeding the maximal number of trials. In this case, the perturbations of the partial solutions should be allowed. The number of moves during the perturbation may depend on the difficulty of reinserting the customers, i.e. it will increase with the decrease of the number of routes. The initial number of moves is multiplied by a constant factor, e.g. 2, every defined number of iterations until it reaches the maximal value.

3.3. Description of a parallel algorithm

The p available processors may be used either to achieve a higher accuracy of solution or to speed up the computations. In the first case, the goal is to obtain a solution that is closer to the global optimum. The main goal of the parallel heuristics is to improve the accuracy of solutions. The algorithm consists of p components denoted as $P_0, P_1, \ldots, P_{p-1}$. The initial solution is treated as the starting solution for each team member (Fig. 2, line 4). Each component calls *RemoveRoute* (line 8) in parallel with others *nrOfThreadSteps* times. The components co-operate to exchange the best solutions found up-to-date. The solutions are assessed according to their costs. The solutions with the smaller number of routes K are preferred. If the number of routes is equal, then the solution with the shorter total travel distance is considered better. The cost may be also defined as a weighted sum of the number of routes K and total travel distance T [2].The parallel algorithm guides the search towards the optimal solutions with respect to the number of vehicles and the travel distance.

The co-operation of components starts from thread P_0. Thread P_1 receives the solution σ_0 from thread P_0, and compares the costs of solution σ_1 with the received one. The solution

with the smaller cost replaces the current solution of thread P_1. Consequently, thread P_{p-1} compares the solution σ_{p-1} with σ_{p-2} received from thread P_{p-2}. Finally, the best solution is held by thread P_{p-1}. If the best solution is found by thread P_0, then all the components get σ_0. The co-operation scheme is presented in Fig. 3.

```
1:  Generate σinit;
2:  Set the co-operation mode;
3:  parfor Pi, i = 0,1,…,p – 1 do
4:      σi := σinit;
5:  end parfor
6:  while currAlgTime < maxAlgTime and currK > minK do
7:      parfor Pi, i =0,1,…, p – 1 do
8:          RemoveRoute(σi);
9:      end parfor
10:     Call TeamCooperation procedure;
11: end while
```

Fig. 2. A parallel heuristic algorithm for minimizing the total number of routes
Rys. 2. Równoległy heurystyczny algorytm minimalizacji liczby tras

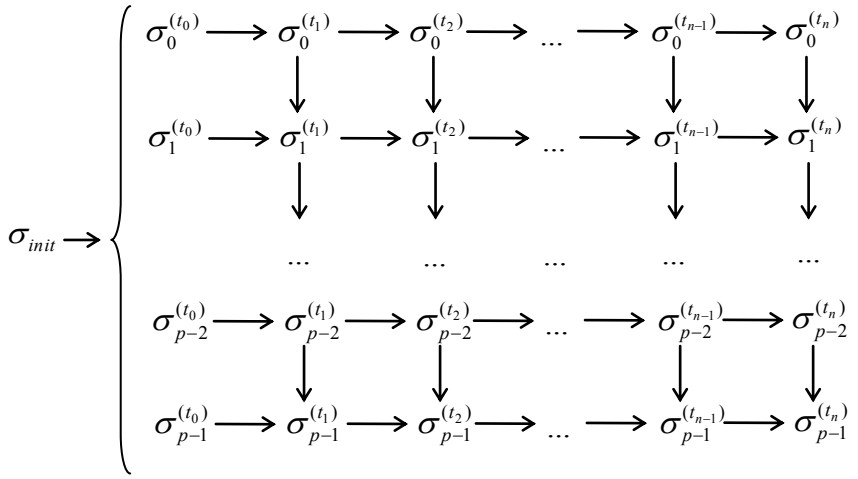

Fig. 3. The co-operation scheme
Rys. 3. Schemat kooperacji

The second variant of the co-operation scheme is cyclic and includes the communication between threads P_{p-1} and P_0. The better solution will be sent to thread P_0. The solution σ_0 should be updated only if the number of routes is larger than the number of routes in σ_{p-1}. Keeping the original solution σ_0 with the same number of vehicles by thread P_0 may prevent from having the same solution held by all the threads. Introducing the probability of replacing a worse solution by the solution received from the neighbor may further decrease the likelihood of having only one solution in the team. The probability of choosing the better solution should be large, but not equal to 1.

The number of steps that are executed in parallel before the co-operation must be determined sensitively. If the co-operation is too frequent, e.g. for large problem instances, the total parallel overhead becomes more significant and the execution time increases rapidly.

However, if the components co-operate rarely for small instances, the gain from parallelism is hardly noticeable.

The following possibilities for determining the co-operation frequency are introduced:

- Constant – *nrOfThreadSteps* is constant.
- Rare –*nrOfThreadSteps* is defined as a ratio of the problem size and a value of rare co-operation factor. After a number of co-operations *nrOfThreadSteps* is divided by the factor until it reaches a defined lower limit.
- Frequent – the scheme is similar to the rare co-operation, but the frequent co-operation factor is larger than the rare co-operation factor.
- Adaptive –*nrOfThreadSteps* is defined as a ratio of the problem size and a value of adaptive co-operation factor. The *nrOfThreadSteps* parameter is divided by the ratio of the last average time and the previous average time (if $t_{avg}^{prev} \neq 0$, otherwise *nrOfThreadSteps* is divided by the adaptive factor) of *RemoveRoute* executions.

4. Minimizing the total traveled distance by the memetic algorithm

4.1. Description of a sequential algorithm

The initial population of size N containing the feasible solutions with K routes is found by the parallel heuristics discussed in Section 3.3. If the maximal time of generating the initial population is exceeded, then the solutions already found are copied and perturbed until the population size reaches N. Each individual is chosen once as the parent p_A and p_B in a random order to generate the child solutions using the EAX operator [4] (Fig. 4, line 2). N_{ch} defines the number of children generated for each pair p_A and p_B. The feasibility of a child is restored by the *Repair* function if necessary (line 11) using the concept of local moves utilized while squeezing an infeasible solution in the route minimization heuristics. If the solution is feasible, then a number of moves are performed to improve its quality, i.e. to decrease the total travel distance (line 14). The moves are limited to the customers belonging to the routes modified by the EAX operator and the repairing procedure [4]. The total distance of a new solution is compared with the total distance of the best child found up-to-date (line 15). If a new solution is of higher quality, then the best child is updated (line 16). After generation of child solutions for N pairs of parents (line 6) the population is updated, i.e. the best individuals form a new population (lines 21-25). It is easy to see that the best child obtained for p_A and p_B replaces the first parent, not the worst individual in the population. Removal of p_A is motivated by the fact that the better individual replaces the worse with the similar characteristics to ensure the population diversification. The additional

termination condition addresses the steady state, i.e. the situation in which for a large number of subsequent generations the quality of the best individual is not improved. The algorithm finishes if the number of generations in the steady state is larger than the defined maximum, maximal number of generations *maxGen* is reached or the maximal execution time is exceeded (line 1). The best individual from the population is finally returned (line 27).

```
function MinimizeDistance(σ, N, N_ch)
begin
1:  while currTime < maxTime and currGen < maxGen and steadyGen < maxSteadyGen do
2:      Determine a random permutation r(i) ∈ {1,2,...,N};
3:      for i := 1 to N do
4:          σ_i^best := σ_i;
5:      end for
6:      for i := 1 to N do
7:          p_A := σ_r(i); p_B := σ_r((i+1)%N);
8:          for j := 1 to N_ch do
9:              σ_tmp := EAX(p_A, p_B);
10:             if F_p(σ_tmp) > 0 then
11:                 σ_tmp := Repair(σ_tmp);
12:             end if
13:             if F_p(σ_tmp) ≤ 0 then
14:                 σ_tmp := LocalSearch(σ_tmp);
15:                 if T(σ_r(i)^best) > T(σ_tmp) then
16:                     σ_r(i)^best := σ_tmp;
17:                 end if
18:             end if
19:         end for
20:     end for
        {Updating the population}
21:     for i := 1 to N do
22:         if T(σ_i) > T(σ_i^best) then
23:             σ_i := σ_i^best;
24:         end if
25:     end for
26: end while
27: return the best individual in the population;
end
```

Fig. 4. A memetic algorithm to minimize the total traveled distance
Rys. 4. Algorytm memetyczny minimalizacji przebytej drogi

4.2. Description of a parallel algorithm

The main goal of the presented parallel algorithm is to reduce the execution time without decreasing the quality of feasible solutions. The algorithm consists of p components denoted as $P_0, P_1, \ldots, P_{p-1}$. The main part of the memetic algorithm, i.e. generating the child solutions,

is the most computationally intensive (Fig. 4, lines 6-20). The iterations of the loop may be executed in parallel, since N_{ch} children are generated for the parents p_A and p_B independently. The best child solution is stored as $\sigma_{r(i)}^{best}$. Each individual in the population serves once as p_A and p_B during the combination stage, therefore different $\sigma_{r(i)}^{best}$ solutions are updated in every iteration. The N iterations are distributed between p threads, where $N \gg p$. The number of individuals in the population is usually large to avoid the similarities between the individuals. Once the loop finishes, the best child solutions are found and the current generation is updated. The cost, i.e. the total travel distance, of each individual in the current solution is compared with the cost of the best child. If the cost of the child σ_i^{best} is smaller, then the child becomes a new individual in the population and replaces the solution σ_i. The N solutions are compared independently (lines 21-25), therefore the iterations may be executed in parallel. Processing of the next generation of solutions starts with initializing of the set of the best child solutions (lines 3-5). Similarly, the loop iterations are independent and may be executed in parallel.

5. Experimental results

The algorithms were implemented in C++ using the OpenMP interface and were tested on Gehring and Homberger's problem instances. The code was compiled using Intel C++ Compiler 10.1.015 with `-fast` and `-openmp` flags. Calculations were carried out at a single node of Galera supercomputer at the Academic Computer Center in Gdańsk [7]. The computations were performed on the nodes with 16 GB RAM (2 GB/core) equipped with Intel Xeon Quad Core (2.33 GHz) processors with 12 MB of level 3 cache. The parameters used during the experiments are given in Table 1 and Table 3. The percentage of the nearest customers is limited for neighborhood calculations to decrease the execution time [3]. The number of additional customers allowed to reside in the EP has been proposed in [1]. The minimal number of local moves used during the solution perturbation should allow transforming a current solution to the neighboring, but still not too similar one. If the additional ejections are necessary for a successful customer insertion, then the number of moves is multiplied by *IrandFactor* to increase the probability of getting the new configurations. The maximal number of moves (for both stages) prevents from a rapid increase of the execution time. The maximal number of iterations in the steady state corresponds to a decent fraction of the maximal number of allowed algorithm iterations. The settings of the co-operation are given in Table 2. The EAX strategy [4] for recombination is chosen randomly. If a significant number of consecutive generations, e.g. 50, does not result in improving the best individual in a population, then the probability of further improvements

drops rapidly. A formula for the maximal execution time calculation of the memetic algorithm has been proposed in [4].

Table 1

The parameters of the route minimization heuristic algorithm

Parameter	Description	Value
neighborPerc	percentage of the nearest customers in the neighborhood	0.6
kmax	maximal number of customers to be ejected	3
lmax	number of iterations without ejecting a customer after the insertion	5
epAdd	additional customers allowed to reside in the EP	7
maxIter	maximal number of iterations of the first-phase algorithm	1000
maxIterFraction	maximal number of iterations in the steady state	$\frac{maxIter}{5}$
IrandMin	minimal number of feasible moves while perturbing	80
IrandMax	maximal number of feasible moves while perturbing	400
IrandFactor	update factor for the number of moves while perturbing	2
IrandFreq	frequency of updating the number of moves in iterations	50
maxTime	maximal time for reinsertions in *RemoveRoute* in seconds	300
maxTotalTime	maximal execution time in seconds	1200

Table 2

The co-operation frequency settings; CM – co-operation mode, CF – co-operation factor, UF – update factor, Ufr – update frequency, Mfr – minimal frequency

Size	CM	CF	UF	Ufr	Mfr
200	Frequent	10	2	4	1
400	Frequent	10	2	4	1
600	Adaptive	10	-	1	1
800	Rare	5	2	3	1
1000	Rare	5	2	3	1

A number of possible modifications and improvements have been suggested in Section 3.2. The exemplary average execution times of the sequential route minimization heuristic algorithm are given in Fig. 5. If the algorithm gets stuck in the local minima of the search space (e.g. for rc2_4_1), then the decreased initial number of local search moves results in the increase of the total number of iterations necessary to leave the local minimum. However, it is not always necessary to explore the vast solution space for large instances (e.g. for r1_10_2, c1_8_2) and a relatively small number of moves during the perturbation is enough to get satisfactory results. The average execution time has been decreased for a number of

instances that were relatively easy to solve (Fig. 5, b) and for time-consuming ones (a). However, the modifications are less suitable for the problems with solution spaces containing a large number of local minima.

Table 3

The parameters of the memetic algorithm to minimize the travel distance

Parameter	Description	Value
N_{ch}	number of child solutions generated for each pair of parents	20
IrandGen	maximal number of moves improving the child solution	100
genRandInit	number of moves used during copying and perturbing	50
maxTime	maximal execution time in minutes	$\frac{N_s N}{400}$
gMax	maximal number of generations without the improvement	50

Table 4

The percentage of the best known CVNs obtained with the sequential and the parallel heuristic algorithms

Class	OPT_{JNs}	OPT_{JNp}
C1	82%	84%
C2	70%	78%
R1	94%	94%
R2	100%	100%
RC1	100%	100%
RC2	84%	86%
Total	88%	90%

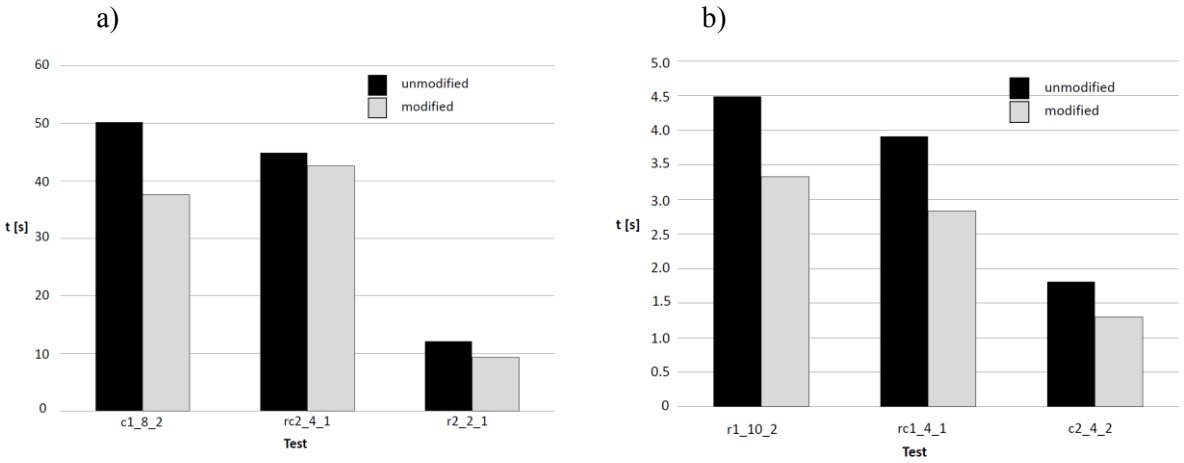

Fig. 5. The average execution time t (in seconds) of a sequential algorithm for minimizing the number of routes for 100 experiments tests: a) more time-consuming, b) less time-consuming

Rys. 5. Średni czas wykonania t (w sekundach) algorytmu sekwencyjnego minimalizacji liczby tras dla testów (100 eksperymentów): a) bardziej czasochłonnych, b) mniej czasochłonnych

The cumulative numbers of vehicles (CVNs), i.e. the number of vehicles servicing all instances, are presented in Table 4 for sequential and parallel algorithms. The number of vehicles was decreased for 16 instances, whereas the world's best results were obtained in 6 cases using the parallel heuristics. Therefore, in 271 out of 300 (90%) cases the benchmarking tests were solved to the current optimum with respect to the number of vehicles using the parallel algorithm. The parallel memetic algorithm significantly improved the current world's best result for the problem instance c1_8_2. The solution has been already published on the SINTEF website [6]. The travel distances in the solutions obtained with the parallel algorithm are successively decreased, since the higher-quality solutions replace the worse during the co-operation. The exemplary distances are given in Fig. 6.

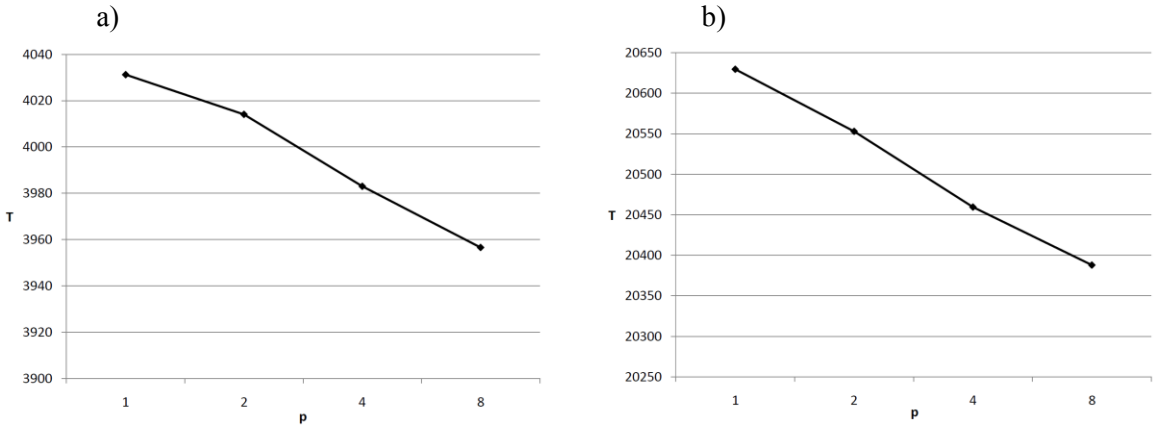

Fig. 6. The average distance T vs. number of threads p for 100 experiments tests: a) c1_2_1, b) r1_4_2

Rys. 6. Średnia długość trasy T w zależności of liczby wątków p dla testów (100 eksperymentów): a) c1_2_1, b) r1_4_2

The size of the population influences the execution time necessary to create a new generation of solutions. However, the probability of ending up with a set of similar individuals is lower in case of large populations. The problem of saturating the population is illustrated in Fig. 7. The experiments with the clustered customers have shown that the saturation of the population with similar individuals may occur relatively fast. The larger population should imply a larger population diversification. However, the populations with a large percentage of perturbed copies converge to the steady state fast and cannot be improved during the subsequent generations. If the number of individuals with similar configurations exceeds a certain threshold, then the population is in the diversity crisis [1].

The influence of the number of children N_{ch} on the population quality is presented in Fig. 7. Increasing N_{ch} results in the populations consisting of better individuals, since the child combines the best characteristics of parents. Obviously, the time necessary for creating the larger number of children increases. However, if the populations are of higher quality, then the smaller number of subsequent generations is necessary to converge to the similar results.

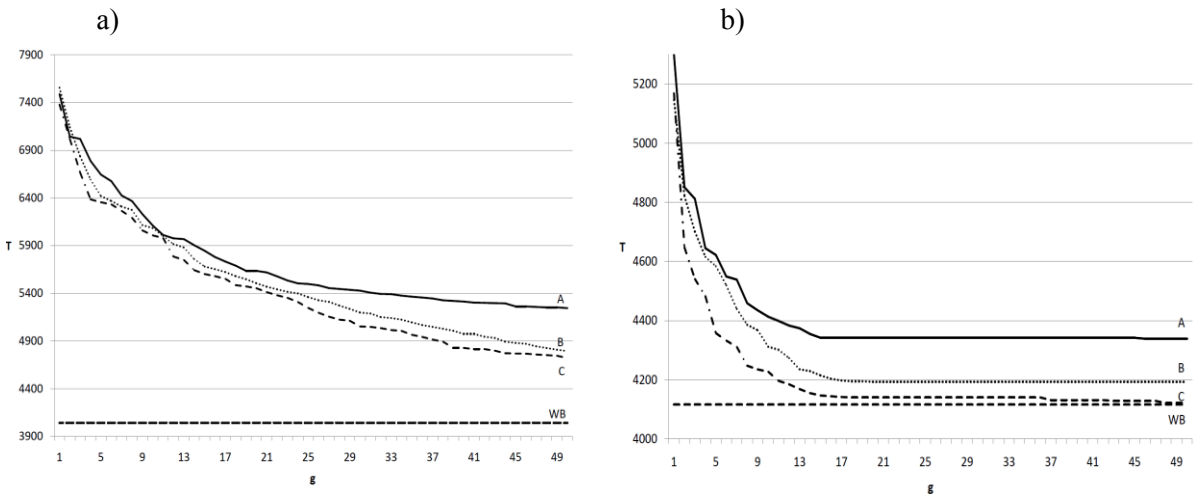

Fig. 7. Travel distance T of the best individual vs. generation g for different population sizes N (A-10, B-25, C-50), for tests: a) r1_2_2, b) c2_4_1; WB – the world's best known travel distance

Rys. 7. Najmniejsza długość trasy T w zależności od pokolenia g dla różnych wielkości populacji N (A-10, B-25, C-50), dla testów: a) r1_2_2, b) c2_4_1; WB – najlepszy obecnie znany wynik na świecie

The relative speedups obtained for two given problem instances are presented in Fig. 9. The population size is usually larger than the number of threads. The speedup depends not only on the problem size but also on its structure. If the number of generations required to obtain a minimal travel distance is large, then the relative speedup is almost ideal. However, if the solution converges to the minimum relatively fast, then the further improvements become difficult. It is possible to end up with a pair of parents p_A and p_B for which the children generation is more time-consuming than expected. The parallel overhead becomes more significant once the steady state is reached.

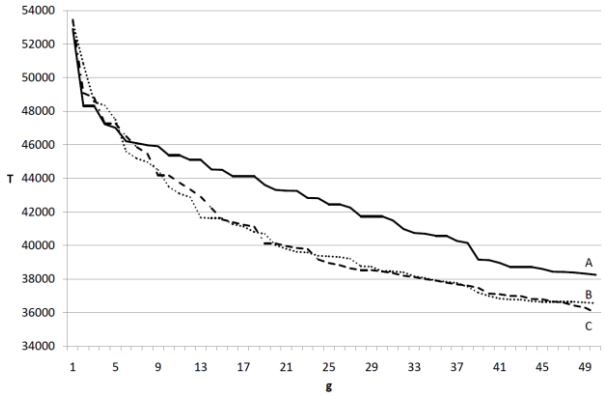

Fig. 8. Travel distance T of the best individual vs. generation g for test rc1_6_3 for different number of children N_{ch}(A-5, B-15, C-20)

Rys. 8. Długość trasy T najlepszego osobnika w zależności od pokolenia g dla testu rc1_6_3 dla różnej liczby potomków N_{ch}(A-5, B-15, C-20)

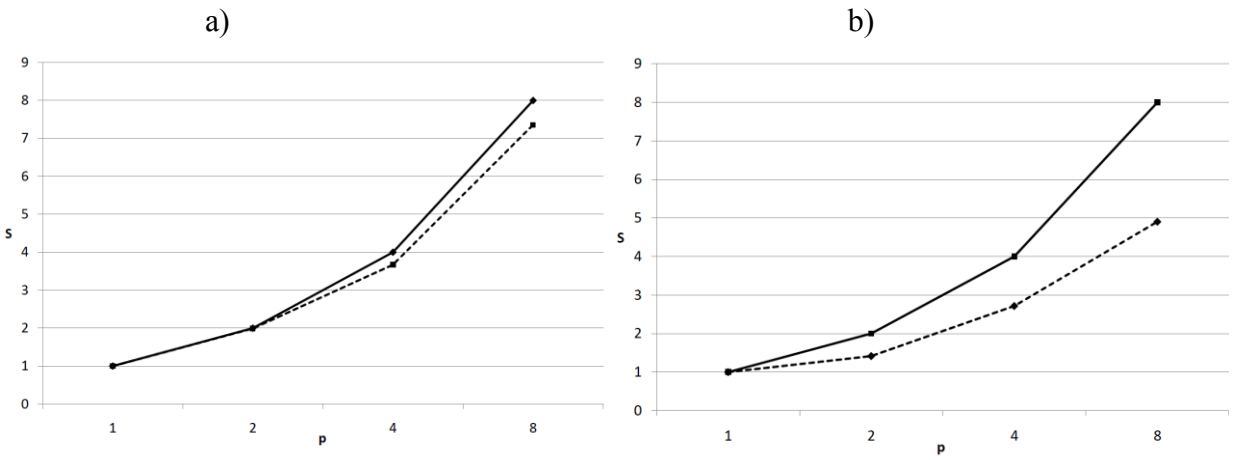

Fig. 9. Speedup S vs. number of threads p for tests: a) rc1_6_3, b) c1_2_1 (continuous line shows the ideal speedup)
Rys. 9. Przyspieszenie S w zależności od liczby wątków p dla testów: a) rc1_6_3, b) c1_2_1 (idelane przyspieszenie pokazano linią ciągłą)

6. Conclusions

The parallel heuristic algorithm for minimization of the fleet size has proven to be effective and competitive by solving 90% of problem instances to the current known optimum. The memetic algorithm for the distance minimization turned out to be powerful. A large number of parameters, both for the exploration and the exploitation of the search space, allow for adjusting the algorithm to the instance characteristics. The optimal assignment of parameters is to be cleared up during the further experiments. The experiments performed for various problem instances showed that the relative speedup is linear and close to the ideal one in many cases. The parallel algorithm significantly improved the world's best known solution of the clustered Gehring and Homberger's test c1_8_2 containing 800 customers.

A two-stage approach of solving the VRPTW makes it possible to combine the presented algorithms with other well-known heuristics, e.g. simulated annealing or tabu search. The parallel implementations for each stage can be compared to determine the most effective and scalable combination of heuristics addressing both objectives of the VRPTW.

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 grant 027/2004), Poznań Supercomputing and Networking Center, Interdisciplinary Centre for Mathematical and Computational Modeling, Warsaw University (computing grant G27-9), Wrocław Centre for Networking and Supercomputing (computing grant 30).

BIBLIOGRAPHY

1. Błocho M., Czech Z. J.: An improved route minimization algorithm for the vehicle routing problem with time windows. ZN Pol. Śl. Studia Informatica Vol. 30, No. 1(39), Gliwice 2010, p. 5÷19.
2. Czech Z. J., Mikanik W., Skinderowicz R.: Implementing a parallel simulated annealing algorithm. Proceedings of the 8th international conference on parallel processing and applied mathematics, 2010, Vol. 6067, p. 146÷155.
3. Nagata Y., Bräysy O.: A powerful route minimization heuristic for the vehicle routing problem with time windows. Operation Research Letters, 2009, Vol. 37, p. 333÷338.
4. Nagata Y., Bräysy O., Dullaert W.: A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows. Computers and Operations Research, 2010, Vol. 37, p. 724÷737.
5. Moscato P., Cotta C.: A Gentle Introduction to Memetic Algorithms in F. Glover (Ed.), Handbook of Metaheuristics, Kluwer 2003, p. 105÷144.
6. Problems and benchmarks: The world's best solutions for Gehring and Homberger's benchmark: http://www.sintef.no/Projectweb/TOP/Problems/VRPTW/
7. CI TASK – Galera: http://www.task.gda.pl/kdm/sprzet/Galera

Wpłynęło do Redakcji 22 listopada 2011 r.

Omówienie

W niniejszej pracy zaprezentowano równoległy algorytm heurystyczny rozwiązywania problemu trasowania pojazdów z oknami czasowymi (ang. *vehicle routing problem with time windows*). Równoległa heurystyka minimalizacji liczby tras została oparta na algorytmie przedstawionym w pracy [3] i ulepszonym w pracy [1]. Zaproponowane zostały kolejne modyfikacje, mające na celu zwiększenie prawdopodobieństwa otrzymania rozwiązania o wyższej jakości oraz skrócenie czasu wykonywania obliczeń. Do najistotniejszych ulepszeń należą:

- wprowadzenie maksymalnego rozmiaru puli usuniętych klientów (ang. *ejection pool*), dla którego algorytm nie zostanie przerwany pomimo przekroczenia maksymalnej liczby iteracji,
- wprowadzenie koncepcji stanu ustalonego puli usuniętych klientów,
- zdefiniowanie dwóch klas tras – z liczbą klientów większą lub równą liczbie średniej oraz liczbą klientów mniejszą od średniej,
- zmodyfikowanie strategii urozmaicania otrzymanych rozwiązań (ang. *diversification strategy*).

W równoległej heurystyce kooperacja wątków ma na celu wymianę najlepszych rozwiązań oraz zmniejszenie ryzyka utknięcia w lokalnym minimum przestrzeni poszukiwań. Ulepszone heurystyki – sekwencyjna oraz równoległa – były testowane przy użyciu testów Gehringa i Hombergera. W 90% przypadków otrzymano rozwiązania z liczbą tras równą opublikowanym najlepszym wynikom na świecie używając algorytmu równoległego.

Całkowita długość przebytych tras została zminimalizowana przy użyciu równoległego algorytmu memetycznego, którego wersja sekwencyjna została opisana w artykule [4]. Zostały w nim przedstawione badania, mające na celu określenie wpływu zróżnicowania populacji rozwiązań oraz liczby rozwiązań potomnych na jakość kolejnych generacji. Przedstawiono przyspieszenia dla testów o różnych strukturach i właściwościach. Dla przypadku testowego c1_8_2 otrzymano rozwiązanie z całkowitą długością tras mniejszą od światowego minimum. Rozwiązanie zostało opublikowane na stronie norweskiej organizacji SINTEF (25 sierpnia 2011 r.) [6].

Adresses

Jakub NALEPA: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16,
44-100 Gliwice, Polska, jakub.nalepa@polsl.pl
Zbigniew J. CZECH: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16,
44-100 Gliwice, Polska, zbigniew.czech@polsl.pl