International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:2, 2007

An Agent-Based Approach to Vehicle Routing Problem

Dariusz Barbucha and Piotr Jędrzejowicz

Abstract**—** The paper proposes and validates a new method of solving instances of the vehicle routing problem (VRP). The approach is based on a multiple agent system paradigm. The paper contains the VRP formulation, an overview of the multiple agent environment used and a description of the proposed implementation. The approach is validated experimentally. The experiment plan and the discussion of experiment results follow.

Keywords**—** multi-agent systems, population-based methods, vehicle routing problem.

I. Introduction

Last years many researchers has focused on the field of multi-agent systems. A number of significant advances have been made in both the design and implementation of autonomous agents. Also a number of approaches based on intelligent agents have been proposed to solve different types of optimization problems (see for example [1], [12]).

One of the successful approaches to agent-based optimization is the concept of an asynchronous team (A-Team), originally introduced by Talukdar [14]. An A-Team is a collection of software agents that cooperate to solve a problem by dynamically evolving a population of solutions. It is especially dedicated for solving computationally difficult problems.

The area in which the the agent based approaches could be successfully used is transportation and logistics. A widely researched and probably most important combinatorial optimization problem in this area is a vehicle routing problem (VRP).

Till now only few approaches based on using intelligent agents solving some transportation scheduling problems were proposed. Some of them can be find in [4], [9], [15]. Others present the description of the systems offering much complex architecture. The survey of approaches based on autonomous intelligent agents solving some transportation logistics problems can be found in [6].

The paper proposes using the JADE-based A-Team environment called JABAT for solving the vehicle routing problem. The following sections contain the vehicle routing problem formulation, a short overview of the functionality and structure of the JABAT, a description of the proposed implementation, as well as the experiment plan and the discussion of experiment results.

Manuscript received February 28, 2007. The research was supported by grants: KBN grant no. 3 T11C 059 28 and Gdynia Maritime University grant.

D. Barbucha is with the Department of Information Systems, Gdynia Maritime University, Gdynia, Poland (corresponding author, e-mail: barbucha@am.gdynia.pl).

P. Jędrzejowicz is with the Department of Information Systems, Gdynia Maritime University, Gdynia, Poland (e-mail: pj@am.gdynia.pl).

II. Vehicle Routing Problem

Let $G = (V, E)$ be an undirected graph, where $V = \{0, 1, \ldots, N\}$ is the set of nodes and E is a set of edges. Node 0 is a central depot with NV identical vehicles of capacity W. Each other node $i \in V - \{0\}$ denotes customer with a non-negative demand d_i. Each link $(i, j) \in E$ denotes the shortest path from customer i to j and is described by the cost c_{ij} of travel from i to j by shortest path $(i, j = 1 \ldots, N)$. It is assumed that $c_{ij} = c_{ji}$. The goal is to find vehicle routes which minimize total cost of travel (or travel distance) and satisfy following constraints:

- each route starts and ends at the depot,
- each customer is serviced exactly once by a single vehicle,
- the total load on any vehicle associated with a given route does not exceed vehicle capacity.

In addition to the vehicle capacity constraint, in some problems, a further limitation is imposed on the total route duration. In such case t_{ij} $(i, j = 1 \ldots, N)$ is defined to represent the travel time for each edge (i, j), and t_i $(i = 1 \ldots, N)$ represents the service time at any vertex i. It is required that the total duration of any route should not exceed a preset bound T.

There have been important advances in the development of exact and approximate algorithms for solving VRP. Because of the fact that this problem is computationally difficult, most of them are of heuristic nature. The simplest and fastest are constructive heuristics (savings algorithms, insertion algorithms, etc.) but the quality of solution is not satisfactory. On the other hand, metaheuristics (evolutionary algorithms, tabu search, ant algorithms) provide much better solutions, especially in case of large-scale instances.

For a recent survey of algorithms for solving classical VRP see [10], [16].

III. Overview of JABAT

JABAT is a middleware allowing to design and implement an A-Team architecture for solving various combinatorial optimization problems. The problem-solving paradigm on which the proposed system is based can be best defined as the population based approach.

The JABAT produces solutions to combinatorial optimization problems using a set of agents, each representing an improvement algorithm. To escape getting trapped into a local optimum an initial population of solutions called individuals is generated or constructed. Individuals forming an initial population are, at the following computation stages, improved

by independently acting agents, thus increasing chances for reaching a global optimum [2].

Main functionality of the proposed environment includes organizing and conducting the process of search for the best solution. It involves a sequence of the following steps:

1) Generating an initial population of solutions.
2) Applying solution improvement algorithms which draw individuals from the common memory and store them back after attempted improvement, using some user defined replacement strategy.
3) Continuing reading-improving-replacing cycle until a stopping criterion is met.

This functionality is realized mainly by two types of agents: *OptiAgents - OA* and *SolutionManagers - SMa*. The JABAT produces solutions to combinatorial optimization problems using a set of optimizing agents (*OptiAgents*), each representing a single optimizing algorithm. To escape getting trapped into a local optimum the initial population of solutions called individuals is generated. Individuals forming the initial population are, at the subsequent computation stages, improved by independently acting agents, thus increasing chances for reaching a global optimum. Each *SolutionManager* is responsible for finding the best solution of a single instance of the problem and maintains a single population of solutions of this problem. The agents of both types act in parallel and communicate with each other exchanging solutions that are either to be improved (when solutions are sent to *OptiAgent*) or stored back (when solutions are sent to *SolutionManager*) [3].

JABAT offers the possibility of defining how a solution received from *SolutionManager* may change the population of solutions. This is done through providing the *SolutionManager* with a replacement strategy. Such strategy in JABAT defines:

- how the initial population of individuals is created (for example how many solutions it consists of and what method is used to obtain an initial population) - implemented as the *initPopulationOfSolutions()* function,
- how solutions are chosen to be sent to optimizing agents - implemented as the *readSolution()* function,
- how solutions that has been received from optimizing agents are merged with the population - implemented as the *addSolution()* function,
- when the process of searching stops (after a predefined number of iteration, after reaching given solution, or when calculations do not improve current best solution for some special time, etc.) - implemented as the *stop()* function.

Fig. 1 presents an UML communication diagram which shows messages exchange between agents in the process of solving a task. After the *SolutionManager* initializes the population of solution, it waits for the information from an *OptiAgent* announcing its readiness to act. *SolutionManager* reads the required number of solutions from common memory and sends them to the *OptiAgent*, which improves it with its inbuilt improvement algorithm. Next, the improved solution is sent back to *SolutionManager* and added to the common memory. *SolutionManager* may also sent the solution to *SolutionMonitor*, which generates a report on the progress and

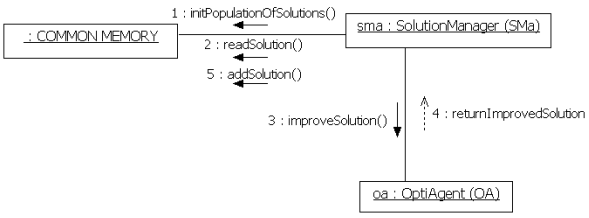

Fig. 1. UML communication diagram of solving process in JABAT

results. Then, again, the *OptiAgent* sends a message informing about its readiness. The whole process continues until some stopping criterion is met.

There are several different predefined strategies in the system to choose from. The simplest strategy is to allow the *SolutionManager* to draw a random individual from the common memory, forward it to some *OptiAgents* and replace the worst solution from the common memory if agents have been successful in improving the original one.

Apart from the above mentioned, there are also other agents working within the system - responsible for initializing, organizing the process of migrations agents, writing down the results etc. More detailed description of JABAT environment, its structure and full functionality was described in [2], [3].

IV. JABAT implementation for solving VRP

As it was mentioned earlier in order to use JABAT for solving an optimization problem several elements need to be defined. These include: representation of individuals, method of creating an initial population, the form of fitness function, improvement algorithms represented as optimization agents, and chosen strategy for managing a population of individuals.

A. Representation of an individual

There are several known representation of individuals for the VRP. One of the most common is a permutation of N numbers (customers), where the order of numbers reflects order in which customers are visited. This approach is based on the path representation of the Traveling Salesman Problem (TSP) but used for VRP requires procedure of splitting the individual on segments (routes) [13], for example by taking successive elements and forming the cluster (route) which satisfies the problem constraints (elements in each cluster do not exceed the capacity of the vehicle).

B. Creating the initial population

It is proposed to generate an initial population randomly and divide each individual by taking successive elements and forming the cluster (route) which satisfies the problem constraints (elements in each cluster do not exceed the capacity of the vehicle and the total cost of any route does not exceed a given limit). Note, that the order of elements in each cluster (order of visited customers by each vehicle) is determined by the order in which they are placed in the permutation representing the respective individual.

```
Algorithm OA_2-Opt
For each route r
     For each two edges (v_i, v_i+1) and (v_j, v_j+1), where all
     vertices v_i, v_i+1, v_j, v_j+1 are different
        1. Delete these edges
        2. Reconnect remaining edges by inserting new edges
           (v_i, v_j) and (v_i+1, v_j+1)
        3. If this replacement lead to improvement remember the
           obtained solution as the best improved solution
Return the best improved solution
```

Fig. 2. Pseudocode of the *OA_2-Opt* algorithm

```
Algorithm OA_StringCross
     1. Randomly choose two routes r_i and r_j from the current
        solution.
     2. Consider all possible crossing points p_i and p_j in r_i
        and r_j, respectively.
     3. Reconnect the routes by joining second part of r_j (from
        p_j to the end) to first part of r_i (from beginning to
        p_i) and the second part of r_i (from p_i to the end) to
        the first part of r_j (from beginning to p_j) (if this
        operation do not violate capacity constraints).
     4. If this replacement lead to improvement remember the
        obtained solution as the best improved solution, and go
        to the step 2.
     5. Return the best improved solution
```

Fig. 3. Pseudocode of the *OA_StringCross* algorithm

```
Algorithm OA_2-Lambda
For i =1...MAX_IT (MAX_IT = 100)
   1. Randomly choose two routes r_i and r_j from the current
      solution.
   2. Randomly choose w_i and w_j (w_i, w_j = 0,1,2) vertices from r_i
      and r_j, respectively.
   3. Try to move or exchange selected vertices between routes
      r_i and r_j (if this operation do not violate capacity
      constraints). Procedure of moving or exchanging is
      realized by removing and next inserting respective
      vertices using a mentioned earlier the cheapest insertion
      method.
   4. If this operation improve a solution, save it, and
      obtained solution becomes the current solution.
Return the best improved solution
```

Fig. 4. Pseudocode of the *OA_2-Lambda* algorithm

The proposed approach results in creation of the initial population which is based on polar representation of each vertex (customer) and uses an idea originated from *split* phase of the *sweep* algorithm of Gillett and Miller [7]. First, each vertex $i \in V$ is transformed from cartesian coordinates to polar coordinates (θ_i, ρ_i), where θ_i is the angle and ρ_i is the ray length.

Generation of each individual in the population starts from randomly choosing an arbitrary vertex i^* and assigning a value $\theta_i^* = 0$ to it. Next, the remaining angles centered at 0 from the initial ray $(0, i^*)$ are computed and the vertices are ranked in increasing order of their θ_i value. Resulting ranking determines an individual. The process of assignment vertices to clusters (routes) is similar to the previously described:

- Starting from the first unrouted vertex having the smallest angle, assign vertices to the cluster (route) as long as its capacity or the maximal route length is not exceeded.
- Repeat this process until end of the individual is reached.

Finally, the elements in each route are reordered using the cheapest insertion method [8]. It has been also decided to include into the initial population one individual which is obtained by solving the problem using savings method of Clarke and Wright [5].

C. Fitness

Each individual from the population is evaluated and value of its fitness is calculated. It is the cost of the whole route of all vehicles and is calculated as a sum of the costs related to each vehicles' route.

D. Optimization agents

In JABAT for VRP algorithms which operate on an individual are implemented as *OptiAgents*. It has been decided to implement four local improvement procedures:

- *OA_2-Opt* - is an implementation of the $2-opt$ local search algorithm and operates on a single route. For a given feasible solution, 2 edges are removed forming 2 disconnected segments. Next these segments are reconnected in all possible ways until a new feasible tour is obtained. The pseudocode of an implementation of the *OA_2-Opt* agent is shown on Figure 2.
- *OA_StringCross* - agent in which two strings (routes) of customers are exchanged by crossing two edges of two different routes. The pseudocode of an implementation of the *OA_StringCross* agent is shown on Figure 3.
- *OA_2-Lambda* - is an implementation of local search algorithms based on λ - interchange local optimization method [11]. It operates on many routes and is based on the move or interchange of customers between sets of routes. For each pair of routes from an individual a parts of routes of length less than or equal to λ are chosen and next these parts are shifted or exchanged, according to the selected operator OP. Possible operators are defined as pairs: (v, u), where $u, v = 1, \ldots, \lambda$. and denote the lengths of the part of routes which are moved or exchanged. For example, operator $(2, 0)$ indicates a shift a two customers from the first route to the second route, operator $(2, 2)$ indicates an exchange of two customers between routes. Typically, $\lambda = 2$ and such value was used in the *OA_2-Lambda* agent. The pseudocode of an implementation of the *OA_2-Lambda* agent is shown on Figure 4.
- *OA_2-LambdaC* - is similar to the *OA_2-Lambda* algorithm, but while the *OA_2-Lambda* is oriented for solving the instances in which the customers are uniformly arranged on the plane, this agent concentrates on instances in which the customers are clustered. The main idea of the algorithm is to remove vertices which are relatively far from the centroid of the current route and insert it to the route for each the distance between the vertices and the centroid of the other route is smaller. The pseudocode of an implementation of the *OA_2-LambdaC* agent is shown on Figure 5.

E. Replacement strategies

The system offers two main methods of reading a solution from the population:

- *Random* - random solution is chosen from the population,
- *Worst* - the worst solution from the population is selected,

```
Algorithm OA_2-LambdaC
For i =1...MAX_IT (MAX_IT = 100)
   1. Randomly choose two routes r_i and r_j from the current
      solution.
   2. Find the centroids of r_i and r_j.
   3. Randomly choose w_i and w_j (wi, wj = 0,1,2).
   4. Select w_i and w_j vertices from r_i and r_j, respectively,
      which are relatively far form the centroids of their own
      routes and remove them.
   5. Try to insert removed vertices from r_i to r_j and vertices
      removed from r_j to r_i using a mentioned earlier the
      cheapest insertion method (if this operation do not
      violate capacity constraints).
   6. If this operation improve a solution, save it, and
      obtained solution becomes the current solution.
Return the best improved solution
```

Fig. 5. Pseudocode of the *OA_2-LambdaC* algorithm

and four methods of adding an improved solution back to the population:

- *Random* - a randomly chosen solution from the current population is replaced by the received solution,
- *RandomNotBest* - a randomly chosen solution (except the best one) from the current population is replaced by the received solution,
- *Worse* - some randomly chosen worse solution from the current population is replaced by the received solution,
- *Worst* - the worst solution from the current population is replaced by the received solution).

The additional possible options in replacement strategies are also available. One of them is mechanism of blocking for a period a randomly chosen solution sent to optimization agents (*RandomBlocking*), whereas the other removes the worst solution from the population and adds a newly generated one if last consecutive five solutions received from the optimization agents did not improve existing solutions in population (*WorstBlocking*).

Also, the mechanism of proportional selection known from evolutionary algorithms has been implemented in JABAT. In evolutionary algorithms it is used for selecting potentially useful solutions for recombination in a way that is proportional to their fitness. So, candidate solutions with a higher fitness will be more likely to be selected but it does not guarantee that the fittest member goes through to the next generation. In JABAT implementation of selection strategy - *Select*, a solution is selected using the proportional selection and next send for improvement.

By combining possible settings of read and add operations, one obtains twenty strategies, which are summarized in Table I. It contains the name of strategy and *readSolution()* and *addSolution()* methods used in managing a population of solutions.

V. Computational Experiment

Computational experiment aimed at validating effectiveness of the approach and evaluating how the replacement strategy mix influences computation results.

JABAT implementation for VRP was tested on 7 test problems from *ORLibrary* benchmark set [17]. The selected problems (no. 1-5 and 11-12) contain 50-199 customers and have only capacity restriction. In the problems 1-5, the locations of the vertices are randomly generated over the plane while in the problems 11-12 vertices appear in clusters.

TABLE I
Replacement strategies available in JABAT implementation

Strategy	*readSolution()*	*addSolution()*
strategy01	*Random*	*Random*
strategy02	*Random*	*RandomNotBest*
strategy03	*Random*	*Worse*
strategy04	*Random*	*Worst*
strategy05	*Random*	*WorstBlocking*
strategy06	*Worst*	*Random*
strategy07	*Worst*	*RandomNotBest*
strategy08	*Worst*	*Worse*
strategy09	*Worst*	*Worst*
strategy10	*Worst*	*WorstBlocking*
strategy11	*RandomBlocking*	*Random*
strategy12	*RandomBlocking*	*RandomNotBest*
strategy13	*RandomBlocking*	*Worse*
strategy14	*RandomBlocking*	*Worst*
strategy15	*RandomBlocking*	*WorstBlocking*
strategy16	*Select*	*Random*
strategy17	*Select*	*RandomNotBest*
strategy18	*Select*	*Worse*
strategy19	*Select*	*Worst*
strategy20	*Select*	*WorstBlocking*

As it was mentioned in the previous section, the population of individuals was generated randomly with dividing procedure. The set of optimization agents included the four local improvement methods introduced earlier. The process of searching for the best solution stops when there are no improvements of the best solution during last 5 minutes of computation.

During the experiment population size was equal to 10 and the replacement strategy was determined by *readSolution()* and *addSolution()* functions. Twenty strategies presented in table I were tested. Each of the 7 instances of the problem was run for each possible strategy, in total giving 140 (7x20) test problems. Moreover, each test problem was repeatedly executed five times and mean results from these runs were recorded.

The proposed approach was evaluated using the mean relative error (MRE) from the best known solution, and the computation time (T).

All computations have been run on PC Pentium IV 2.8 GHz under the MS Windows XP Professional operating system.

The experiment results are shown in tables II, III and IV. The table II shows mean relative errors averages over all runs for each tested strategy and each problem. Together with the problem names the number of customers are presented in the brackets. The table shows also the average value (AVG) of errors for each strategy, as well as the average (AVG), minimum (MIN) and maximum (MAX) value of MRE for each problem separately. Table III presents the average time (in seconds) in which the best solution was reached for each problem and each strategy of population management and the table IV summarizes average values of MRE and computation time by grouping data from tables II and II by *readSolution()* and *addSolution()* operations.

Results obtained during the experiment and presented in table II show that the proposed approach produces good

TABLE IV
AVERAGE VALUE OF MRE (IN %) AND TIME (IN SEC.) FOR ALL KIND OF OPERATION READ AND ADD SOLUTION FROM/TO POPULATION

Operations		MRE	Time
readSolution()	*Random*	5.00%	33.60
	Worst	5.19%	25.60
	RandomBlocking	4.73%	40.75
	Select	4.97%	32.89
addSolution()	*Random*	4.83%	30.85
	RandomNotBest	4.53%	45.39
	Worse	5.04%	25.36
	Worst	5.32%	34.99
	WorstBlocking	5.14%	29.47

solutions of analyzed instances of the VRP. The average value of mean relative error is less then 5%, but it depends on the instance and for some instances is smaller. Minimal value of MRE observed during the experiment is close to 1% or less than 1% for most of the instances which confirms the usefulness of the proposed method and its effectiveness.

By looking at the columns of table II corresponding to the instances of the analyzed problem and trying to determine how the chosen strategy of management of population of solution implemented in the system influences the quality of solutions, one can see that although there are differences between solution obtained for particular instances there are no significant differences between results obtained for each instance using different strategies. Taking into account both, reading and adding operations, for all instances the best operation reading solution from population seems to be a *RandomBlocking* (randomly choosing a solution with blocking mechanism), and the best operation which adds improved solution back to the population is *RandomNotBest* (replacing randomly chosen solution except the best one). It is represented as strategy12 in the system and provides the best value of MRE.

By analyzing the MRE values presented in table IV one can conclude that the mean relative error calculated for strategies grouped by *readSolution()* and *addSolution()* operations is rather similar (4-5%) and it is difficult to point the best one. More detailed statistical analysis shows that the only one significant difference exists between *RandomNotBest* and *Worst* method of adding a solution to the population.

It can be easily observed there are significant differences of the average computation time needed for different replacement strategies tested. Analysis of average values of time presented in table III provides conclusion that there exists significant differences between *Worst* and *RandomBlocking* methods (*readSolution()* operation) and between *RandomNotBest* and *Random*, *RandomNotBest* and *Worse* and *RandomNotBest* and *WorstBlocking* (*addSolution()* operation). The smallest value of computation time for all instances was obtained for the *Worst* (the worst solution from the population is read from population) and *Worse* (some randomly chosen worse solution from the current population is replaced by the received solution) methods in one strategy (strategy08).

VI. CONCLUSIONS

In the paper an approach based on a multiple agent system paradigm was proposed for solving the vehicle routing problem (VRP). The validating experiment confirmed that the approach based on multiple agents may be useful for solving instances of the vehicle routing problem. Although results obtained by JABAT are slightly inferior in comparison with the best results obtained for example by some implementations of tabu search [10], the overall evaluation is positive thanks to several features typical for multiple agent systems. Among these one should note ability to increase computational efficiency through parallelization and possibility of using distributed environment.

REFERENCES

[1] M.E. Aydin and T.C. Fogarty, "Teams of autonomous agents for job-shop scheduling problems: An Experimental Study", *Journal of Intelligent Manufacturing*, 15(4), pp. 455-462, 2004.
[2] D. Barbucha, I. Czarnowski, P. Jędrzejowicz, E. Ratajczak, and I. Wierzbowska, "JADE-based A-Team as a tool for implementing population-based algorithms", in *Proc. of 6th IEEE International Conference on Intelligent System Design and Applications (ISDA 2006)*, Jinan, 2006, IEEE Press, vol. III, pp. 155-160.
[3] D. Barbucha, I. Czarnowski, P. Jędrzejowicz, E. Ratajczak, and I. Wierzbowska, "JABAT - An implementation of the A-Team concept", in *Proc. International Multiconference on Computer Science and Information Technology*, Wisa, 2006, PTI, vol. 1, pp. 235241.
[4] K. Burckert, H.J. Fischer, and G. Vierke, "Holonic transport scheduling with TeleTruck", *Journal of Applied Artificial Intelligence*, 14, pp. 697-725, 2000,
[5] G. Clarke and J.W. Wright, "Scheduling of vehicles from central depot to a numbes of delivery points", *Operations Research* 12, 1964, pp. 568-581.
[6] P. Davidson, L. Henesey, L. Ramstedt, J. Tornquist, and F. Wernstedt, "An analysis of agent-based approaches to transport logistics", *Transportation Research Part C*, 13, pp. 255-271, 2005
[7] B.E. Gillett, L.R. Miller, "A heuristic algorithm for the vehicle dispatch problem", *Operations Research* 22, pp. 240-349, 1974.
[8] B. Golden and W. Stewart, "Empirical analysis of heuristics", in *Traveling Salesman Problem*, E. Lawler, J. Lenstra, A. Rinnooy, and D. Shmoys, Eds. New York: Wiley-Interscience, 1985, pp. 207-249.
[9] J. Kozlak, J.C. Creput, V. Hilaire, and A. Koukam, "Multi-agent environment for dynamic transport planning and scheduling", in *Computational Science - ICCS'2004*, Bubak M. Albada, G.D.V., Sloot, P.M.A., Dongarra, J. (Eds.), Lecture Notes in Computer Science 3038, 2004, pp. 638-645.
[10] G. Laporte, M. Gendreau, J. Potvin, and F. Semet, "Classical and modern heuristics for the vehicle routing problem", *International Transactions in Operational Research*, 7, pp. 285-300, 2000.
[11] I.H. Osman, "Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem", *Annals of Operations Research*, vol. 41, pp. 421-451, 1993.
[12] H.V.D. Parunak, "Agents in overalls: Experiences and issues in the development and deployment of industrial agent-based systems", *International Journal of Cooperative Information Systems*, 9(3), pp. 209-228, 2000.
[13] Ch. Prins, "A simple and effective evolutionary algorithm for the vehicle routing problem", *Computers & Operations Research*, 31, pp. 1985-2002, 2004.
[14] S. Talukdar, L. Baeretzen, A. Gove, and P. de Souza, "Asynchronous teams: Cooperation schemes for autonomous agents", *Journal of Heuristics*, 4, pp. 295-321, 1998.
[15] S.R. Thangiah, O. Shmygelska, and W. Mennell, "An agent architecture for vehicle routing problem", in *Proc. of the ACM Symposium on Applied Computing (SAC'2001)*, Las Vegas, 2001, pp. 517-521.
[16] P. Toth, and D. Vigo, Ed. *The Vehicle Routing Problem*, SIAM Monographs Discrete Mathematics and Applications, SIAM:Philadelphia, 2002.
[17] ORLibrary, http://people.brunel.ac.uk/m̃astjjb/jeb/orlib/vrpinfo.html

TABLE II
THE MEAN RELATIVE ERROR (MRE) IN % FROM THE BEST KNOWN SOLUTION OBTAINED BY JABAT IMPLEMENTATION FOR VRP FOR SELECTED INSTANCES FROM ORLIBRARY

	PROBLEM							
STRATEGY	vrpnc1 (50)	vrpnc2 (75)	vrpnc3 (100)	vrpnc4 (150)	vrpnc5 (199)	vrpnc11 (120)	vrpnc12 (100)	AVG
strategy01	0.24%	9.45%	6.27%	8.09%	5.32%	2.38%	1.93%	4.81%
strategy02	0.40%	5.73%	6.07%	8.09%	8.40%	0.41%	1.15%	4.32%
strategy03	0.48%	9.36%	6.17%	8.09%	10.11%	0.43%	2.36%	5.28%
strategy04	0.34%	9.29%	6.27%	8.09%	8.25%	3.03%	5.23%	5.78%
strategy05	0.09%	9.46%	6.17%	8.09%	4.66%	2.86%	2.36%	4.81%
strategy06	1.73%	6.99%	6.01%	8.09%	9.48%	2.86%	2.36%	5.36%
strategy07	2.30%	6.77%	4.70%	7.04%	6.88%	2.86%	2.29%	4.69%
strategy08	2.21%	9.37%	5.63%	8.09%	7.74%	2.86%	2.36%	5.47%
strategy09	1.53%	5.79%	4.73%	8.09%	7.54%	2.86%	2.36%	4.70%
strategy10	2.01%	9.29%	6.17%	8.09%	5.96%	2.86%	5.72%	5.73%
strategy11	1.62%	9.15%	5.28%	7.17%	4.94%	1.40%	1.19%	4.39%
strategy12	0.57%	8.23%	6.16%	8.16%	4.91%	1.42%	0.63%	4.30%
strategy13	0.66%	8.00%	5.42%	8.09%	7.31%	0.82%	1.47%	4.54%
strategy14	0.44%	9.45%	6.27%	8.09%	4.67%	6.88%	2.36%	5.45%
strategy15	0.01%	9.28%	6.16%	8.09%	6.07%	2.86%	2.36%	4.98%
strategy16	0.72%	9.29%	6.17%	8.09%	5.24%	1.90%	1.93%	4.76%
strategy17	0.25%	9.46%	6.27%	8.35%	4.63%	2.38%	2.36%	4.81%
strategy18	0.75%	9.46%	6.17%	8.09%	4.34%	2.86%	2.36%	4.86%
strategy19	0.09%	9.28%	6.27%	8.09%	4.77%	2.86%	6.10%	5.35%
strategy20	0.00%	9.29%	6.17%	8.39%	6.78%	2.38%	2.36%	5.05%
AVG	0.82%	8.62%	5.93%	8.02%	6.40%	2.46%	2.56%	4.97%
MIN	0.00%	1.40%	1.63%	2.82%	3.37%	0.24%	0.13%	
MAX	4.59%	9.77%	8.91%	9.60%	10.11%	22.94%	21.10%	

TABLE III
AVERAGE TIME (IN SEC.) CONSUMED BY SYSTEM FOR OBTAINING THE BEST SOLUTION FOR EACH STRATEGY AND EACH PROBLEM

	PROBLEM							
STRATEGY	vrpnc1 (50)	vrpnc2 (75)	vrpnc3 (100)	vrpnc4 (150)	vrpnc5 (199)	vrpnc11 (120)	vrpnc12 (100)	AVG
strategy01	34.80	1.48	2.23	5.60	114.90	10.58	18.65	26.89
strategy02	55.95	99.28	9.25	4.70	89.63	50.03	72.33	54.45
strategy03	17.50	0.65	1.20	4.55	12.48	56.93	1.60	13.56
strategy04	38.23	0.68	1.23	5.90	77.15	91.00	38.68	36.12
strategy05	55.28	1.75	1.60	4.95	188.75	3.53	3.00	36.98
strategy06	19.40	83.25	32.65	3.65	17.93	2.93	1.08	22.98
strategy07	14.40	56.65	34.48	27.68	86.63	2.10	14.83	33.82
strategy08	9.13	0.98	22.08	5.78	41.15	2.00	1.93	11.86
strategy09	40.10	96.68	32.88	3.35	48.68	2.95	0.98	32.23
strategy10	47.68	1.25	1.93	4.70	130.25	2.80	1.23	27.12
strategy11	19.78	6.38	28.08	33.68	149.20	37.15	34.80	44.15
strategy12	47.73	26.73	1.33	4.30	143.75	52.45	93.05	52.76
strategy13	55.43	20.73	14.03	5.83	69.20	118.83	48.05	47.44
strategy14	60.45	1.75	1.78	4.78	198.98	2.58	1.68	38.85
strategy15	12.63	1.55	2.30	3.98	118.63	2.88	2.03	20.57
strategy16	21.33	1.78	2.23	4.53	135.05	19.88	20.93	29.39
strategy17	30.35	2.10	2.33	4.70	226.98	14.63	2.70	40.54
strategy18	15.93	1.75	1.60	4.98	170.45	2.95	2.30	28.56
strategy19	16.70	1.63	2.63	6.40	196.98	2.45	2.45	32.75
strategy20	49.28	1.48	1.58	5.48	143.85	28.45	2.28	33.20
AVG	33.10	18.54	10.50	8.50	121.24	24.76	16.60	33.32
MIN	0.32	0.36	0.54	0.81	2.69	0.58	0.48	
MAX	169.61	225.90	157.92	142.73	358.90	226.17	196.19	