Evolution of Vehicle Routing Problem Heuristics with Genetic Programming

Matija Gulić, Domagoj Jakobović
Protok d.o.o., Zagreb, Croatia, e-mail: matija.gulic.hr@gmail.com
University of Zagreb, Faculty of Electrical Engineering and Computing, e-mail: domagoj.jakobovic@fer.hr

***Abstract*—Increasingly complex variants of the vehicle routing problem with time windows (VRPTW) are coming into focus, alleviated with advances in the computing power. VRPTW is a combination of the classical traveling salesman and bin packing problems, with many real world applications in various fields – from physical resource manipulation planning to virtual resource management in the ever more popular cloud computing domain. The basis for many VRPTW approaches is a heuristic which builds a candidate solution that is subsequently improved by a search or optimization procedure. The choice of the appropriate heuristic may have a great impact on the resulting quality of the obtained schedules. In this paper we use genetic programming to evolve a suitable heuristic to build initial solutions for different objectives and classes of VRPTW instances. The results show great potential, since this method is applicable to different problem classes and user-defined performance objectives.**

***Index Terms*—vehicle routing problem with time windows, genetic programming, heuristic scheduling**

I. Introduction

Travel logistics have always been important to minimize expenses and maintain a good quality of service. A good formal representation of real-life transportation and service delivery problems is the capacitated vehicle routing problem with time windows (VRPTW). It is a problem of having to minimize the distance that vehicles need to travel in order to deliver a service with multiple realistic constraints regarding space, time and capacity. It has been shown that solutions for this problem could be applied to real-life domains and significantly improve their efficiency – up to 30% in some cases [1].

VRPTW is an NP-hard problem [2], so the aim is to find an approximate solution of high-enough quality. There are many solutions to the VRPTW and the most promising ones, such as [3], [4], apply the so-called optimization decomposition technique to divide the search into separate constraint dimensions. The method used in this paper also applies a decomposition based on separate vehicles and then uses parallel local search for optimization within each decomposed dimension.

However, for a decomposition approach to be successful, a good initial solution is needed which the decomposition algorithm could transform into a solution of an acceptable quality. For this purpose usually simple and fast greedy heuristics are used for creation of initial solutions that are further improved by decomposition operators. Here a choice must be made to select an appropriate heuristic, which may be highly dependent on the given performance objective, as different heuristics yield very large differences in the final solution quality [5].

Instead of manually selecting (and sometimes guessing) which heuristic would be suitable for initial solutions, we propose to evolve (i.e. automatically generate) an appropriate heuristic with the use of genetic programming (GP, [6]). Genetic programming has the ability to evolve any form of algorithm, simply by defining its building elements and a measure of algorithm quality (the fitness function). With this approach, we may create heuristics tailored to the problem at hand, regardless of the given performance objective and specific constraints [7][8]. The GP evolved heuristic does not produce the final solution, but the initial solutions obtained in this way may provide much better final solutions after a decomposition phase, which is investigated in the results. This paper is an extension of our previous project [5] where only the decomposition approach was presented, without the use of GP for the creation of initial solutions.

The rest of the paper is organized as follows. Section II gives a formal problem statement and introduces the considered model, as well as listing the related work in the field. The method of creating the solution with GP generated heuristic functions is described in Section III. Evaluation procedure and numerical results are described in Section IV. Section V gives possible future research directions and concludes the paper.

II. Vehicle Routing Problem with Time Windows

The vehicle routing problem with time windows is an extension of the well-known vehicle routing problem (VRP, defined in [9]). The VRP is described as follows: "A set of n customers must be serviced from a central office using vehicles of equal given capacity". Each customer must be served from exactly one vehicle. Usually two objective criteria are used, the primary to minimize the number of vehicles and the secondary to minimize the total travel distance. An additional constraint associated with the time windows extension is that every customer must be serviced within a given time frame. If a vehicle arrives earlier it must wait for the window opening time (waitTime). If the vehicle arrives after the end of the time window, the solution is not valid. Every customer has the following parameters defined:

- readyTime - window opening time,
- dueDate - window closing time,
- serviceTime - time needed for the customer to be serviced,

- demand - customer capacity
- geographical data.

Other parameters, considered in this work, include:

- distance - geographical distance between two customers,
- timeDistance - time needed to travel from one customer to another,
- windowTime - difference between dueDate and readyTime.

In the algorithm used here to solve the VRPTW, a single objective criterion is defined with the goal to minimize the total number of vehicles multiplied with 10000 and summed with traveled distance in kilometers:

$$objective = (\text{vehicle count} * 10000) + \text{distance traveled} \quad (1)$$

Using such objective criterion primary focuses on generating solution with minimal vehicles number, and in case of equal vehicles number than criterion considers traveled distance. Generally, a smaller vehicle number implies a smaller travel distance. Nevertheless, the proposed method of heuristics generation may be used with any conceivable performance objective.

A. Related Work

A lot of work has been invested in creating efficient solvers for the vehicle routing problem. A constraint decomposition approach, where a possible solution is optimized for each of the constraints in turn and then combined, was proposed in [10], [3]. The problem space is decomposed to spatial, time and vehicle dimensions. As for the optimization techniques inside the VRP domain [11], various algorithms are used. A simulated-annealing like local search was proposed in [12]. In [13] a tabu search optimization is used. The naive ejection chain method for local neighborhood searching is proposed in [14] with a high potential for solution diversification and total vehicle number reduction. Models for evolutionary methods were presented in [15], [16]. Ant-colony optimization was used in [17]. Proposed optimization approach is based on a parallel local search algorithm once the solution space is decomposed into small enough instances. A decomposition technique which reduces the number of vehicles and can also be executed in parallel was described in [5]; this approach is used in this paper, but it requires a suitable initial solution to be constructed beforehand.

Many solving procedures rely on a reasonably good initial solution, which is then improved with metaheuristics, local search algorithms, decomposition techniques and combinations of those. The problem that researches face is the choice of an appropriate method of creation of initial solutions (usually a simple greedy heuristic), which is highly dependent on the given performance objective. To the best of our knowledge, no previous work has employed genetic programming for generation of greedy algorithms that would be used to build the initial solution. This approach allows automatic creation of different greedy heuristics that may be tailored to specific objectives and classes of the problem.

III. Initial VRPTW Solutions with Genetic Programming

A. Initial solution creation using a heuristic function

The VRPTW solving algorithm used in this paper is divided in two major parts. Firstly, an initial solution is generated using a fast greedy algorithm. Then the number of vehicles is reduced using parallel customer insertion, which is described in greater detail in [5]. One of the most important advantages of the implemented parallel method is easy patching of subresults into a global result which fully meets the defined constraints.

To create an initial solution, a greedy algorithm is used that creates a list of customers for each vehicle. The input parameters are the initial customer, the vehicle capacity and a list of unvisited customers (that still need to be visited). The pseudocode of this phase of the algorithm follows:

Algorithm 1 Create initial solution

```
while unvisitedCustomer exists do
    start newRoute;
    set newRoute.first(home depot);
    while true do
        posibleCustomers <- unvisited valid customers;
        if no valid customers then
            break;
        end if
        nextCustomer <- select one posibleCustomer with minimal OBJ value;
        newRoute.add(nextCustomer);
    end while
    set newRoute.last(home depot);
    add newRoute to result[route];
end while
```

In this phase, a list of customers to visit is built by taking into consideration the criterion expressed with the OBJ value. Here the greedy heuristic is used which provides the OBJ value for every combination of current state and customer. The heuristic is in fact reduced to a function that uses customer information (such as distance, due date etc.) to identify the next customer - the one with the lowest function value. In our previous work [5] we experimented with several functions, e.g.:

- $fun1 = distance$,
- $fun2 = readyTime$,
- $fun3 = x * distance + y * waitTime + z * (dueTime - visitedTime)$.

In this case, we use genetic programming to evolve a suitable function. The 'suitable' function may be evolved for a single test case (single set of customers) or the same function may be evolved and then used with several new (unseen) test cases - both approaches are investigated and presented in the results.

B. Genetic programming

Genetic programming [6] is an optimization and machine learning technique that uses evolutionary concept to automatically discover symbolic procedures (functions, programs) to the problem at hand. The main idea behind GP is that the solution to the problem may be represented as a (computer) program, in most applications in the form of a tree (which allows mapping to any procedural language). The elements of the programs (tree nodes) must be predefined by the user and must be sufficient to describe the solution to the problem (e.g. mathematical and logical functions, variables, actions such as move forward, turn left etc). The algorithm randomly generates functions (potential solutions) and evaluates each function on a predefined set of test cases (e.g. how well does the function describes the data). Each potential solution thus receives its quality estimate - the fitness value - which is then used in the selection process.

The selection process imitates natural evolution where weaker individuals (solutions) are eliminated, and better individuals survive. Additionally, better individuals also participate in recombination, where two (or more) individuals are combined to form a new solution. The algorithm also incorporates a mutation mechanism, where a single individual is subject to a change, with a relatively small probability. The process continues, building new generations from old ones, until a suitable termination criterion is reached. These criteria usually include finding a solution of the desired quality or running the algorithm for a predefined amount of time. The examples of human-competitive results of genetic programming may be found in [18].

C. Creating the solution with GP generated heuristics

In this paper the genetic programming is used to automatically create a heuristic function used in initial solution creation. The function is represented as a tree, where inner nodes are operators and leaves are variables. The operator and variable set must be manually defined; the operators used here consist of arithmetic functions +, - and *, with subsequent experiments with a square root function. The variables used are the same values that represent a single VRPTW customer, denoted with single letters for GP readability:

- distance to the next customer (denoted as 'a'),
- readyTime ('b'),
- dueDate ('c'),
- demand ('d'),
- visitedTime - time when services start for customer ('e') and
- waitTime ('f').

This way the GP is able to create any of the manually created greedy heuristics, and possibly find a few better ones. GP creates a set of functions (individuals) in each generation, and each of those must be evaluated. This is performed in the following manner:

```
while there are individuals to evaluate do
    build initial solution using the current individual;
    decompose initial solution to obtain the final solution;
    assign the final solution quality as the current individual's fitness;
end while
```

After the whole generation is evaluated, genetic operators (crossover, mutation and selection) are performed, thus producing the next generation of individuals (new candidate functions). The whole process is repeated until a predefined stopping criterion is met.

D. Decomposition

Once an initial solution is created, the result decomposition is started which tries to reduce the total vehicle number. Each result can be divided into independent vehicles, where each vehicle has a list of customers that have to be visited. The only way to reduce the number of vehicles is to reallocate customers from a specific vehicle and thus remove that vehicle from the solution. The reduction algorithm iterates over all vehicles; for each vehicle all of its customers are assigned to other vehicles. Other vehicles can accept a customer only if the capacity and time constraints have been met. In case other vehicles can accept all these customers, the total number of vehicles is reduced by one and we say that reduction has succeeded. The process is repeated until no more reductions can be made. Specific details for this approach can be found in [5].

IV. Experimental Setup and Results

A. Implementation environment

Implementing the system which would be able to create candidate functions using GP and validate those functions required the use of two frameworks. For greedy functions evolution the Evolutionary computation framework (ECF,[19]) was used. For evaluating greedy functions the VRPTW framework (developed by authors) was used. Communication between ECF and VRPTW consists only of providing a string formatted greedy function, and after that the VRPTW would return a value which is the performance measure for generated function.

B. Experimental Setup

The experiments were conducted on Gehring and Homberger benchmark set [20] containing problems with 1000 customers. Benchmark set is divided into 6 groups (C1, C2, R1, R2, RC1, RC2). Groups named Cx contain problems in which geographical data is clustered, whereas those named Rx have randomly generated geographical data and RCx contain problems with a combination of both. Problems in groups x1 have a short scheduling horizon and allow only a small number of customers per route. In contrast, groups x2 have a long scheduling horizon and the number of customers per route is significantly bigger. For all problems, the travel time and distance is equal to the corresponding euclidean distance [21]. All problems have a central depot, as well as capacity and time window constraints.

Starting parameters for GP include mutation rate of 0.3, population of 30 individuals and stopping criterion as 20 generations without improvement. For greedy function elements we used operators +,- and *, tree depth is varied from 2 to

4, and variables are distance (a), readyTime (b), dueDate (c), demand (d), visitedTime (e) and waitTime (f).

In the first experiment all 60 problem instances were combined in one set of test cases. In other words, we try to find a single greedy function that would be used in initial solution creation for every problem instance. The result of combined instances is defined as sum of results from each instance. Best known result for all combined instances is 3418 vehicles [22], whereas the evaluation of the described method provided the result of 3763 vehicles (with the heuristic function $a + c + d + 2e - f$). In comparison, a hand-made heuristic function used in the previous work [5] achieved the result of 3781 vehicles.

Although slightly better than a hand-made heuristic, this result suggests that it is very difficult to find a single function that would yield good results on a larger set of problem instances. This may be attributed to differing characteristics of the problem set, as well as to a very large solution space, which requires much computation effort to find a good solution. Better results should be obtainable if the set of problem instances is divided into smaller groups, containing only a few (or even one) instances.

In the next phase, therefore, each of 6 groups from the benchmark set is divided into a train set (containing 5 instances for each group, with indexes 1,3,5,6,8) and a test set (containing the other 5 instances for each group, indexed 2,4,7,9,10). For each training group a separate greedy function is evolved with GP. Table I reports results generated using the evolved greedy functions on the corresponding test set. The best known total number vehicles for the 6 test sets is 1693, and with GP evolved functions the result was 1822, which is a deviation of 7.6%. Using training and test sets significantly depends on the benchmark set, and in this case in specific groups of problem instances.

Finally, in the next experiment a separate heuristic function is evolved for each of 60 instances. In this case, with maximum specialization, the obtained vehicles sum from all instances is equal to 3611, which is significantly better than a hand-made solution of 3781, and corresponds to a deviation of 5.6% from the best known solution. This indicates that the best approach is to find a function that would be used with a single test case; since this is rather time consuming, the influence of GP parameters should be investigated to reduce the required computation time.

C. Parameters of GP

After testing on combined multiple instances, our focus changed to single instances in order to tune up parameters. Among the most important ones are the stopping condition and the population size, which are tested below. Three random problem instances (C1_10_5, R2_10_5, RC1_10_5) were chosen for further testing.

To estimate the required computational effort to reach an acceptable solution quality, we try to discover an appropriate termination condition for the GP evolution, without impairing the quality of the final result. A practical termination condition usually includes stopping the evolution after no improvement has been made in a predefined number of generations, or fitness evaluations. The termination criterion was initially fixed to 50 generations without improvement. GP was then applied to the above problem instances and the evolution is repeated 50 times. For these 50 runs, Fig. 1 represents the number of runs in which the best result occured at a certain generation range. It can be seen that the majority of occurrences of the best solutions are within the first 30 generations; in the following experiments the termination condition was therefore set to 30 generations without improvement.

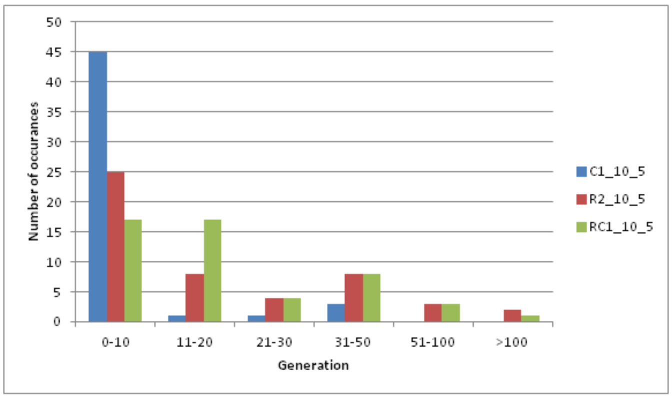

Figure 1. Influence of termination criterion

The second investigated parameter is the adeqaute population size; tables II, III and IV show minimum, average and maximum results for 3 different population sizes (10, 25 and 50 individuals) tested 50 runs for each value. In addition to the objective values, we also report the percentage of runs in which the corresponding minimum value was found (in brackets). The results are presented using the defined goal function (1). The results suggest that using 25 individuals provides a good enough average, so it would be preferable to use population size of at least 25 individuals for Gehring and Homberger test set.

Table II
POPULATION TESTS - C1_10_5

	10 (individuals)	25 (individuals)	50 (individuals)
min	1000050 (10%)	1000050 (20%)	1000050 (38%)
avg	1001859	1000058	1000056
max	1050050	1000060	1000060

Table III
POPULATION TESTS - R2_10_5

	10 (individuals)	25 (individuals)	50 (individuals)
min	200085 (2%)	200082 (2%)	190091 (2%)
avg	226673	221875	216279
max	240080	230087	230067

V. CONCLUSIONS AND FUTURE WORK

This paper presents an application of genetic programming for generation of heuristic functions that guide the creation of

Table I
VEHICLES COUNT FOR 6 TRAINING AND 6 TEST SETS

Group	Train (vehicles)	Best known (vehicles)	Test (vehicles)	Best known (vehicles)	Greedy function
C1	497	482	480	459	2a + c + 7e
C2	159	145	165	144	a * a + (2f + a) * f +e + c - d
R1	492	464	479	455	c + a
R2	104	95	105	95	3e - b - f + d
RC1	486	450	485	450	[(2a + c * c) * e - d + a + e] * c
RC2	114	92	108	90	c + e + a

Table IV
POPULATION TESTS - RC1_10_5

	10 (individuals)	25 (individuals)	50 (individuals)
min	940064 (2%)	930068 (4%)	930065 (2%)
avg	955878	948072	944068
max	990093	960091	960063

initial solutions for the vehicle routing problem. Although the paper presents only preliminary results, this approach shows high potential. Simple and fast greedy functions obtained by GP provide acceptable quality of initial results for further refinement. The presented method can be used as a basis for creating heuristic functions for various performance objectives in the VRP domain, as well as for a diverse set of optimization problems. Moreover, heuristic functions that are evolved beforehand can be used in real-time VRPTW scenarios, where changes in problem parameters may occur unexpectedly (e.g. a time window is changed for a customer, or an additional customer appears). In such cases, the existing search-based algorithms may not be practical since they require a large amount of processing.

Future work will primarily focus on definition and evaluation of additional operators and variables for greedy function creation. Experimenting with different variables could provide significant improvements. Additionally, the quality of the final solution is greatly influenced with the algorithm used in the second phase, where some other metaheuristic can also be applied. Finally, the proposed approach will be suited and evalauted to meet real world problem demands.

REFERENCES

[1] T. Caric, "Improving of transport organization using heuristics methods," *(Croatian) Ph. D thesis, University of Zagreb, Croatia*, 2006.

[2] J. K. Lenstra and A. H. G. R. Kan, "Complexity of vehicle routing and scheduling problems," *Networks*, vol. 11, no. 2, pp. 221–227, Jun. 1981. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1002/net.3230110211/abstract

[3] R. Bent and P. Hentenryck, "A two-stage hybrid local search for the vehicle routing problem with time windows," *Transportation Science*, vol. 38, no. 4, pp. 515–530, 2004.

[4] ——, "A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows," *Computers & Operations Research*, vol. 33, no. 4, pp. 875–893, 2006.

[5] M. Gulic, D. Lucanin, and N. Skorin-Kapov, "A two-phase vehicle based decomposition algorithm for large-scale capacitated vehicle routing with time windows," *MIPRO, 2012 Proceedings of the 35th Internation Convetion*, pp. 1104–1108.

[6] J. R. Koza, "Genetic programming - on the programming of computers by means of natural selection."

[7] D. Jakobovic and K. Marasovic, "Evolving priority scheduling heuristics with genetic programming," *Applied Soft Computing*, vol. 12, no. 9, pp. 2781–2789, Sep. 2012. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1568494612001780

[8] D. Jakobovic, L. J. c, and L. Budin, "Genetic programming heuristics for multiple machine scheduling," *Lecture Notes in Computer Science*, vol. 4445, pp. 321–330, 2007.

[9] M. Solomon, "Algorithms for the vehicle routing and scheduling problems with time window constraints," *Operations research*, pp. 254–265, 1987.

[10] R. Bent and P. Hentenryck, "Spatial, temporal, and hybrid decompositions for large-scale vehicle routing with time windows," *Principles and Practice of Constraint Programming-CP 2010*, pp. 99–113, 2011.

[11] P. Toth and D. Vigo, *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics ans Applications, SIAM, Philadelphia, 2002.

[12] H. Li and A. Lim, "Local search with annealing-like restarts to solve the VRPTW," *European journal of operational research*, vol. 150, no. 1, pp. 115–127, 2003.

[13] E. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J. Potvin, "A tabu search heuristic for the vehicle routing problem with soft time windows," *Transportation science*, vol. 31, no. 2, pp. 170–186, 1997.

[14] L. Rousseau, M. Gendreau, and G. Pesant, "Using constraint-based operators to solve the vehicle routing problem with time windows," *Journal of Heuristics*, vol. 8, no. 1, pp. 43–58, 2002.

[15] G. Schrimpf, J. Schneider, H. Stamm-Wilbrandt, and G. Dueck, "Record breaking optimization results using the ruin and recreate principle," *Journal of Computational Physics*, vol. 159, no. 2, pp. 139–171, 2000.

[16] D. Mester and O. BrÃď ysy, "Active guided evolution strategies for large-scale vehicle routing problems with time windows," *Computers & Operations Research*, vol. 32, no. 6, pp. 1593–1614, 2005.

[17] L. Gambardella, E. Taillard, and G. Agazzi, "Macs-vrptw: A multiple colony system for vehicle routing problems with time windows," in *New ideas in optimization*, 1999.

[18] J. R. Koza, "Human-competitive results produced by genetic programming," *Genetic Programming and Evolvable Machines*, vol. 11, pp. 251–284, September 2010. [Online]. Available: http://dx.doi.org/10.1007/s10710-010-9112-3

[19] *ECF - Evolutionary Computation Framework*, 2011. [Online]. Available: http://gp.zemris.fer.hr/ecf

[20] "Extended SOLOMON's VRPTW instances," http://www.fernuni-hagen.de/WINF/touren/inhalte/probinst.htm. [Online]. Available: http://www.fernuni-hagen.de/WINF/touren/inhalte/probinst.htm

[21] M. Deza and E. Deza, *Encyclopedia of distances*. Springer Verlag, 2009.

[22] "VRPTW," http://www.sintef.no/Projectweb/TOP/Problems/VRPTW/. [Online]. Available: http://www.sintef.no/Projectweb/TOP/Problems/VRPTW/