

Automated Design of Routing Policies for the Dynamic Electric Vehicle Routing Problem with Genetic Programming

Marko Đurasević¹ ^a and Francisco Javier Gil Gala² ^b

¹University of Zagreb Faculty of Electrical Engineering and Computing, Unska 3, Zagreb, Croatia

²University of Oviedo, Gijon, Spain

Keywords: Genetic Programming, Electric Vehicle Routing Problem, Hyper-Heuristics, Routing Policy.

Abstract: The dynamic electric vehicle routing problem (EVRP) with time windows (DEVPTW) is an important combinatorial optimisation problem gaining on importance in today's world due to environmental concern and the requirement of dealing with dynamic and uncertain environments. This represents a problem when solving such problems, as standard improvement based heuristics cannot be used to solve them, since not all information about the problem is known beforehand. This provides a motivation for applying improvement based heuristics, most notably routing policies (RPs), which determine only the next decision that needs to be performed and execute it. Since these RPs do not construct the schedule in advance, they can easily react to any changes in the problem. However, RPs are difficult to design, which motivated the use of genetic programming (GP) in automatically designing such heuristics for various problems. Unfortunately, in the context of EVRP only static problems were considered. This study investigates the application of GP to automatically design new RPs for DEVPTW under different levels of dynamism. The results demonstrate that GP performs well for certain levels of dynamism, although as it increases it is more difficult to perform good decisions.


1 INTRODUCTION


The vehicle routing problem (VRP) represents an important combinatorial optimisation problem often encountered in the real world in various areas of logistic and transportation (Erdelić and Carić, 2019). When solving the VRP the main goal is to construct a set of routes for a number of vehicles that need to visit and serve a given number of customers, in a way that one or more user defined criteria are optimised. Due to the growing environmental concern and the desire to reduce the negative influence of humans on the environment, the research is shifted towards the green VRP problem variants (Moghdani et al., 2021), most commonly the electric VRP (EVRP) (Qin et al., 2021).

VRP, and by extension EVRP, are both NP-hard problems, which in most cases leads to the application of various heuristic and metaheuristic based methods for solving such problems (Erdelić and Carić, 2019). Although these methods can solve various EVRP problems efficiently, they are usually not appropriate when solving dynamic, uncertain, or large

scale problem variants (Mardešić et al., 2023). The reason for this is that they search the entire search space for the best possible solution, however, the entire information about the problem is either unavailable (as in dynamic problems) or certain information is susceptible to change (such as in uncertain problems) (Mardešić et al., 2023). Therefore, the solutions that these methods would construct would likely become infeasible, and would have to be modified according to the changing conditions in the problem.

Under such conditions, simple constructive based heuristics, usually denoted as routing policies (RPs), represent a viable alternative to standard improvement based metaheuristics (Jacobsen-Grocott et al., 2017). Instead of searching the space of all solutions, RPs construct a solution in a step-wise manner using a certain strategy. This means, each time that a certain decision needs to be made, they determine the "best" decision using some kind of a heuristic rule, for example that the vehicle that becomes free should visit the closest customer to it. By constructing the solution in such a way RPs can quickly react to changes that occur in the system, such as the arrival of new customers, or changes in their orders, since they do not construct the entire solution until the end.

^a  <https://orcid.org/0000-0001-8732-4769>

^b  <https://orcid.org/0000-0002-0606-7009>

However, such methods also have certain limitations, with one of the most important being that it is difficult to manually design high quality RPs. Due to this reason researchers have resorted to using various hyper-heuristic methods (Burke et al., 2013), most notably genetic programming (GP) (Poli et al., 2008) to aid them in the design of new RPs for various VRP problem variants that they were faced with. For example, GP was used to design RPs for the VRP variant with time windows for both static and dynamic environments (Jacobsen-Grocott et al., 2017; Jakobović et al., 2023). Furthermore, GP was also successfully used to generate RPs for VRP with zone based pricing (Afsar et al., 2021), a variant in which not all customers need to be served and in which it is also required to determine the prices for different customer zones (Gil-Gala et al., 2023). Regarding the EVRP variant, it was investigated in a short study where different scenarios were considered, although all were static (Gil-Gala et al., 2022).

As we can see, the literature dealing with automated design of RPs with VRPs and especially on the EVRP variant is still scarce, although the problem is of practical relevance. Therefore, in this paper we investigate the ability of GP to generate RPs for the dynamic EVRP with time windows (DEVPTW) problem variant in which the requests of the customers are not known in advance, but rather become available over time. We consider problems with several levels of dynamism and different properties in order to analyse how the RPs generated with GP perform under different conditions. The obtained results demonstrate that automatically generated RPs can perform well also on dynamic problems, although on certain problem types their performance significantly deteriorates as the level of dynamism increases.

2 DYNAMIC EVRP

The DEVPTW problem is usually modelled as a fully connected graph in which the nodes represent different types of locations, with edges representing the connections (for example roads) between those locations (Schneider et al., 2014; Lin et al., 2016). Each node has 2 coordinates that denote its position in a 2-dimensional Euclidean space, whereas each edge has a certain weight that denotes the Euclidean distance between those nodes. In DEVPTW, these nodes represent either a customer, charging station, or depot. Each customer i has a certain demand dem_i that needs to be served by a vehicle, the service time s_i that specifies the time required to serve the customer, and time window during which the service of the customer

needs to begin, which is defined with the release time r_i and deadline d_i of the customer. This variant of the DEVPTW is denoted as DEVPTW with time windows (DEVPTWTW) in the literature (Schneider et al., 2014).

In order to serve the demands of the customers, a fleet of electric vehicles is available, all with a given cargo capacity, battery level, and speed. At the start, all vehicles are located at the depot and they leave the depot with full cargo and battery capacity. In the considered problem it is presumed that there is only a single depot. Each vehicle has a cargo capacity, which is reduced by the demand of the customer once the vehicle finishes servicing him. Furthermore, each vehicle also has a battery level, which is reduced when traversing an edge. The reduction of the battery level depends linearly on the distance that is traversed by the vehicle from the source to the destination. When the vehicle does not have enough cargo capacity to serve a customer, it returns to the depot and finishes the route, since in the considered problem the cargo cannot be reloaded. The energy level of the vehicle can be recharged by visiting one of the available charging stations. When a vehicle reaches a charging station its battery level is fully recharged, with the recharging time depending on the amount of energy being recharged. The charging station have no limit on the number of vehicles or the capacity that can be recharged. In the problem considered in this study it is presumed that the fleet of vehicles is homogeneous, meaning all vehicles have the same properties.

There are various objectives that can be considered when solving the DEVPTWTW. However, the most important objectives is the minimisation of the number of vehicles used to service all the customers (Lin et al., 2016). The reason why this objective is the most important one is since it usually has the largest influence on the cost (due to vehicle maintenance and driver cost). Therefore, this objective is usually optimised as the primary one, after which a secondary objective is optimised. Since in the considered problem it is required to service customers during their time windows, the secondary objective will be defined as the total lateness L , which determines the total time that all vehicles were late when servicing customers on their route. The total objective is finally defined as a weighted sum of these two objectives: $Obj(x) = c * V(x) + L(x)$, where $Obj(x)$ denotes the total objective value for a solution x , V denotes the total number of vehicles used in solution x , c denotes a scaling constant, and L denotes the total lateness of vehicles in solution x . In this study, c is set to 10^9 so that the algorithm primarily focuses on the optimisation of the number of vehicles.

In this study we investigate the dynamic variant

of the aforementioned problem. This dynamic nature of the problem is manifested in the way that the information about the customers becomes available as the problem is being solved. Namely, not all customer information is available from the start, rather only a portion of the problem information is available, which is defined as the look-ahead window. The look-ahead window is defined as an interval $[0, ct + (1 - dl) * max_r]$, where ct represents the current time of the system, max_r the maximum release time of a customer, and dl the level of dynamism. For all the customers whose release times fall into this interval, their information is available to the routing algorithm and they can be considered for routing. The information about all other customers will still be unknown until their release time does not fall into the look-ahead window. We see that as the current time of the system advances, the upper interval will be increasing, thus more customers will fall within the look-ahead window and their information will become known, until finally the information about all the customers becomes known. The dynamic level parameter dl is used to determine the level of dynamism in the system. If it is equal to 0, this means that the look-ahead window will cover the entire horizon and the information about all the jobs will be known from the start, which amounts to the static variant of the problem. On the other hand, if the parameter value is equal to 1, only the customers with a release time until the current moment in time will be known.

3 DESIGNING RPs WITH GP

Genetic programming (GP) is an evolutionary computation method with a similar structure as genetic algorithms (GAs) (Poli et al., 2008). This means that the algorithm starts with a random set of solutions which are iteratively improved using various genetic operators such as selection, crossover, and mutation. The main difference between GP and GAs is the solution representation, since in GP the solutions are represented as expression trees. This representation enables GP to be used as hyper-heuristic, i.e., a method to develop new heuristics for solving a combinatorial optimisation problem (Burke et al., 2013). Until now, GP has been successfully utilised as a hyper-heuristic method for various combinatorial problems, such as scheduling (Branke et al., 2016), container relocation (Đurasević et al., 2024), bin packing (Burke et al., 2012), and travelling salesman (Duflo et al., 2019). Although there are other methods that can be used as hyper-heuristics, such as neural networks (Branke et al., 2014) or methods similar to GP (Planinic et al.,

2022), GP still remained dominant.

The RPs designed with GP can be divided into two components, the routing scheme (RS) and the priority function (PF) (Gil-Gala et al., 2022). The RS defines the outline of the RP, which determines when a routing decision needs to be performed and ensures that the constructed solution is feasible. The RS used in this study to construct the solution to the problem is outlined in Algorithm 1. The RS starts with a set of vehicles, and then it iteratively selects the first available vehicle (at the start all vehicles are available at the same time) and determines the next location that it should visit. This location is determined by the PF, which is used to rank all the remaining available and unserved customers. The best ranked customer is then selected for service by the current vehicle. If the vehicle does not have enough cargo capacity to serve the demand of the selected customer, then it returns to the depot. Otherwise, it visits the selected customer and serves him. If the vehicle arrives before the release time of customer, then it needs to wait until the customer becomes available. If the vehicle arrives during the time window of the customer, then the customer can immediately be served. On the other hand, if the vehicle arrives after the due date of the customer, the customer can still be served, but a certain penalty in the form of lateness is incurred. During the service of customers it is possible that the vehicle does not have enough energy to visit the selected customer. In this case the vehicle will first visit one or more charging stations in order to refill its energy. The charging stations it will visit will be selected so that the path between the source and destination is the shortest possible. This is possible since the number of charging stations is not large and therefore the best path between the source and destination can be determined.

Algorithm 1: Outline of the routing scheme.

```

1: while not all customers are served do
2:    $v \leftarrow$  first available vehicle.
3:   for customer  $c_j$  out of all the considered customers  $C$  do
4:     Calculate the priority  $\pi_j$ 
5:   Determine the customer  $k$  with the smallest priority value  $\pi_k$ 
6:   if Vehicle  $v$  has not enough capacity to serve customer  $c_k$ , or no customer  $k$  was selected then
7:     Return vehicle  $v$  to the depot.
8:   else
9:     Visit customer  $c_k$  by vehicle  $v$ .
```

As previously outlined, the PF is used to determine the customer that will be scheduled next. The PF

represents a mathematical expression in which variables represent different information about the current state of the problem. Since the PF is just a mathematical expression that can easily be represented as an expression tree, this makes GP suitable for developing new PFs. However, in order to be able to do so, it is required to define the building blocks used by GP to construct such a PF. These building blocks represent the functions and variables that will be used in the PF. The functions used by GP are the addition, subtraction, multiplication, protected division (returns 1 in case division by 0 occurs), maximum, and minimum operators, all of which are binary. The variables, also denoted as terminal nodes since they represent the leaf nodes of the expression tree, used to construct the PFs are outlined in Table 1. These terminals represent various kinds of information about the system, ranging from basic system information, such as the demand of the customer, or the remaining cargo capacity of the vehicle. However, this set also includes certain nodes that represent some extended information about the problem, such as the average distance of a customer to other customers, or the remaining time until the vehicle would become late for serving a customer.

With the given set of function and terminal nodes, GP can construct the PF required to solve the problem under consideration. It should be stressed out that the generation process of a new RP is a computationally expensive task. The reason for this is that the PFs need to be evaluated on a set of problem instances to ensure that GP has obtained a PF that performs well across different problems, and not just on a single problem. However, once the PF has been evolved, it can be used in conjunction with the RS as a RP that constructs schedules for new and unseen DEVRPTW problems in time that is negligible compared to improvement based heuristics. This characteristic of the generated routing policies represents an important advantage over improvement based methods.

4 EXPERIMENTAL ANALYSIS

4.1 Experimental Setup

To generate the PFs that will be used by RPs we use a steady state tournament GP algorithm. This algorithm starts with a randomly initialised population of 200 individuals, where each individual is initialised using the ramped half-and-half method. The size of the individuals is limited to a maximum depth of 5. In each iteration of the algorithm, the 3-tournament selection procedure is performed, meaning that 3 individuals are randomly selected from the population,

Table 1: The set of terminal nodes used by GP to construct PFs.

Parameter	Description
d_j	Distance from the vehicle's current location to customer j
r_i	Ready time of customer j
dd_j	Due date of customer j
dem_j	Demand of customer j
t	Current time
s_j	Slack (time remaining until being late) of vehicle for customer j
ddc	Distance of customer j to the depot
dc	Distance of customer j to the nearest charging station
du	Distance of customer i to the nearest unserved customer
vc	Remaining vehicle cargo capacity
bc	Remaining vehicle battery capacity
d_{avg}	Average distance of customer j to all other locations
dc_{avg}	Average distance to 10 closest destinations.
nn	Number of locations within a given radius (10% of the maximum distance between any customers)

with the better two being used in crossover to generate a new individual. The new individual is then mutated with a probability of 0.3 and inserted into the population by replacing the worst individual in the tournament. For crossover, the uniform, size fair, context preserving, subtree and one point operators are used, whereas for mutation the hoist, subtree, permutation, node complement, node replacement, and shrink mutation operators (Poli et al., 2008). Since several operators are defined for crossover and mutation one of them is randomly selected each time either crossover or mutation need to be performed. The algorithm terminates when 50 000 iterations were executed. All the parameter values were selected based on a preliminary experimental analysis.

To evaluate the fitness of each individual, or rather PF, it is coupled with the RS outlined in Algorithm 1 and used to solve a set of predefined problem instances. The problem instances that are used for this purpose are those proposed in (Schneider et al., 2014). This dataset consists of 56 instances that contain 100 customers that need to be served, 21 charging stations, and a single depot. However, regarding the way in which they were generated, the instances can be divided into several types. Regarding the way in which the customers are distributed, the instances can be divided into random (R), cluster (C), or ran-

dom cluster (RC). In the random instances the position of customers are generated completely randomly within a given interval. On the other hand, in the cluster instances the position of customers are generated randomly around certain clusters, meaning that customers are organised in certain groups. Finally, the random cluster instances represent a combination of both previous types. The second distinction between the problem instances comes from the way in which the time windows of customers were generated. In this case the problem instances can be grouped into instances with tight (T) time windows or loose time windows (L). In instances with tight time windows the size of the time window will be small, meaning it will be more difficult for vehicles to arrive in a timely manner at the customer. On the other hand, in problem instances with loose time windows there is more time available to serve each customer, thus making it easier to timely serve more customers. The original dataset defined in (Schneider et al., 2014) is used as the test set, meaning it will be used to evaluate the generalisation ability of the generated PFs after they have been evolved by GP. However, GP also requires a dataset that is used to evaluate the quality of solutions during the evolution process. Therefore, another dataset has been generated based on the description from (Schneider et al., 2014) and used as a training set, i.e., a set used by GP during the generation process of PFs. This set can be obtained from <https://github.com/nfridFER/EVRP-data>.

The aforementioned instances were designed with static scheduling conditions in mind. However, they can easily be considered as dynamic instances in a way described in Section 2. This means that a dynamic level is defined that specifies how many customers in the future are known at the current moment in time. To test the performance of the algorithm in different situations, scenarios with different levels of dynamism were used, which include levels of 0, 0.1, 0.3, 0.5, and 0.7, where 0 represents the static case.

Finally, to obtain an objective measure of the performance of the algorithm on the different scenarios, each experiment was repeated 30 times, and the best individual on the training set from the final population was stored. This resulted in 30 PFs being obtained for each scenario, out of which the average value was calculated and is outlined in the results table.

4.2 Experimental Results

Table 2 outlines the results obtained by the automatically designed RPs. They are evolved by GP for different dynamic levels and problem sets, with the best result for each dynamic level being outlined in bold.

As expected, the best results are obtained in scenarios with the lowest levels of dynamism. For two problem types we can see that the best result was obtained in the case of 0.1. Although surprising, it seems that in these scenarios the knowledge about the customers that are available at the end does not influence the algorithm much. The worst results are obtained when the level of dynamism reaches the largest levels, since in those cases the least amount of information is available.

The results are also illustrated in Figures 1, 2 and 3 as box plots, to gain a better notion on the distribution of the results across the 30 executions of the algorithm. Figure 1 represents the results for problems in which the customers are randomly distributed. It is interesting to note how for the tight due dates for the dynamism levels of 0.5 and 0.7 the results greatly deteriorate. And although sometimes the algorithm can find a good rule, in most cases the evolved rules achieve a poor result. The situation is better for the case with loose time windows, although for level of 0.5 the algorithm again performed poorly. However, this might simply be a unlucky case, as for a larger level of dynamism this behaviour is not observed.

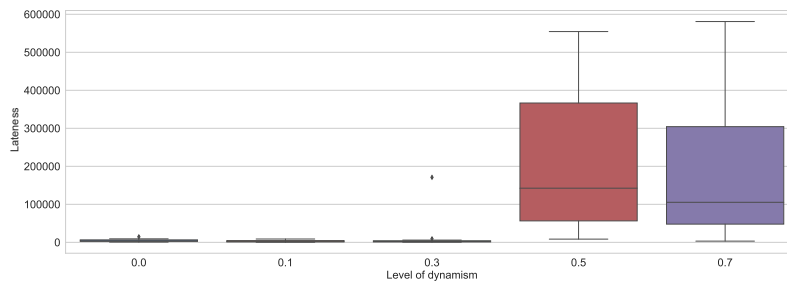
The results for the case when the customers are distributed randomly are shown in Figure 2. Here we can observe a completely different situation, in which for the instances with tight time windows we have similar results for all levels of dynamic levels, except for the static case. Therefore, it seems that here not knowing the complete problem from the start can be quite penalising for the algorithm. On the other hand, in the case of loose time windows, we see that there are almost no difference between the rules that were generated for the different levels of dynamism.

Results for random clustered instances are outlined in Figure 3. In this case the tight time windows the algorithm are more sensitive to the level of dynamism, although the case for 0.1 could again indicate a special case, as for larger levels we see that the algorithm performs again better. For the case of loose time windows there is very little difference between the RPs generated for different level of dynamism.

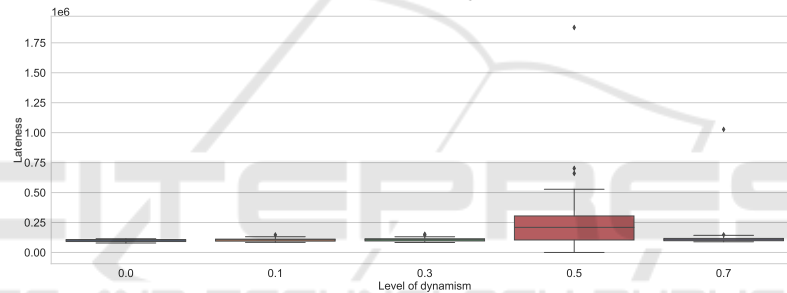
Based on the previously described results, we can draw certain conclusions on the performance of GP when generating RPs for dynamic DEVRPTW problems. Regarding the level of dynamism, we see that almost consistently the generated RPs perform well for smaller levels of dynamism, like 0.1 and 0.3, with no difference between these results and those obtained by the RPs that were generated for the static case. This suggests that knowing the information about customers that have their time windows far in the future does not play a vital role in most of the

Table 2: Results obtained by automatically designed routing rules for different dynamic levels and problem types.

	0.0	0.1	0.3	0.5	0.7
Cluster 1	4474	3139	8278	206258	183120
Cluster 2	99865	105233	107392	235837	105710
Random 1	93246	337253	286211	360598	272148
Random 2	875653	745569	793060	844117	760951
Random cluster 1	107385	169637	109494	370116	188407
Random cluster 2	863325	868767	870696	852218	876266



(a) Results for instances with tight time windows.



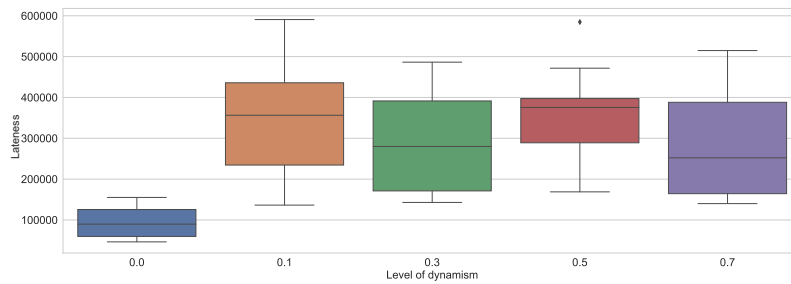
(b) Results for instances with loose time windows.

Figure 1: Results for the problem instances with clustered customer distribution.

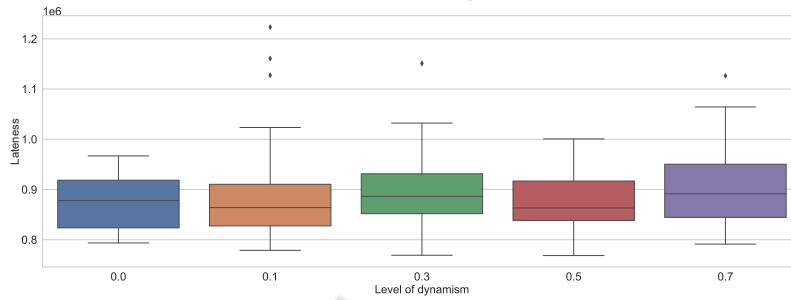
tested scenarios and that the RPs can perform as well as in the case then all customer information is available from the start. However, for the larger levels of dynamism we see that the behaviour of the generated RPs depends on the problem instances that are considered. In the case of problem instances with loose due dates the increase in the level of dynamism does not lead to a significant deterioration of the performance of the generated RPs. On the other hand, in the cases where the due dates are tight we observe that the performance of the automatically generated PRs deteriorates significantly as the level of dynamism increases to 0.5 and 0.7. The reason for this is that in the case of problem instances with loose time windows the vehicles have a lot of more flexibility to serve a customer. For example, even if the vehicle is quite far away it would be possible to reach the customer in time, as the time windows are wide. On the other hand, when customers have tight time windows they can be served only during a very limited time, thus if all the vehicles are far away it can be difficult to serve the customer

timely. In this case it is important to plan the route while considering future requests, however, not all are available, and therefore the RP has a myopic view on the problem and cannot follow a long term strategy.

The approach proposed in this study has certain limitations. Although it is applicable in dynamic environments, it suffers from the problem that without the information about future requests the solutions will be of poor quality as it cannot develop a long term strategy. The reason for this is the way in which the dynamic nature of the problem was modelled, since no information about future customers is available. Such a model can be considered pessimistic, as it provides the least possible information about the problem. By relaxing this model it would be possible to obtain better solutions, as more information could be integrated into terminal nodes, and used by GP. For example, the positions of the customers could be known, only their demands and time windows could be modelled with uncertainty.

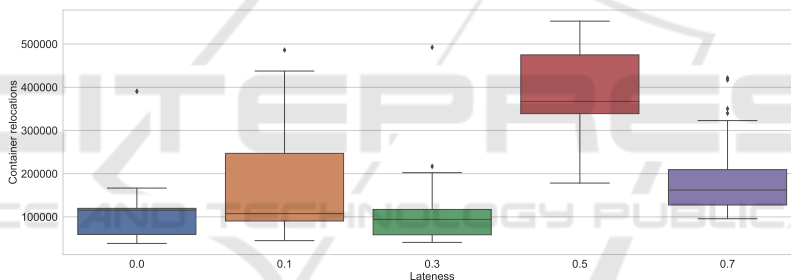


(a) Results for instances with tight time windows.

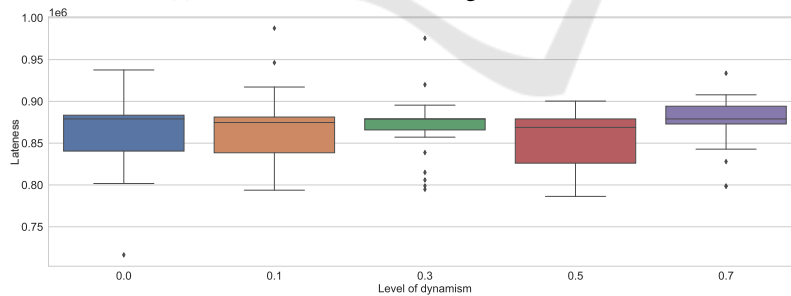


(b) Results for instances with loose time windows.

Figure 2: Results for the problem instances with random customer distribution.



(a) Results for instances with tight time windows.



(b) Results for instances with loose time windows.

Figure 3: Results for the problem instances with random clustered customer distribution.

5 CONCLUSION

This study investigated the application of GP to automatically generate RPs for DEVRPTW. The goal was to analyse how GP performs for various levels of dynamism. The algorithm was examined on sev-

eral scenarios and results demonstrate that GP can generate efficient RPs for lower values of dynamism. As the level of dynamism increases to larger levels, GP struggles to generate RPs that perform well. This shows the limitation of the approach as the RPs do not have enough information to perform good decisions

and motivates the investigation of problems in which at least some information about customers is known.

This study dealt only with one form of dynamic behaviour, but other forms of dynamic changes like the weights of the edges, or various uncertainties about the travel times or customer demands, could also be considered. Furthermore, in this paper we considered what we could call a pessimistic variant of the problem, in which nothing was known about future customers. However, it is very likely that some information is known and could be used by the heuristics to better perform their decisions. Such information could be modelled through various terminal nodes that would be used by GP when designing new RPs. Finally, subsequent studies should focus on problems that model more real world characteristics, such as nonlinear recharging functions or partial recharging.

ACKNOWLEDGEMENTS

This research has been supported by the European Union - NextGenerationEU under the grant NPOO.C3.2.R2-II.06.0110. and the Spanish Government under projects MCINN-23-PID2022 and TED2021-131938B-I00.

REFERENCES

- Afsar, H. M., Afsar, S., and Palacios, J. J. (2021). Vehicle routing problem with zone-based pricing. *Transportation Research Part E: Logistics and Transportation Review*, 152:102383.
- Branke, J., Hildebrandt, T., and Scholz-Reiter, B. (2014). Hyper-heuristic evolution of dispatching rules: A comparison of rule representations. *Evolutionary computation*, 23.
- Branke, J., Nguyen, S., Pickardt, C. W., and Zhang, M. (2016). Automated design of production scheduling heuristics: A review. *IEEE Transactions on Evolutionary Computation*, 20(1):110–124.
- Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Qu, R. (2013). Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724.
- Burke, E. K., Hyde, M. R., Kendall, G., and Woodward, J. (2012). Automating the packing heuristic design process with genetic programming. *Evolutionary Computation*, 20(1):63–89.
- Đurasević, M., Đumić, M., Corić, R., and Gil-Gala, F. J. (2024). Automated design of relocation rules for minimising energy consumption in the container relocation problem. *Expert Systems with Applications*, 237:121624.
- Duflo, G., Kieffer, E., Brust, M. R., Danoy, G., and Bouvry, P. (2019). A gp hyper-heuristic approach for generating tsp heuristics. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 521–529.
- Erdelić, T. and Carić, T. (2019). A survey on the electric vehicle routing problem: Variants and solution approaches. *Journal of Advanced Transportation*, 2019:1–48.
- Gil-Gala, F. J., Afsar, S., Durasevic, M., Palacios, J. J., and Afsar, M. (2023). Genetic programming for the vehicle routing problem with zone-based pricing. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '23*, page 1118–1126, New York, NY, USA. Association for Computing Machinery.
- Gil-Gala, F. J., Durasević, M., and Jakobović, D. (2022). Genetic programming for electric vehicle routing problem with soft time windows. In *Proceedings of the '22 Genetic and Evolutionary Computation Conference, GECCO'22*.
- Jacobsen-Grocott, J., Mei, Y., Chen, G., and Zhang, M. (2017). Evolving heuristics for dynamic vehicle routing with time windows using genetic programming. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 1948–1955.
- Jakobović, D., Đurasević, M., Brkić, K., Fosin, J., Carić, T., and Davidović, D. (2023). Evolving dispatching rules for dynamic vehicle routing with genetic programming. *Algorithms*, 16(6).
- Lin, J., Zhou, W., and Wolfson, O. (2016). Electric vehicle routing problem. *Transportation Research Procedia*, 12:508–521.
- Mardešić, N., Erdelić, T., Carić, T., and Đurasević, M. (2023). Review of stochastic dynamic vehicle routing in the evolving urban logistics environment. *Mathematics*, 12(1):28.
- Moghdani, R., Salimifard, K., Demir, E., and Benyettou, A. (2021). The green vehicle routing problem: A systematic literature review. *Journal of Cleaner Production*, 279:123691.
- Planinic, L., Backovic, H., Durasevic, M., and Jakobovic, D. (2022). A comparative study of dispatching rule representations in evolutionary algorithms for the dynamic unrelated machines environment. *IEEE Access*, 10:22886–22901.
- Poli, R., Langdon, W., and Mcphee, N. (2008). *A Field Guide to Genetic Programming*.
- Qin, H., Su, X., Ren, T., and Luo, Z. (2021). A review on the electric vehicle routing problems: Variants and algorithms. *Frontiers of Engineering Management*, 8(3):370–389.
- Schneider, M., Stenger, A., and Goeke, D. (2014). The electric vehicle-routing problem with time windows and recharging stations. *Transportation Science*, 48:500–520.