9-2020

A genetic algorithm to minimise number of vehicles in an electric vehicle routing problem

Kiian Leong Bertran QUECK
Singapore Management University, klqueck.2015@sis.smu.edu.sg

Hoong Chuin LAU
Singapore Management University, hclau@smu.edu.sg

Citation

1

A Genetic Algorithm to Minimise the Number of Vehicles in the Electric Vehicle Routing Problem

Bertran Queck and Hoong Chuin Lau(B)

Fujitsu-SMU Urban Computing and Engineering Corporate Lab, School of Information Systems, Singapore Management University, Singapore, Singapore
klqueck.2015@sis.smu.edu.sg,hclau@smu.edu.sg

Abstract. Electric Vehicles (EVs) and charging infrastructure are starting to become commonplace in major cities around the world. For logistics providers to adopt an EV fleet, there are many factors up for consideration, such as route planning for EVs with limited travel range as well as long-term planning of fleet size. In this paper, we present a genetic algorithm to perform route planning that minimises the number of vehicles required. Specifically, we discuss the challenges on the violations of constraints in the EV routing problem (EVRP) arising from applying genetic algorithm operators. To overcome the challenges, techniques specific to addressing the infeasibility of solutions are discussed. We test our genetic algorithm against EVRP benchmarks and show that it outperforms them for most problem instances on both the number of vehicles as well as total time travelled.

Keywords: Electric Vehicle Routing Problem · Genetic algorithm

Published in Computational Logistics: ICCL 2020, September 28-30, Enschede, Netherlands. LNCS 12433, pp. 200-214. DOI: 10.1007/978-3-030-59747-4_13

1 Introduction

With the rise of electric vehicles (EVs) and the charging stations' infrastructure, corporations may look to adopt an EV fleet as a green alternative. Traditional fuel-powered combustion engine vehicles are less energy-efficient than battery-powered EVs. However, one weakness of EVs is that they have a shorter driving range as compared to fuel-powered vehicles. This is largely related to the size of the battery in the EV. In [4], we see that in terms of cost-efficiency, EVs are only comparable to fuel-powered vehicles if they use a small battery.

Thus, it is important for EVs to incorporate charging station visits to recharge its battery in route planning. While charging stations are being built, they are not as ubiquitous as petrol stations. Charging of EVs is also unlike refueling traditional vehicles as it costs a considerable amount of time. Hence, there is a need to factor queuing time at charging stations as well [7]. Due to the higher upfront investment cost of acquiring an electric vehicle as compared to a fuel-powered vehicle, fleet sizing and management becomes even more important. In that regard, being able to know the minimum number of required EVs can help corporations plan their investment strategy.

This paper provides a genetic algorithm (GA) approach to solve the Electric Vehicle Routing Problem (EVRP) that minimises the number of vehicles while accounting for charging and waiting times at charging stations. The classical Vehicle Routing Problem (VRP) is a well-researched topic, with many different variants like Vehicle Routing Problem with Time Windows, etc. However, as EV adoption and research are nascent, there are opportunities to adapt or explore new methods to solve the EVRP variants. The variant and constraints of EVRP we focus on is adopted from [7], and can be defined as follows (detailed mathematical notations can be found in [7]):

1. A single depot
2. Set of customer nodes with varying deterministic demands and service time; all customers have to be served
3. Set of identical charging stations, each of which can only be used sequentially, later arrivals are to wait in queue
4. Set of identical EVs V, each starting and ending at the depot, and each having a load capacity, an upper limit on battery capacity and a battery which cannot fall to 0, and an upper limit on the time taken for a route.

The objective is first to minimise the number of vehicles used, and secondarily to minimise total travel time. More precisely, our objective function is defined as follows:

$$\min\,(K \times M) + \sum_{k \in V} t^k. \tag{1}$$

where

K $(\leq V)$ is the number of vehicles used
M is a large constant representing the penalty coefficient
t^k is the total travel time of vehicle k

As seen in [7], with the same constraints, finding optimal solutions for a 10-customer problem takes longer than 3 h. Rather, our aim is to achieve good results by a genetic algorithm. In this paper, we will discuss the challenges that arise from applying genetic algorithm operators to obtain feasible solutions for EVRP. To overcome these challenges, we discuss modifications to the selection, crossover, and mutation operators of the GA, respectively. For each technique, we explain the challenge we address as well as the reasons why the proposed approach is suitable. In addition, were sent methods to enhance the diversity of search and improvement of runtime performance.

This paper proceeds as follows. Section 2 gives a brief literature review. In Sect. 3, we provide the limitations of the current GA approach. Sections 4–6 give our detailed solution approach in terms of the modified crossover, mutation, and enhancing diversity. We discuss our experimental analysis in Sect. 7, and conclude in Sect. 8.

2 Literature

Prior research in EVRP focused on minimising electric consumption using a hybrid-GA approach with Tabu Search (TS) [3]. The authors' approach was to use GA and

TS as a local search, and a Tabu List to avoid getting stuck in a local optimum. [7] proposes to solve EVRP by a GA with popular operators, such as tournament selection, partially matched crossover, and also local search. Some papers discuss how infeasibility is handled through the use of a penalty function to chromosomes that violate constraints, while other approaches include repairing the chromosome by fixing the infeasibility [6, 11, 13]. In [2], the authors discuss a crossover for CVRP to not even create infeasible solutions in the first place. Also, using a crossover that is designed for their constraint gave them superior results as compared to using the commonly good crossover operators. When it comes to mutation operators, commonly used mutation operators like swap and insertion are used [12]. These operators work is a way where nodes are randomly selected, and they have their positions changed, without regards to their context or feasibility. In [7], a local search technique that is similar to the insertion mutation operator is used, but it only accepts the new solution if there is an improvement. On population selection and diversity management, there are a few methods. One is the diversity-controlling adaptive genetic algorithm [15], which calculates the current population diversity, and increase and decrease the rate of crossovers and mutations to achieve a targeted diversity. Other methods include a Fitness Uniform Selection Scheme, and a Fitness Uniform Deletion Scheme where population selection is based on their fitness value [5, 8]. The claim is that by selecting chromosomes of differing fitness, they are able to maintain population diversity.

3 Limitations of the Base Genetic Algorithm

A GA consists of 3 key steps: selection, crossovers as well as mutation. A solution is encoded as a chromosome and multiple chromosomes are maintained in a population, and each has a fitness value. The steps to a basic genetic algorithm are as follows.

1. Initialise the population of chromosomes and compute their fitness values.
2. The crossover between 2 good chromosomes to create a new chromosome.
3. The mutation operation is applied randomly on any chromosome.
4. Recalculate fitness value of chromosomes.
5. The selection mechanism is used to determine who makes it to the next generation.
6. Repeat from step 2 until we complete the stipulated number of generations.

Our work extends the existing GA from [7]. Having a baseline algorithm to work with, we can observe the performance of the algorithm to identify potential gaps and address them accordingly. Specifically, the challenge is to find good feasible routes that satisfy the *vehicle load capacity*, *battery levels*, and *vehicle time taken limit*. We term the three crucial constraints as the *triple constraints*. Intuitively, for a route where any of these constraints is binding, we know that it is unable to include another customer node at any position without additional modifications (such as local search). Any algorithm not only has to account for travel time to the charging station but also the charging time and waiting time (if there are other vehicles in queue). A random mutation can easily cause the resulting solution to become infeasible. In the following, we will describe the challenges and proposed techniques to deal with the triple constraints.

4 Modified Crossover

The triple constraints tend to create an offspring solution that is infeasible. In this section, we present our modified crossover operator which improves the one proposed in [7].

Table 1 illustrates the used chromosome representation. Depot denotes the start and end of a single route, same as [7], Cust is a customer, and CStat is a charging station.

Table 1. Single-point crossover example

Parent1	Depot	Cust1	CStat1	Cust2	CStat2	Cust3	Depot	Cust4	Depot
Parent2	Depot	Cust1	Cust2	Depot	Cust3	CStat2	Cust4	Depot	
Offspring1	Depot	Cust1	CStat1	Cust2	Cust3	CStat2	Cust4	Depot	
Offspring2	Depot	Cust1	Cust2	Depot	CStat2	Cust3	Depot	Cust4	Depot

From the example, we observe that Offspring1 which previously had 2 routes, has become a single route. While this is an improvement to reduce the number of vehicles by 1, from the Cust2 to Cust3, it is essential to make a stop to CStat2, like in Parent1 to recharge the battery, otherwise, it will cause a battery level violation. This is just a simple case where constraints are violated. Others that make random crossovers challenging are, for example, crossovers or routes which have missing customer nodes, or repeated visits to the same customer. Additional steps are required to remedy the infeasibility and violations. Unfortunately, randomly occurring crossovers are ineffective as they frequently generate infeasible offspring.

Some interesting points to note are that while a solution has multiple routes, 2 constraints, i.e. vehicle load utilisation and battery levels, are route-specific and will not change if the route does not. While the constraint on used vehicle time is affected by changes in other routes, it is largely only affected by a change in charging station visits. Hence, the crossover operator would ideally be one that can pass on desirable routes from parents to offspring, with minimal or no changes to individual routes. Also, it should have minimal changes to charging station visits. It should also be quick to compute because it is a common operator that would be run multiple times throughout the course of the genetic algorithm.

A crossover operator that fulfils these requirements is the Best Route Better Adjustment Recombination (BRBAX) proposed in [2]. From two parents, the operator will pass on the top 50% of routes verbatim from one parent to the offspring, then insert the rest of the nodes not in the offspring from the second parent, preserving the same order. The top 50% of routes are determined by the vehicle load utilisation, and higher utilisation is preferred. With that, BRBAX works within the triple constraints in a fitting manner, with vehicle load utilisation and battery level remaining the same, or even reduced in the offspring. The only possibly affected constraint is the used vehicle time as there might be changes to charging station visit times.

5 Modified Mutation

Due to the triple constraints of our problem, random mutations tend to generate infeasible solutions. In this section, we propose modified mutation operators.

We observe that the mutation operation in [7] exploited some form of local search. Something noteworthy about the local search mutation operator used is the mitigation technique employed when the mutation causes routes within the solution to become infeasible. What was done previously to remedy the solution was that for routes that were infeasible, their customer nodes were removed until they became feasible, and the evicted nodes were placed into a newly created route. While this might work well for the objective in [7] to minimise total travel, it only serves to worsen the objective value of minimising the number of vehicles. Hence, the conventional mutation operators with the focus of primarily improving population diversity would not be effective. In the following, we propose a method that is more akin to the existing local search mutation where there is a higher chance of a good mutation that improves the solution.

5.1 Route Elimination Algorithm

A desirable mutation operator is one that can effectively work within the triple constraints, is quick to compute, and able to improve the objective value. The Route Elimination Algorithm in [9] proposed to immediately eliminate one route by putting all nodes into an ejection pool and then inserting them into existing routes. However, inserting nodes into existing routes proves to be challenging. To address this, the authors introduced Squeeze and Perturb operators which iteratively attempt to remove nodes from the ejection pool and insert them into routes. In essence, the concept is to insert the customer nodes which are *hard* to insert first, followed by the easier customers later. The difficulty to insert is represented by a heuristic penalty term, which is derived through multiple iterations. These operators are not suitable in the mutation operator because it is an iterative process, which is computationally expensive.

5.2 Node Insertion with Triple Constraints

Even though we are unable to utilise the Squeeze and Perturb operators directly, the route elimination algorithm can be adapted. The idea is to insert a customer into a relatively good route and position, without having to do so iteratively. A critical problem is to insert ejected customer nodes into existing routes that will satisfy the triple constraints. First, ensuring vehicle capacity constraints is easy, since cumulative loads can be computed in constant time. Second, the constraint on total vehicle time per route is the most challenging, because it cannot be computed on a route level due to possible queuing time caused by *other* vehicles waiting at charging stations. Furthermore, even if a route has not changed, it might become infeasible due to changes on a different route. Thirdly, we have a constraint on the battery level. Unlike the constraint on vehicle time per route, this constraint is self-contained. However, it is not as simple as the vehicle capacity constraint, because when a new node is inserted to the route, we must calculate whether it is feasible in terms of battery level.

In our problem, we always charge the battery to full in every charging station visit; therefore, we can calculate the battery levels starting from the depot or charging station prior to the insertion until the depot or charging station, whichever comes first. However, this means that the computation for battery level feasibility is always done retrospectively after a new customer node has been inserted. To reduce the number of failed insertions due to battery level constraints, we can precompute the minimum battery level requirement of each node, which is the lowest battery level a vehicle can be at when visiting that node in order to safely travel to a charging station to recharge.

5.3 Battery-Feasible Neighbours

To further improve the efficiency of deriving feasible nodes for insertion, we can make use of the minimum battery level to derive a set of battery-feasible neighbours for every node. A node's battery feasible neighbour is one that a vehicle can safely visit and be sure that the battery level constraint is not violated, i.e. we can subsequently make an insertion of a charging station to recharge the battery *prior to* arriving at the neighbour, if required. The algorithm to generate the set of battery-feasible neighbours should take into consideration that the vehicle may have had to travel from a previous node, and that there is a maximum battery level that a vehicle can have, once it is at that node. For example, assuming a full battery of 100, node i to node j takes 90 units battery cost, but node i is 20 units battery cost from its nearest charging station. Hence, the maximum battery level at node i is $100 - 20 = 80$, and a vehicle at node i will never make it to node j. The algorithm for computing battery level feasibility is given as follows:

Algorithm for Computing Maximum Battery Level for Node *i*

1. Iterate through all charging station nodes to find the *nearest* charging station to node i, denoted *cs1*
2. Calculate the battery cost for travelling from node i to *cs1*, bc(i, *cs1*)
3. Calculate the battery cost for travelling from node i to depot, bc(i, depot)
4. Calculate *minimumBatteryLevelRequirement* = min(bc(i, *cs1*), bc(i, depot))
5. Calculate *maximumBatteryLevel* = *fullBatteryLevel* – *minimumBatteryLevelRequirement*

For a node j to be a battery-feasible neighbour of node i, we check if it is reachable with the maximum battery level of node i, and if it has sufficient battery to visit a charging station afterwards. For example, assuming a full battery of 100 units, node i to node j takes 90 units battery cost, but the nearest charging station is 20 units battery cost away. Once a vehicle travels from node i to node j, it will have a remaining battery level of $100 - 90 = 10$, which is insufficient for going to the nearest charging station. Hence, node j should not be included in node i's battery-feasible neighbour set.

Algorithm for Generating Battery-Feasible Neighbours for Node *i*
Iterate through every node $j \neq i$

a. Calculate battery cost from node i to node j, bc(i,j)

b. If *maximumBatteryLevel* > *minimumBatteryLevelRequirement* + bc(i,j) add node j to the battery-feasible neighbour set of node i

5.4 Route Merge Local Search (RMLS) as Mutation Operator

We integrate the above two subsections to form our mutation operator. The first step of the route elimination is to remove a route. Given a chromosome, we would compute the total vehicle load utilisation of each route, and select the route with the lowest load utilisation for elimination. All customer nodes of the eliminated route will be added to an ejection pool, ignoring depot and charging stations.

For insertion, it is done one node at a time. First, a node to be inserted, node i, is randomly selected from the ejection pool. Then, the route with the lowest vehicle load utilisation with nodes in the node i's battery-feasible neighbour set is selected as a candidate route for insertion. If battery level constraint causes infeasibility, there is a remedy in the form of charging station reallocation, explained in the next section.ote that we might still violate the used vehicle time constraint. Then, we proceed to find the position to insert node i that adds the shortestdistance to the route.

The algorithm will repeat node insertion until the ejection pool is empty or it is unable to insert any nodes due to violating the constraints. If it fails to insert, we would consider it a failed mutation and skip it entirely. One possible downside to the route merges local search is that it can be too aggressive in hill-climbing, as it is following a simple greedy strategy. However, the mutation is only performed on a certain percentage of the population in the genetic algorithm, and that it would be able to pass the good sections of the solution to the next generation. Moreover, this is just one part in the overall genetic algorithm, we look to employ other methods to prevent convergence into a local optimum.

5.5 Charging Station Reallocation

After every node has been inserted, it is still possible that the battery level constraint is violated. However, we can perform some reallocation of charging stations to ensure that the constraint is satisfied. An example where battery level constraint is violated before and after charging station reallocation is given in Table 2. Assume that it costs exactly 20 battery units to move from one customer node to another, and exactly 10 battery units to move to a charging station.

Table 2. Reallocate charging stations

Before – Route	Cust1	Cust2	Cust3	
Before – Battery Level	30	10	−10	
After – Route	Cust1	Cust2	CStat1	Cust3
After – Battery Level	30	10	100	80

Algorithm for Charging Station Reallocation for a Route

1. Remove all charging stations in the route.
2. Iterate through the route to calculate the battery cost for every node the vehicle visits and find out at which node the battery is lower than 0, node *i*.
3. From node *i*, we will then iterate backwards until we identify a charging station or depot node, node *j*.
4. Insert a charging station where there is the smallest increase in route travel time.
5. Repeat from Step 2 until the solution is battery-feasible.

6 Diversity Induced Population Selection

The diversity-controlling adaptive genetic algorithm [15] is only useful if an increased rate of crossovers and mutations is able to introduce diverse offspring or solutions. Based on our prior experiments with BRBAX and Route Merge Local Search operators, the observed diversity is reduced as the GA continues into the later generations. In the following, we discuss our approach for ensuring population diversity.

6.1 Fitness Function vs Objective Function

For our work, there is a distinct difference between the objective function and fitness function. One such application of differing objective function and fitness function is fitness sharing [10], a popular method to increase diversity in a population. The idea is to penalise the fitness of chromosomes that are closely similar to other chromosomes in the population, hence "sharing" their fitness with look-alikes.

Note that our objective function is to minimise the number of vehicles used. To encourage diversity we derive the fitness value depending on how different a chromosome is from the rest of the population, by adding a multiplier to its objective value. In doing so, the chromosome with the best fitness might not be one with the best objective value.

Our proposed fitness function after accounting for diversity is as follows:

$$fCi = nCi \times \left(\frac{\text{average } \mathit{similarityScore}}{\mathit{diversityMultiplier}} \right) \tag{2}$$

where fCi is the fitness of Chromosome i and nCi is the number of vehicles in Chromosome i

The idea is to have higher diversity in the early generations and to add selection pressure in the later generations, a diversity multiplier is added. It is set at the start and reduced in every iteration. In our experiments, the range is from 1.0 to 5.0.

6.2 Similarity Score: Hamming Distance

To derive the similarity scores, we study different types of measures, such as genotypic and phenotypic. [14] proposed that the genotypic similarity between chromosomes can be measured by the Hamming distance, and phenotypic similarity can be measured with the difference of the objective value, e.g. A chromosome that uses 11 vehicles is more similar to a chromosome that has 10 vehicles than a chromosome with 13 vehicles.

For example in Table 3, suppose each chromosome has a single route and the distance is 1 for every non-match.

Table 3. Hamming distance between chromosomes

Chromsome1	Depot	Cust1	Cust2	Cust3	Cust4	Depot
Chromsome2	Depot	Cust4	Cust1	Cust2	Cust3	Depot
Distance	0	1	1	1	1	0

The Hamming distance between Chromosome1 and Chromosome2 will be 4, even though they serve the same nodes, and also share a common subsequence [1–3]. Clearly, there are some features of similarity that are not captured by Hamming distance. Hamming distance would be a good measure to use for when the sequence of visits matters. In addition, if we were to introduce more routes, there would also be the complexity of deciding which pairs of routes would be used to calculate the Hamming distance. Since our focus is on minimising the number of vehicles, the sequence of visits would be less important, because no matter which sequence of visits, it is still served by 1 vehicle. Using a phenotypic measure, such as the objective value is also not desirable, since 2 chromosomes with good, diverse solutions will be penalised purely for having the same objective result. What is preferred is a population comprising a diverse set of good chromosomes, instead of only 1 good chromosome with the other chromosomes having a poor performance just for the sake of diversity.

6.3 Jaccard Index for Chromosomes

The key feature to capture is which customers are visited by the same vehicle. Here, we can utilise the features of an unordered collection of unique elements. For a given chromosome, it contains a set of routes, and within each route, contains a set of nodes to visit. A simple measure for set similarity is the Jaccard Index:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{3}$$

In Table 4, since the two chromosomes share 1 common route, Chromosome3 $\cap$ Chromosome4 $= 1$, and since the number of unique routes combined is 3, Chromosome3 $\cup$ Chromosome4 $= 3$. Hence, J(Chromosome3, Chromosome4) $= 1/3$.

Table 4. Chromosomes for Jaccard Index example

Chromosome3	Depot	Cust1	Cust2	Cust3	Depot	Cust4	Cust5	Cust6	Depot
Chromosome4	Depot	Cust3	Cust1	Cust2	Depot	Cust7	Cust8	Cust9	Depot

6.4 Double Jaccard Index

We now introduce the Double Jaccard Index as our similarity measure, as it can handle sets of routes that are highly similar even though they are not *exact* matches. The idea is that the Jaccard Index on chromosome-level is calculated with the Jaccard Index on route-level, with set membership determined by a pre-defined threshold (routeMatchThreshold).

As shown in Table 5, even though RouteA and RouteB are similar (out of 7 nodes, first 5 are the same), the routes are not an exact match and will not be considered as a match in the Jaccard Index calculation above.

Table 5. Routes for Double Jaccard Index example

RouteA	Dpot	Cust1	Cust2	Cust3	Cust4	Cust5	Cust6	Cust7	Dpot
RouteB	Dpot	Cust5	Cust1	Cust2	Cust3	Cust4	Cust8	Cust9	Dpot

To account for such cases, when determining if routes match, we would also use the route Jaccard Index, and consider it a match when the Jaccard index is greater than a predefined route matching threshold. For simplicity, we will only calculate the route Jaccard Index based on customer nodes only. In the above, Cust1 to Cust5 are common in both RouteA and RouteB, hence RouteA $\cap$ RouteB = 5. The number of unique routes in RouteA $\cup$ RouteB = 9. Hence, J(RouteA, RouteB) = 5/9.

Then, to calculate the Jaccard Index between Chromosomes i and j, we iterate every route in Chromosome i, and compare with every route in Chromosome j to calculate the Jaccard Index between each route pair given above. If this Jaccard Index exceeds the *routeMatchThreshold*, we increment the size of the intersection size (i.e. numerator) by 1. A similar calculation is performed on the denominator.

Finally, to calculate the similarity score of a chromosome with respect to the population, we apply the following algorithm:

Naïve Algorithm to Calculate Similarity Score for a Chromosome *i*:

1. Initialise *sumJaccardIndex* to 0
2. Iterate through all chromosomes in population except for chromosome i to compute Jaccard Index with these chromosomes and add the computed value to *sumJaccardIndex*
3. *similarityScore* = *sumJaccardIndex*/(population size – 1)

The Double Jaccard Index calculation, if performed naively, is computationally expensive. In essence, the above algorithm exhaustively iterates through all combinations. Let R denote the number of routes, and N denote the number of customer nodes. Since every pair of nodes of a given pair of routes must be considered, and every possible route pair is compared, the time complexity is $O(R^2N^2)$ to calculate one Double Jaccard Index. Since the calculation of similarity score has to be done in every generation and for every chromosome, the resulting complexity is high. From our experiments, when we included the Double Jaccard Index calculation, the runtime of the GA increases tremendously. As such, this algorithm must be made more efficient.

One way to achieve that is to use a lookup table that maps which nodes are on which route. The algorithm is given as follows:

Efficient Double Jaccard Index Algorithm between Chromosomes(i, j)

1. Initialise *numUniqueRoutes* to be number of routes in Chromosome i
2. Initialise *numRoutesIntersectBothChromosome* = 0
3. Iterate through every customer node in Chromosome i, and create a lookup table of node to route mapping for Chromosome i, *nodeIdToRouteCiMapping*
4. Iterate through every route r in Chromosome j
    a. Initialise set of customer nodes from r, *c2RouteNodes*
    b. Initialise *minNumNodeMatchesForC2Route* = ⌈size of *c2RouteNodes* * *routeMatchThreshold*⌉
    c. Iterate through every customer node in route r, and get their corresponding route in Chromosome 1 by looking up *nodeIdToRouteCiMapping*, then keep a count of how many occurrences on each route into a separate table, *routeToCountTable*
    d. Using the largest count from the *routeToCountTable*, if it is smaller than *minNumNodeMatchesForC2Route*, increment *numUniqueRoutes*
    e. Otherwise, union route r and route with the largest count from Chromosome1, and get the size of the union set.
    f. If the size of the union set * *routeMatchThreshold* is greater than the largest count, increment *numUniqueRoutes*, else increment *numRoutesIntersectBothChromosome*
5. Set Jaccard Index = *numRoutesIntersectBothChromosome*/*numUniqueRoutes*

With the above algorithm, we have reduced the complexity from the naïve algorithm of O(R^2N^2) to O(N), which greatly improves the run time complexity of GA.

7 Experimental Analysis

7.1 Experimental Setup

To evaluate the effectiveness of our algorithm, we performed experiments on the same problem instances as those provided in [7], which is a set of modified Solomon instances with charging stations as well as battery capacity. We will omit instance c101_21 as there is no feasible solution. The instances are divided into 3 categories: clustered (C) random (R) and randomly clustered (RC).

We first obtained a set of EVRP benchmark results by running the GA proposed in [7] with the *modified* objective function defined in Sect. 1 (to ensure fairness in comparison) with the value of M set as 100,000. We used a population size of 100, and 5000 generations. Crossover occurs once per generation, with parents selected using binary tournament selection. The mutation occurs twice per generation, one is for the chromosome with the best fitness, another is selected using uniform-random. All experiments were performed for 5 times, and the best results were presented. We then tested the effectiveness of our proposed crossover and mutation operators individually to evaluate their performance. Finally, we applied them together with the diversity population selection technique, with the route matching Jaccard index thresholds at 1.0 and 0.25. This way, we evaluated how well the operators work together, as well as the impact of different route matching thresholds.

Table 6 shows the benchmark instances and results obtained on all 100-customer modified Solomon instances presented in [7]. The numbers in brackets () represent the results published in [7] with their *original* objective function (that minimises total travel duration). It can be seen that the modified objective achieves a smaller number of vehicles in 20 out of 28 instances, equal numbers in 4 instances, and more vehicles in 4 instances.

Table 6. EVRP benchmarks

Instances	# of Vehicles	Instances	# of Vehicles
c102	11 (13)	r107	12 (14)
c103	13 (12)	r108	13 (14)
c104	12 (15)	r109	15 (15)
c105	12 (13)	r110	15 (14)
c106	14 (11)	r111	12 (14)
c107	14 (17)	r112	14 (12)
c108	11 (13)	rc101	13 (15)
c109	10 (19)	rc102	14 (17)
r101	14 (14)	rc103	15 (18)
r102	13 (16)	rc104	15 (15)
r103	13 (13)	rc105	14 (17)
r104	13 (14)	rc106	14 (18)
r105	14 (17)	rc107	15 (15)
r106	15 (12)	rc108	14 (15)

7.2 Minimum Number of Vehicles Comparison

Figure 1 shows the comparison of results of best solutions obtained by each combination of our proposed approaches against the EVRP benchmarks.

From Fig. 1, we observe improvement over the EVRP benchmarks (as shown in Table 6). Interestingly, our proposed approach could outperform both the old and new results given in Table 6, for both the old and new results. In fact, for the clustered instances, we managed to achieve the minimum number of vehicles possible. For these clustered instances the total demand of all customers is 1810, and the load capacity per vehicle is 200. Hence, the minimum number of vehicles required $= \lceil 1810/200 \rceil = 10$.

Another interesting point is that RMLS yields slightly better results compared with BRBAX+RMLS+Diversity(1.0). However, using BRBAX+RMLS+Diversity(0.25), we can obtain the same average performance as RMLS for the best result. We suspect that

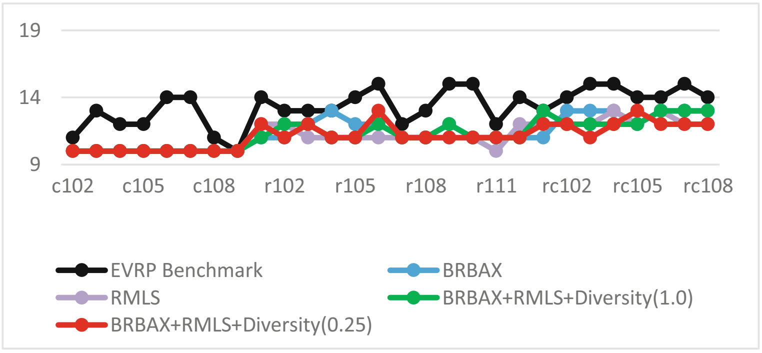

Fig. 1. Best minimum number of vehicles

a route matching threshold of 1.0 might be too conservative, leading to high population similarity. In other words, allowing partial matching between routes can potentially yield an improvement in results.

BRBAX+RMLS+Diversity(0.25) runtime performance across all instances averages to around 50 min per run.

7.3 Average Minimum Number of Vehicles Comparison

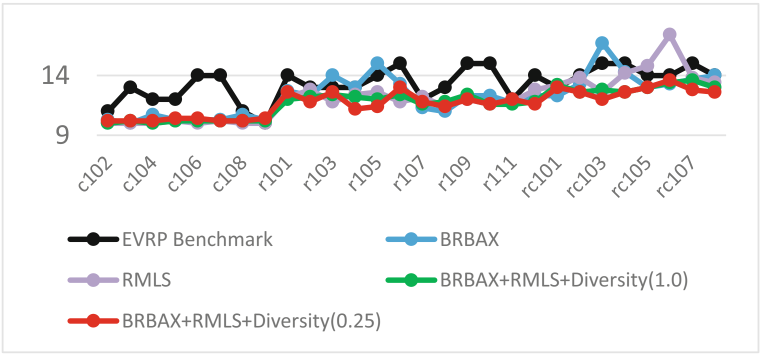

Fig. 2. Average minimum number of vehicles

Figure 2 shows the *average* (instead of best) performance over the 5 runs. Apart from 4 outlier instances, both BRBAX and RMLS showed improved performance against the benchmark.

We observe that regardless of route matching threshold value, BRBAX+RMLS+Diversity outperformed RMLS. We believe this phenomenon is observed because RMLS is highly exploitative and can easily be stuck in a local optimum.

In summary, BRBAX+RMLS+Diversity would be the preferred algorithm if consistent results are preferred. For best results, use a low route matching threshold like 0.25 as a threshold of 1.0 can be too conservative.

7.4 Total Travel Duration Comparison

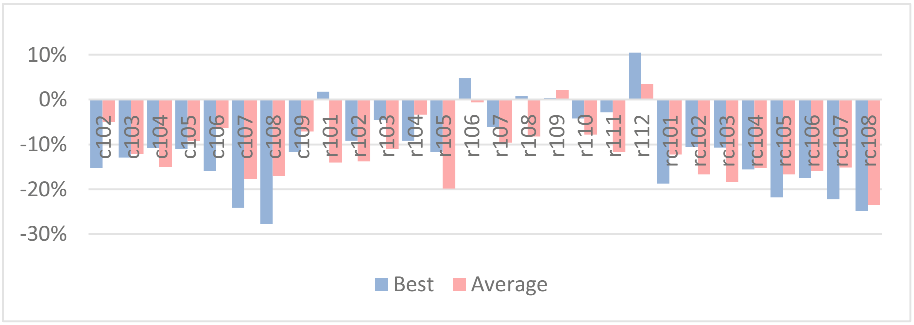

Fig. 3. Total route duration, BRBAX+RMLS+Diversity(0.25) vs [7]

Figure 3 compares BRBAX+RMLS+Diversity(0.25) against [7] on the total route duration measure based on percentage change. We observe that there is a significant reduction in the total route duration across all (C) and (RC) instances. However, for (R) instances, there is a small handful where our algorithm performs worse. This might be because there is a correlation between the number of used vehicles and the total route duration for each category of problem instances. The reason that our algorithm tends to perform better in clustered than uniform-random instances is that its focus is to reduce the number of used vehicles. With fewer vehicles, individual vehicles would have to serve more customers, and if customers are part of a cluster, travel time and battery usage will be reduced if the cluster is served by the same vehicle. On the contrary, random instances of customer nodes are unable to take advantage of cluster characteristics.

On average, we observe an 11% reduction in total route duration for both best and average measures. This further validates the efficacy of the algorithm; not only were we able to achieve reductions in the number of vehicles, but also in the total travel duration.

8 Conclusion and Future Works

In this paper, we proposed an efficient GA that solves EVRP which minimises the total number of vehicles. While the experimental results look very promising, we believe there are many opportunities to improve both solution efficiency and effectiveness. One possibility to explore is to find another heuristic to remove nodes for RMLS instead of the greedy heuristic on vehicle load utilization. Removal of nodes into the ejection pool does not need to be strictly from one route. GA will come to a point where it is challenging to reduce the number of routes, where it might be good to start improving on the secondary objective, to reduce the total of the needed time. GA seems to perform well for clustered instances; this would probably be because the BRBAX and RMLS were using vehicle load utilisation as a heuristic measure. One possibility to explore is using different heuristic measures to select good routes for BRBAX.

Acknowledgements. This research is supported by the National Research Foundation, Singapore under its Corp Lab @ University scheme and Fujitsu Limited as part of the A*STAR-Fujitsu-SMU Urban Computing and Engineering Centre of Excellence.

References

1. Baker, B., Ayechew, M.: A genetic algorithm for the vehicle routing problem. Comput. Oper. Res. **30**(5), 787–800 (2003)
2. Bermudez, C., Graglia, P., Stark, N., Salto, C., Alfonso, H.: A comparison of recombination operators for capacitate vehicle routing problem. Inteligencia Artif. **14**(46), 34–44 (2010)
3. Chen, H., Murata, T.: Optimal electric vehicle routing for minimizing electrical energy consumption based on hybrid genetic algorithm. Lect. Notes Eng. Comput. Sci. **2239**, 526–531 (2019)
4. Gustafsson, T., Johansson, A.: Comparison between battery electric vehicles and internal combustion engine vehicles fueled by electrofuels—from an energy efficiency and cost perspective (2015)
5. Hutter, M., Legg, S.: Fitness uniform optimization. IEEE Trans. Evol. Comput. **10**(5), 568–589 (2006)
6. Kumar, S., Panneerselvam, R.: A survey on the vehicle routing problem and its variants. Intell. Inf. Manag. **04**(03), 66–74 (2012)
7. Li, B., Jha, S.S., Lau, H.C.: Route planning for a fleet of electric vehicles with waiting times at charging stations. In: Liefooghe, A., Paquete, L. (eds.) EvoCOP 2019. LNCS, vol. 11452, pp. 66–82. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-16711-0_5
8. Miller, B., Goldberg, D.: Genetic algorithms, selection schemes, and the varying effects of noise. Evol. Comput. **4**(2), 113–131 (1996)
9. Nagata, Y., Bräysy, O.: A powerful route minimization heuristic for the vehicle routing problem with time windows. Oper. Res. Lett. **37**(5), 333–338 (2009)
10. Oliveto, P., Sudholt, D., Zarges, C.: On the benefits and risks of using fitness sharing for multimodal optimization. Theoret. Comput. Sci. **773**, 53–70 (2019)
11. Ombuki, B., Ross, B., Hanshar, F.: Multi-objective genetic algorithms for vehicle routing problem with time windows. Appl. Intell. **24**(1), 17–30 (2006). https://doi.org/10.1007/s10489-006-6926-z
12. Pereira, F.B., Tavares, J., Machado, P., Costa, E.: GVR: a new genetic representation for the vehicle routing problem. In: O'Neill, M., Sutcliffe, R.F.E., Ryan, C., Eaton, M., Griffith, N.J.L. (eds.) AICS 2002. LNCS (LNAI), vol. 2464, pp. 95–102. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45750-X_12
13. Prins, C.: A simple and effective evolutionary algorithm for the vehicle routing problem. Comput. Oper. Res. **31**(12), 1985–2002 (2004)
14. Zhu, K.: Population Diversity in Genetic Algorithm for Vehicle Routing Problem with Time Windows (2004). http://www.cs.sjtu.edu.cn/~kzhu/papers/zhu-ecml04.pdf. Accessed 6 Apr 2020
15. Zhu, K.: A diversity-controlling adaptive genetic algorithm for the vehicle routing problem with time windows. In: Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence (2003)