IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received September 3, 2020, accepted September 7, 2020, date of publication September 14, 2020,
date of current version September 25, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3023741

A Two-Phase Distributed Ruin-and-Recreate Genetic Algorithm for Solving the Vehicle Routing Problem With Time Windows

THAU-SOON KHOO[1], BABRDEL BONAB MOHAMMAD[1], (Member, IEEE), VOON-HEE WONG[2], YONG-HAUR TAY[3], AND MADHAVAN NAIR[1]

[1]Centre of Artificial Intelligence and Computing Applications, Universiti Tunku Abdul Rahman, Selangor 43000, Malaysia
[2]Centre of Mathematical Science, Universiti Tunku Abdul Rahman, Selangor 43000, Malaysia
[3]Recogine Technology Sdn Bhd, Putra Heights, Selangor 47650, Malaysia

Corresponding authors: Thau-Soon Khoo (khoothausoon@yahoo.com) and Babrdel Bonab Mohammad (babrdel@utar.edu.my)

This research is supported by the Universiti Tunku Abdul Rahman, UTAR Research Fund (UTARRF), no. IPSR/RMC/UTARRF/2019-C2/M01.

ABSTRACT Developing an algorithm that can solve the vehicle routing problem with time windows (VRPTW) and create near-optimal solutions with the least difference in magnitude is a challenging task. This task is evident from the fact that when an algorithm runs multiple times based on a given instance, the generated solutions deviate from each other and may not near-optimal. For this reason, an algorithm that can solve these problems is effective and highly sought after. This article proposes a novel systematic framework using a two-phase distributed ruin-and-recreate genetic algorithm (RRGA). The two-phase consists of the RRGA phase and ruin-and-recreate (RR) phase, which is designed to run in the distributed computing environment that leveraging these networked resources. This combination of algorithms harnesses the strength of the search diversification and intensification, thereby producing very high-quality solutions. Experiments with Solomon's benchmark show the RRGA can produce results superior to the recently published hybrid algorithms, best-known solutions, and nine leading hybrid algorithms.

INDEX TERMS Combinatorial optimization, genetic algorithm, objective function, ruin-and-recreate, vehicle routing problem with time windows.

I. INTRODUCTION

The VRP was introduced by Dantzig and Ramser [1] as a truck dispatching problem. It is a prevalent logistics optimization problem and ubiquitous in the logistics and distribution industry. There are many variants [2] in the VRP, and one of the popular variants is VRPTW. The original VRPTW was introduced by Solomon [3]. It is an NP-hard [4] combinatorial optimization problem. The objective is to minimize the total travelled distance within the constraint of vehicle load capacity, customer time window (customer's availability time), service time, depot time window (depot's availability time), among others. In VRPTW, the vehicles are homogenous in their specifications. Initially, a dedicated fleet of vehicles is assigned and stationed at the central depot. These vehicles serve customers who are located in a disparate location. They should pick up all customer orders at different locations based on x and y coordinates, which mimic the real-life example of a merchandise delivery to a customer location. The distance is calculated using the Euclidean distance, which is a straight-line distance. Furthermore, these vehicles must also satisfy the following constraints:

- All vehicles must end at the same central depot.
- All vehicles must operate within the start and end time of the central depot.
- There is a finite set of vehicles to deliver customer demands.
- Each customer must serve only once by a vehicle.
- The vehicle must arrive at the customer's location within the customer's availability time. If it arrives early, it must wait until the customer is available.
- Each service time is associated with each customer.

Initially, VRPTW was solved using the exact algorithm. Despite the structural problem, the computational times increase with the size and complexity of the problem [5]; thus, the exact algorithms were not able to cope. Therefore,

The associate editor coordinating the review of this manuscript and approving it for publication was Shiping Wen.

a heuristic/metaheuristic algorithm was created to overcome these complexities. Metaheuristic algorithms have been used since the 40s, even though they were not studied formally [6]. These algorithms scale well to a large and complex problem; They achieve better results with the current problem complexity [7]. GA is one of the popular metaheuristic algorithms. It was discovered and presented by Holland [8] as a nonlinearity mathematical model that mimicked the biological process of adaptation [9]. GA modifies each solution in the population. During the selection step, it selects the individuals as the parents based on fitness value to produce offspring, which is later used for crossover and mutation step [10]. Through this repeatable process, the results evolve towards better and optimal results. However, the classical design of GA has its drawbacks [10], [11]. It is computationally expensive, requires parameter tuning and no guarantee of achieving the optimal result within a finite time [12], [13]. Despite these drawbacks, many applications still use GAs for deriving better solutions either by changing its internal structure [14], [15] or by combining with other algorithms [16], [17]. Another popular algorithm for solving VRPTW is the neighbourhood method, which is based on ruin-and-recreate (RR) principle. This principle was initially coined by Schrimpf *et al.* [18]. It is often used for finding a satisfactory solution within a reasonable time by ruin (removal) the existing solution and followed by an improvement (reconstruction) procedure [19], [20]. Some neighbourhood algorithms are complicated [21], [22]. However, the RR principle is still a low-level, adaptable, fast, and powerful to be used with other VRP variants. The removal and reconstruction of the incumbent solution can be varied and depending on the strategy in use. It consumes less memory and obtains a reasonable solution in a large search space. Often, it is used with other algorithms as it improves the local search ability [23], [24] and results [25]–[27].

There is a less plethora of VRPTW literature which combines the evolutionary algorithm with the RR principle. Panagiotis *et al.* [28] proposed an arc-guided evolutionary algorithm that manipulated the individuals in the population using an evolution strategy. It used a discrete arc-based representation with a binary vector of strategy parameters. The proposed mutation operator is based on the ruin-and-recreate principle. Their algorithm improved the best reported cumulative and mean results within a reasonable computational requirement. Tan *et al.* [29] presented a hybrid multiobjective evolutionary algorithm (HMOEA), which included different heuristics for local exploration search in the evolutionary search and the Pareto optimality concept. It used a specialized genetic operator and a variable-length chromosome representation which accommodated the sequence-oriented optimization. Furthermore, their algorithm has lowered routing solutions, wider scattering areas, and better convergence traces. Their results are better and competitive with the best solutions in their published literature. Nacima *et al.* [30] suggested a memetic algorithm (MA) [31] with a local search. Each iteration of the local search evaluates different moves. The iteration in each of the local searches (neighbourhood exploration) for the first improving move and executes using MA it if it finds it. The local search stops when there is no possible further improvement. This algorithm offers faster execution time and attains several best-known results. JingTian *et al.* [28] suggested the evolutionary algorithm, scatter search, and particle swarm optimization algorithm (ESS-PSO) to solve VRPTW. In the evolutionary algorithm and scatter search (ESS), they use the genetic algorithm, and a new "route +/−" evolutionary operator is introduced in the scatter search template. Besides, a discrete particle swarm optimization (PSO) is also proposed, which set the route segment as the particle velocity. The velocity and position updating rules are based on the ruin-and-recreate principle. It also uses the cascade learning architecture in which PSO learns the ideal solution set by ESS. Their results are competitive, and ESS-PSO proves to be effective and efficient. Ziauddin *et al.* [17] proposed a novel framework which was called the localized optimization framework (LOF). It is composed of two phases, which are the optimization and deoptimization. The optimization was performed on the part of the problem, whereas the de-optimization was on the whole problem. On average, LOF performs better on a small scale than other heuristics. It also attains a few best solutions on the test datasets.

The most proposed algorithms achieve remarkable results by combining different algorithms. In this article, we propose a novel hybrid algorithm that takes advantage of the GA with the RR principle. The characteristics of a GA are a nature-inspired and a population-based searching algorithm. It uses a memory usage method, and their searching is stochastic and iterative. In RR, it is characterized as a nonnature-inspired and single-based search algorithm. It uses a memory usage method, and their searching is deterministic and iterative. Our proposed algorithm is designed by harnessing the strength of exploration and exploitation inherent in GA and RR. In space searching, exploration tends to be diversified, whereas exploitation tends to be intensified. These characteristics match perfectly in our proposed algorithm because we leverage the benefit of GA as the space searching is based on exploration (diversified-oriented) approach, and the space searching in RR is based on exploitation (intensified-oriented) approach. In the exploration approach, all regions in the search space are evenly explored, and no number of regions is reduced for searching. In the exploitation approach, the promising regions are explored more thoroughly with the hope of finding better solutions.

We made a few modifications to the classical GA. Firstly, we modify the selection step in GA to target the nonelite candidates. These changes mean the higher the nonelite candidates, the higher the fitness value. These nonelite candidates have a better chance to be selected as a candidate for crossover. Secondly, we remove the solution generation in each of the genetic operators (selection, crossover, mutation). In our design, all genetic operators are only responsible for generating customers, and thirdly, we integrate each of the genetic operators with the RR principle. In this step, each of

the genetic operators will pass over a customer list to the RR principle for solution generation.

The RR principle consists of the ruin and the recreate method. In the ruin method, we have evaluated a few strategies (random, radial, worst, clustered, and string) before it is integrated with the GA. We discovered, among other strategies proposed in the ruin method, the random and radial strategies deliver excellent results when we integrated into the proposed algorithm. However, in the recreate method, we propose the best insertion and regret insertion strategy. The main contributions of this article are listed as follows:

(1) We proposed an RRGA distributed application architecture.
(2) We presented two phases in the execution of the proposed algorithm, which consists of an RRGA phase and the RR phase that harness the exploration and exploitation potential.
(3) We introduced the varied strategies used in the ruin method of the RR principle, which enhances the exploitation of searching.
(4) We suggested a separation of duties and synergistic effects in which the GA is responsible for generating customers, and the RR principle is to transform the customers into a solution.
(5) We implemented a continuous improvement paradigm using the RR strategies for continuous generating near-optimal results for comparison to achieve the best optimal result.

The remainder of this article is organized as follows. Section 2 contains a description of the problem assumptions and notations. In section 3 explains the proposed algorithm. The result analysis is shown in Section 4. Finally, Section 5 gives an overall conclusion.

II. PROBLEM ASSUMPTIONS AND NOTATIONS

A. PROBLEM DESCRIPTION

The VRPTW problem consists of a set of homogenous vehicles and loading capacity. The objective is to achieve the best-travelled distance by traversing all customers. It must satisfy the constraints of the customer's time window and vehicle loading capacity.

B. PROBLEM DEFINITION

Let $G = (V, A)$ define a complete directed graph, where $V = \{c_0, \ldots\ldots, c_n\}$ is the node set, and c_0 is the depot and $c_{1..n}$ represents the first customer until the last customer n. Each customer is associated with a demand quantity q_i, except node c_0 is associated with $q_0 = 0$. The A is the arcs set, where $A = \{(c_i, c_j)|\ c_i, c_j \in V,\ c_i \neq c_j\ \}$ and each arc (c_i, c_j) is associated with travel time (t_{ij}) and represented as a distance cost $d_{ij} = d_{ji}$. Each customer is associated with preplanned loading demand and service time information. There is a finite vehicle (v) that has a homogenous capacity. The customer time windows (e_i, l_i) are known before the start of the journey. Each customer has a loading demand where the demand (q_i) is greater than or equal to zero. A hard time window is the predefined time interval where it must serve the customer between the customer earliest (e_i) and the latest available time (l_i), in which $e_i \geq 0$ and $l_i \geq 0$. The central depot is also associated with a time window and represented in e_0 and l_0, in which e_0 is the earliest start time, and l_0 is the latest end time at depot. The vehicle needs to wait if it arrives at the customer before the earliest available time (e_i). It could not serve the customer if it exceeded the latest available time. The vehicle also cannot serve the customer if its loading capacity has exceeded. Each customer (c_i) is assigned a time window, which is denoted as $[e_i, l_i]$. The accumulated service time (s_i) and travelling time (t_i) for each customer (c_i) must not exceed the depot latest available time (l_0). Each arc a_{ij} represents the unidirectional and distance (d_{ij}) of that arc. Each customer is served only once.

C. MATHEMATICAL MODELS

The following notations and formulas for deriving the VRPTW model [36] are listed as follows:

q_i	The demand quantity of the customer c_i
Q	The capacity of the vehicle.
s_i	The service time of the customer
d_{ij}	The travel time from customer c_i to customer c_j.
w_i	The waiting time of the customer c_i
e_i	The earliest available time of the customer c_i
l_i	The latest available time of the customer c_i

$$y_i^k = \left\{ \begin{array}{ll} v & \text{if vehicle } k \text{ travels from customer } ci \\ & \text{to customer } cj \\ 0 & \text{otherwise} \end{array} \right\}$$

The objective function is to minimize the total travelled distance, which is described as follows:

$$min\ TD = \sum_{k=1}^{K} \sum_{i=0}^{N} \sum_{j=0}^{N} t_{ij}.x_{ij}^k \tag{1}$$

$$\text{s.t. } x_{ij}^k = \left\{ \begin{array}{ll} 1 & \textit{if vehicle } k \textit{ travels from customer} \\ & i \textit{ to customer } j \\ 0 & \textit{otherwise} \end{array} \right\} \tag{2}$$

$$\sum_{j=1}^{N} x_{0j}^k = \sum_{i=1}^{N} x_{i0}^k = 1 \quad (\forall k = 1, 2, \ldots, K) \tag{3}$$

$$\sum_{j=0}^{N} x_{ij}^k = \sum_{j=0}^{N} x_{ji}^k \leq 1 \quad (i \neq j,\ \forall i = 1, \ldots, N;\ \forall k = 1, \ldots, K) \tag{4}$$

$$\sum_{k=1}^{K} \sum_{i=0}^{N} x_{ij}^k = 1 \quad (i \neq j,\ \forall j = 1, 2, \ldots., n) \tag{5}$$

$$\sum_{k=1}^{K} \sum_{j=0}^{N} x_{ij}^k = 1 \quad (i \neq j,\ \forall j = 1, 2, \ldots., n) \tag{6}$$

$$\sum_{i=0}^{N} qi \sum_{j=0}^{N} x_{ij}^k \leq Q \quad (i \neq j;\ \forall k = 1, 2, \ldots., K) \tag{7}$$

$$t_i^k + s_i + t_{ij} t_j^k \leq (1 x_{ij}^k).M \quad (i \neq j,\ \forall i, j = 0, 1, \ldots, N;\ \forall k = 1, 2, \ldots, K) \tag{8}$$

$$e_j \sum_{j=0} x_{ij}^k \leq t_j^k \leq l_j \sum_{j=0} x_{ij}^k \quad (i \neq j, \forall i = 0, 1, \ldots, N,\ \forall k = 1, 2, \ldots, K)\ (i \neq j,\ \forall i = 0, 1, \ldots, N;\ \forall k = 1, 2, \ldots, K) \tag{9}$$

$$x_{ij}^k \in 0, 1\} \quad (i \neq j, \forall i, j = 0, 1, \ldots, N;\ \forall k = 1, 2, \ldots, K) \tag{10}$$

Equation (1) shows that the primary objective function of the problem is to minimize the total travel distance (TD). Equations (2) and (3) represent the decision variable and ensures that each vehicle must depart and return to the same depot, respectively. Equation (4) is the node conservation constraints flow. Equations (5) and (6) ensure that each vehicle can serve the customer only once. Equation (7) represents that the total customers' demands cannot exceed the vehicle capacity. Equations (8) and (9) represent the time window constraint, where t_i is the time the vehicle k arrives at customer. M is a large constant. Equation (10) ensures binary conditions on the decision variable.

III. THE PROPOSED ALGORITHM

Our proposed algorithm is a distributed application architecture that consists of two main phases, which harness the strength of GA and the RR principle. We suggested three types of RR strategies. In the following sections, we explain the overall distributed application architecture, the phases of RRGA execution, the proposed strategies in each of the phases, and the pseudocodes.

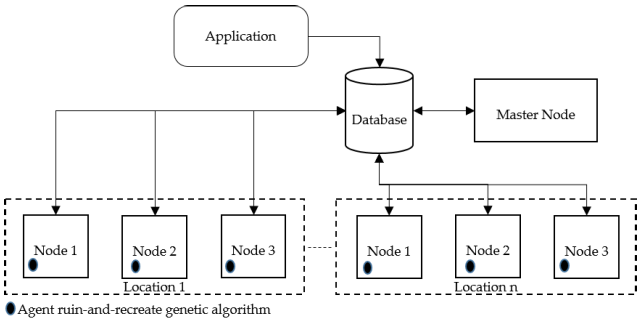

FIGURE 1. Distributed RRGA system architecture.

A. DISTRIBUTED RRGA SYSTEM ARCHITECTURE

Figure 1 illustrates the distributed RRGA system architecture. The design of the distributed RRGA system is based on master-slave architecture. In this diagram, the master node is the central computing that instructs the other nodes (slaves) to receive an order and perform computations. In order for the slave node to receive the instruction from the master, each slave node must install an RRGA agent first before the master node can send the instruction to the slave nodes. Once the RRGA agents are installed, it will register itself in the database so that the master node knows which slave node is ready to receive instruction.

Before the master node can instruct the slave nodes to carry out the instruction, the user must key in a request using an application. This application stores the data into the database. The master node will continuously read from the database, and if a new record is found, the master node will inform the slave nodes. The master node then updates the application table in the database to mark as the job started. Once the job finishes, the slave node will inform the master node on job completion, and the master node will update the record in the data. In this case, the master node updates the record in the database as the job completed.

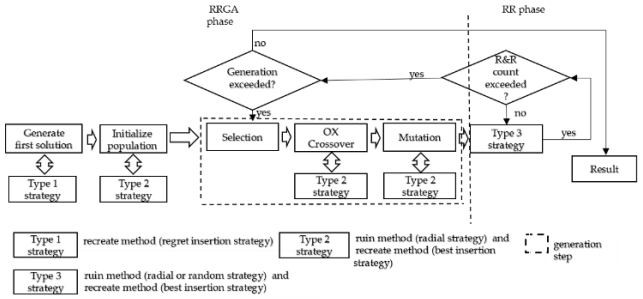

FIGURE 2. RRGA process flows lifecycle.

B. RRGA PROCESS FLOWS LIFECYCLE

The implementation of the proposed algorithm is organized into two phases, as shown in Fig. 2. The first phase is the RRGA phase, and the second phase is the RR phase. These phases constitute the lifecycle of the RRGA. In RRGA phase, each of the executing procedure is divided into two subprocedures. The first subprocedure is to generate a customer list and pass it to the second subprocedure to generate a solution. There are two types of RR strategies that can be used in the second subprocedure. The type 1 strategy combines the radial strategy in the ruin method with the regret insertion strategy in the recreate method. In type 2 strategy, the radial strategy is used in the ruin method is combined with the best insertion strategy in the recreate method. As shown in figure 1, the customer list from the generated first solution is passed to the type 1 strategy. The outcome of this procedure is passed to the population initialization subprocedure in which the customer list is executed using a type 2 strategy subprocedure. The customer list of this procedure is passed to the selection subprocedure, which it will call the type 2 strategy subprocedure. In summary, each of the customer list derived from the successive procedure will be passed to the next procedure until it reaches the final procedure. Each successive procedure is executed in sequential order.

In the RR phase, only the type 3 strategy is used in this procedure. The solution derives from RRGA phase is continuously improved until it reaches a terminating criterion. In type 3 strategy, the radial or random strategy in the ruin

method is randomly selected, and the best insertion strategy is used in the recreate method.

These segregations of duties in RRGA and RR phase promote global optimization where all possible solutions are explored thoroughly, and the final solution is intensely improved. Hence, the outcome of executing RRGA in phases leads to a better solution. We also evaluated other common strategies (worst, string, and cluster) in the ruin method [33]. However, their results are not better than the strategies suggested in this proposed algorithm. The pseudocode of the RRGA framework is shown in Algorithm 1. It shows the two phases, which are the RRGA phase (lines 1 to 9) and the RR phase (lines 11 to 17). The RRGA phase consists of generating the first solution using a type 1 strategy (line 1), population initialization (line 2), and generation step (line 6). In the generation steps procedure, it encapsulates the nonelite selection, crossover, and mutation procedures.

The RR phase uses the type 3 strategy (lines 12 to 13). The type 3 strategy of the RR principle consists of randomly selecting either a radial or random strategy procedure in the ruin method (line 12), and the best insertion strategy procedure in the recreate method (line 13). The generated solution is improved continuously until the termination criteria are made (line 11).

Algorithm 1 RRGA Framework

Input *R*: routes *RC*: removed customers
Output *Bs*: best solution
/* RRGA phase */
1. $S \leftarrow RegretInsertion(R, RC)$ //*S - solution*
2. $P \leftarrow Initialize\_population(S)$
3. $Bs \leftarrow$ *Get best solution based on P*
4. **for** $i \leftarrow 1$ *to* N **do** /* N – *number of generation steps* */
5. /* *execute the generation step* */
6. $S \leftarrow Generation\_step(P)$
7. **if** $S < Bs$ **then**
8. $Bs \leftarrow S$
9. **end if**
10. /* *RR phase* */
11. **for** $i \leftarrow 1$ *to* M **do** /* M – *number of RR iteration* */
12. $S, RC \leftarrow radial\_or\_random\_ruin\ (Bs)$
13. $S \leftarrow bestinsertion(S, RC)$
14. **if** $S < Bs$ **then**
15. $Bs \leftarrow S$
16. **end if**
17. **end for**
18. **end for**
19. **return** *Bs*

1) CHROMOSOME REPRESENTATIONS

Each chromosome consists of a string of genes. These genes are analogous to a string of customers and represented in digits. Each digit represents the identity of the customer. The position of the genes in the chromosome is essential for route and distance calculation. These genes are shuffled and arranged randomly using RR strategy before they are used for the next procedure. The terms ''chromosome'' and ''individual'' are used interchangeably in this article. Fig. 3 depicts how the customers in VRPTW are represented, encoded as the customers, and decoded into individuals or chromosomes with the defined number of routes, total travelled distance, and prearranged customers in the list. The number of routes in an individual is calculated based on the defined constraints (i.e., time windows, service time, vehicle capacity, depot start and end time, and among others), and the total travelled distance is calculated based on the sum of distances on all the routes. Algorithm 2 initializes the population. In lines 1 to 6, each of the individuals is created using type 2 strategy (lines 4 to 5).

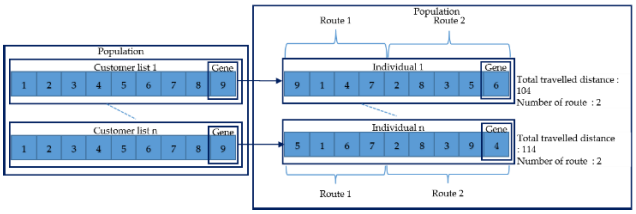

FIGURE 3. Individual (Chromosome).

Algorithm 2 Population Initialization

Input *S*: solution
Output *Pop*: population
1. $R \leftarrow null$
2. **for** $i \leftarrow 1$ *to* N *do //loop the customer size*
3. /* $Pop_i - i^{th}$ *individual (solution) in the population* */
4. $R, RC \leftarrow radialruin(S)$ /* R – *routes, RC –removed customers* */
5. $Pop_i \leftarrow bestinsertion(R, RC)$
6. **end for**
7. **return** *Pop*

2) GENERATION STEP

The generation step shows in Algorithm 3 consists of three main steps. They are the nonelite selection procedure (lines 2 to 16), the order crossover (OX) procedure (lines 18 to 24), and the mutation procedure (line 29).

Firstly, the individuals in the population are sorted ascendingly (line 1). This way of sorting indicates the individual with the shortest total travelled distance will appear at the top of the population list. Next, each of the solution fitness is calculated (line 2) using formula 10. In this computation, each individual in the population list is associated with a calculated fitness value. In this manner, the bottom of the population list has a higher fitness value than the individual appears at the top. This outcome explains the nonelite individuals to appear at the bottom of the population list have a higher chance to be selected as a parent for crossover. The elite individuals that appear at the top of the population list may cause premature convergence and diversity loss.

Therefore, we choose the nonelite individuals. However, this high probability of nonelite individuals to be selected is tied to selection pressure (sp). The higher the selection pressure that applies to the individuals in the population, the better the individual evolves. Each fitness value for each individual is calculated using equation (11). Once the fitness value of all individuals in the population list is calculated, it is summed up (line 3) in equation (12). Line 6 shows a fitness value is randomly selected from the total sum of fitness values using the equation (13).

$$f_{\mathrm{i}} = 2 - sp + 2^{*}\,(sp - 1)^{*}\,\frac{i}{N-1} \tag{11}$$

$$Sum\ of\ Fitness(F) = \sum_{\mathrm{i}=1}^{\mathrm{N}} \mathrm{f_i} \tag{12}$$

$$individual = \sum_{i=1}^{N} F_i > random[0..1].F \tag{13}$$

Lines 8 to 17 identifies the position of each individual in the population to be selected for the order crossover (OX). In lines 19 to 20, the two cutoff points are selected. If the two cutoff points have the same position, it will randomly select two cutoff points until it does not match (lines 21 to 23). Lines 24 to 25 show the two newly generated offspring using the OX procedure. Lines 26 to 28 indicate the two newly generated offsprings are added to the new population. The process of generating new offsprings into a new population continues until it exceeds its population size. Each individual in the population is processed by mutation procedure (line 31) and sorted ascendingly (line 32).

Lines 33 to 35 show the individuals in the new population replace the individuals in the original population. Finally, the individual in the original population will replace the best solution if the result is better (line 36).

3) ORDER CROSSOVER

Algorithm 4 shows an OX algorithm [38], and this procedure requires four parameters. The first two parameters are the different parents (individuals), and the second two parameters are the two different cutoff points which are essential for the crossover process. If the second cutoff point has the same position as the last position of the first parent, then the entire customers in the first parent will be copied to the segment array (lines 3 to 6). If it is not, then the customers between the second cutoff point until the last position of the first parent are copied to the segment array (lines 8 to 11). Next, the customers between the first position and the second cutoff point of the first parent are appended to the earlier segment array (lines 12 to 15). The customers between the first cutoff and second cutoff points of the second parent are removed from the segment array (lines 17 to 23). Lines 24 to 26 show that the customers between the first and second cutoff points are copied to the same position of the offspring. The rest of the customers in the segment array fill the offspring starting after the offspring second cutoff point (lines 28 to 41). The offspring is formed using the type 2 strategy (lines 42 to 43). Figure 4 summarizes the crossover process.

Algorithm 3 Generation Step

Input *Pop*: population
Output *S*: solution
1: sortPopulationAscending(Pop) //sort the population ascendingly
2: f = computeFitness() //compute fitness
3: sf = sumFitness(f) //Sum of all fitness value
4: Pop′ ← Pop;
5: **while** *(Pop′ < 2. Pop)* **do**
6: sf′ ← sf.random[0..1] // Select a fitness value randomly
7: index1 ← 0, sum ← f0 //Get the first fitness value
8: **while** *(sum < sf′)* **do**
9: index1 ← index1 + 1
10: sum ← sum + findex1
11: **end while**
12: sf′ ← sf.random[0..1] //Select a fitness value randomly
13: index2 ← 0, sum = f0 //Get the first fitness value
14: **while** *(sum < sf′)* do
15: index2 ← index2 + 1
16: sum ← sum + findex2
17: **end while**
18: z ← get customer size
19: cutpoint1 ← random[0..z].1
20: cutpoint2 ← random[0..z].1
21: **while** *(cutpoint1 == cutpoint2)* **do**
22: cutPoint2 ← random[0..z]
23: **end while**
24: offspring1 ← OX(pindex1, pindex2, cutpoint1, cutpoint2)
25: offspring2 ← OX(pindex2, pindex1, cutpoint1, cutpoint2)
26: Pop′i ← offspring1
27: i ← i + 1
28: Pop′i + 1 ← offspring2
29: i ← i + 1
30: **end while**
31: Pop′ ← mutatePopulation(Pop′) //mutation function
32: Pop′ ← sortPopulationAscending(Pop′) // sorted based on lowest total travelled distance
33: **for** *i ← 1 to size* ***of*** *Pop* **do**
34: Pop i ← Pop′i
35: **end for**
36: updateBestSolution(Pop) //Replace the best solution if the individual in population performs better
37: **return** *S*

For generating the second offspring, the process is repeated by reversing the parents.

4) MUTATION

Algorithm 5 uses a swap mutation technique [32]. Each individual in the population is loop through (line 3). This

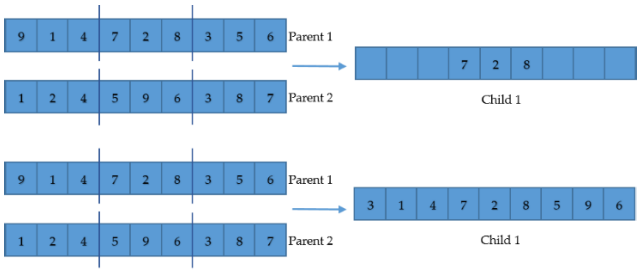

FIGURE 4. Order crossover.

individual is subjected to swap mutation if the random value is less than the mutation ratio (line 4). If the mutation ratio is high, the individuals in the population will have a high likelihood to be selected for mutation and more areas in the search space will be explored, but this may prevent the population from converging to any optimum solution. However, if the mutation ratio is low, there will be fewer individuals in the population to be selected for mutation and may result in local optimal. If the random positions of the genes in the individual are the same, then the selection of the random genes will be repeated (lines 8 to 11). If both genes positions are different, then these genes will interchange with one another (line 12).

The swapped genes in the individual will call the type 2 strategy (lines 13 to 14) to generate a new solution which is shown in Fig 5.

FIGURE 5. Swap mutation.

C. RUIN AND RECREATE PRINCIPLE

The RR principle consists of a ruin method, which continuously destroys a part of the same solution and the recreate method to reconstruct the solution. This process avoids local optimum and can obtain high-quality results. The ruin and recreate procedure is performed on an earlier GA generated solution, and this phenomenon leads to a faster execution time than a population-based algorithm. The RR principle is considered a metaheuristic algorithm as there are several ways to ruin a solution. In this article, we suggest two strategies (radial and random) in the ruin method and two strategies (best insertion and regret insertion) in the recreate method.

1) RADIAL RUIN

In the radial ruin method [18], the customers in the radial deletion area are marked for deletion, which is shown in Fig 6.

Algorithm 6 shows the radial ruin strategy. Initially, the number of customers to be removed from the solution is calculated based on the radial ratio (line 1). The higher the radial ratio, the more the customers will be removed from the solution and vice versa. Next, a random customer is selected from the solution (line 2) and removed from the solution (line 3). Based on the random customer, the customers in the neighbourhood list are retrieved (line 4). Each customer is enumerated from the neighbourhood list and removed from the solution as long as it does not exceed the calculated customer removal number (lines 5 to 8). The removed customer

Algorithm 4 OX

Input *Parent1:* first parent, *Parent2:* second parent, *Cutoffpoint1:* first cutoff point, *Cutoffpoint2:* second cutoff point
Output *offspring*: offspring

```
1:  tempindex ← 0
2:  index ← Cutoffpoint2 + 1
3:  if (index == Parent1.length) then
4:   for i ← 0 to Parent1.length do
5:    outerSegmentBuildArrayi = Parent1i
6:   end for
7:  else
8:   for index ← Cutoffpoint2 + 1 to Parent1.length do
9:   outerSegmentBuildArraytempindex ← Parent1index
10:    tempindex ← tempindex + 1
11:   end for
12:   for index ← 0 to Cutoffpoint2 do
13:    outerSegmentBuildArraytempindex ←
Parent1index
14:    tempindex ← tempindex + 1
15:   end for
16:  end if
17:  for index ← Cutoffpoint1 to Cutoffpoint2 do
18:   for index ← 0 to outerSegmentBuildArray.length
do
19:   if (outerSegmentBuildArrayindex == Parent2index)
then
20:     remove(outerSegmentBuildArrayindex)
21:    end if
22:   end for
23:  end for
24:  for index ← Cutoffpoint1 to Cutoffpoint2 do
25:   offspringindex ← Parent2index
26:  end for
27:  tempIndex ← 0
28:  for y ← Cutoffpoint2 + 1 to length(offspring) do
29:   if (y == length(offspring))
30:    break
31:   end if
32:   Offspringy ← outerSegmentBuildArraytempIndex
33:   tempIndex ← tempIndex + 1
34:  end for
35:  for z ← 0 to Cutoffpoint1 do
36:   if (z == length(offspring))  then
37:    exit for loop
38:   end if
39:   offspringz ← outerSegmentBuildArraytempIndex
40:   tempIndex ← tempIndex + 1
41:  end for
42:  offspring ← radialruin(getroutes(offspring))
43:  offspring ← bestinsertion(getroutes(offspring))
44:  return offspring
```

TABLE 1. Instance type.

Instance Type	No. of instances	No. of customers	No. of Vehicles	Vehicle Capacity	Type of customer distribution	Size of time windows
C1	9	100	25	200	C	Small
C2	8	100	25	700	C	Large
R1	12	100	25	200	R	Small
R2	11	100	25	1000	R	Large
RC1	8	100	25	200	RC	Small
RC2	8	100	25	1000	RC	Large

C is the clustered customers, R is the randomly distributed customers, and RC is a random clustered customers.

Algorithm 5 Mutate Population

```
1:  Input Pop: Population, MR: Mutation Ratio
2:  Output Pop′: Population
3:  for i ← 0 to Pop.length do
4:  if (random[0..1].1 < MR) then
5:    Q ← getCustomers(Popi)
6:     Index1 ← random [0.. Pop.length].1
7:     Index2 ← random [0.. Pop.length].1
8:     while (Index1 == Index2) do
9:        Index1 ← random [0.. Pop.length].1
10:       Index2 ← random [0.. Pop.length].1
11:    end while
12:    swap(Q, index1, index2) // swap the position of
customer
13:    RC ← radialruin(Q) // RC − removed customers
14:    Pop′i ← bestinsertion(Pop′i, RC)
15:   end if
16:  end for
17:  return Pop′
```

Algorithm 6 Radial Ruin

```
Input R: Routes
Output R routes, RC: removedCustomers
1:  noOfCustomersBeRemoved ← getNumberofCustomer
BeRemoved() //Depend on the radial ratio be set
2:   targetCustomer ← random[1..customerSize].1
3:   removedCustomers ← removeCustomer
(targetCustomer, R)
4:  neighborhoodList ← neighbourhood[targetCustomer]
5:   foreach customer in neighbourhoodList do
6:    if (noOfCustomersBeRemoved == i) then
7:      break;
8:    end if
9:    if (R ← removeCustomer(customer, R)) then
10:     RC ← RC + customer
11:   end if
12:   i = i + 1
13:  end for
14:  return  R, RC
```

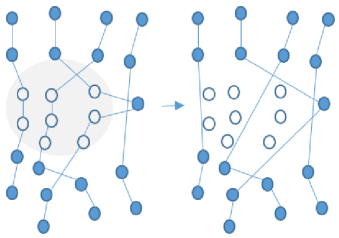

FIGURE 6. Radial ruin.

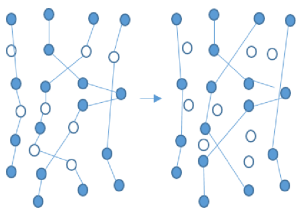

FIGURE 7. Random ruin.

is added to a removed list (line 10), which will be used for reconstruction later. Radial ruin strategy tends to be more local.

2) RANDOM RUIN

In the random ruin method [18], the customers are randomly marked for deletion, as shown in Fig 7. In this strategy (algorithm 7), the number of customers to be removed is randomly calculated from the solution (line 1). Next, the customers are shuffled in the list (line 2). Each customer is removed from the solution as long as it does not exceed the number of customer removals (lines 4 to 6). The removed customer is added to a removed list (line 8), which will be used for later reconstruction. This strategy tends to be global.

3) BEST INSERTION

The best insertion strategy [18] is based on each customer that selects from the list randomly before performing the best insertion. Each customer is evaluated with each vehicle and will choose the route if it has the minimum cost of insertion. However, if the customer cannot be inserted into the route, an additional vehicle and a new route will be added into the solution.

Algorithm 8 shows each customer in the removal list is enumerated and reinserted into each route (lines 1 to 13). Each insertion cost is calculated in each route to determine which route has the best insertion cost. The route with the best insertion cost will be selected. Lines 14 to 20 explain that a new route is created with the customer and add a new route to the existing routes if it has a better insertion cost. However, the customer will not be able to add to the route if the best insertion cannot be found. In this case, this customer will be added to an unassigned customer pool (lines 21 to 25). If the

Algorithm 7 Random Ruin

Input S: solution
Output R routes, RC: removed Customers
1: *noOfCustomersBeRemoved* ← *getNumberofCustomer BeRemoved()*
2: *availableCustomers* ← *shuffle(getAllCustomers(S))*
3: *removed* = 0
4: **foreach** *cust* **in** *availableCustomers* **do**
5: **if** *removed* == *noOfCustomersBeRemoved* **exit for**
6: $S \leftarrow removedCustomer(cust, S)$
7: $RC \leftarrow RC + cust$
8: $removed \leftarrow removed + 1$
9: **end if**
10: **end for**
11: $R \leftarrow getroutes(S')$
12: **return** *R, RC*

TABLE 2. Parameters.

Parameter	Value
Population Size	750
Mutation Ratio	0.1
Number of Runs	30
Generation	60
Ruin and recreate Iteration	50000
Radial Ruin fraction	0.3
Ruin Random fraction	0.5
Neighbourhood fraction	0.5

TABLE 3. Parameters.

Strategy	Ruin method	Recreate method
Type 1		Regret insertion strategy
Type 2	Radial strategy	Best insertion strategy
Type 3	Radial or random strategy	Best insertion strategy
Type 4	Random strategy	Best insertion strategy

best insertion can be found, this customer will be added to the route (line 27). Line 30 explains that a new solution is created based on the created routes.

Lines 1 to 26 explain the best Insertion score of each customer according to the cost insertion calculation. The customer with the lowest cost will be selected for reinsertion. A new solution is generated due to this process. Lines 21 to 24 explain that customers who cannot make it for reinsertion due to constraints violation will be reported as unassigned customers.

4) REGRET INSERTION

Regret Insertion algorithm is used to initialize routes in VRPTW by Potvin and Rousseau [36]. In algorithm 9, it is based on inserting the best possible route for each customer, how much the ''regret'' of the first and second best customer insertion cost is calculated, and the score difference between the two insertion costs is compared. If the second best cost is higher than the first score, the customer will be inserted into the route immediately. If it is not, the customer will be inserted later. Lines 2 to 15 explain each customer, traverse all other customers, and calculate which one has the best score or the shortest distance. The best score is added to the route, and finally, it returns a solution (line 16).

Algorithm 8 Best Insertion

Input R: Routes, C: Removed Customers
Output S': New Solution
1: **foreach** *customer* **in** C **do**
2: *bestInsertion* ← *null*
3: *bestInsertioncost* ← *MAX_VALUE*
4: **foreach** *route* **in** *newRoutes* **do**
5: *insertionData* ← *bestInsertionCostCalculator .getInsertionData (route, customer, bestInsertioncost)*
6: **If** *(insertionData cannot insert)* **then**
7: *continue*
8: **end if**
9: **if** *(insertionData.getInsertionCost() < bestInsertion-cost)* **then**
10: *bestInsertion* ← *getInsertion(route, insertionData)*
11: *bestInsertioncost* ← *insertionData .getInsertioncost()*
12: **end if**
13: **end for**
14: *newRoute* ← *createEmptyRoute()*
15: *newInsertionData* ← *bestInsertionCostCalculator .getInsertionData (newRoute, customer, bestInsertioncost)*
16: **if** *(newInsertionData can insert into existing routes)* **then**
17: **if** *(newInsertionData.getInsertionCost() < bestIn-sertioncost)* **then**
18: *newRoutes* ← *newRoutes + newRoute*
19: *bestInsertion* ← *generateInsertion(newRoute, newInsertionData)*
20: **end if**
21: **else**
/* *Get failed constraints* */
22: *empty* ← *empty + getFailedConstraintNames()*
23: **end if**
24: **if** *bestinsertion == null* **then**
25: *unassignedcustomers* ← *unassignedcustomers + customer*
26: **else**
27: *insertCustomer(unassignedCustomer, bestInser-tion.getInsertionData, bestInsertion.getRoute())*
28: **end if**
29: **end for**
30: S' ← *createNewSolution(routes, unassignedCus-tomers, empty)*
31: **return** S'

IV. RESULTS

A. EXPERIMENT SETUP

In this experiment, the Solomon [16] instances are used for benchmarking. This benchmarking is to test algorithm vulnerability and effectiveness. These instances are grouped into C1, C2, R1, R2, RC1, and RC2, respectively. Each group of instances is subdivided into two types. The first type is the clustered (C), random (R), and a combination of random

TABLE 4. Variants.

Variant	RRGA phase					RR phase
	GA					
	Generate first solution	Initialize population	Selection	crossover	Mutation	
1	Type 1	Type 4		Type 4	Type 4	Type 4
2	Type 1	Type 2		Type 2	Type 2	Type 2
3						Type 3
4						Type 4
5						Type 2

Algorithm 9 Regret Insertion

Input, C: Customers
Output S': New Solution
routes ← *null*
while there *is still customer* ***in*** C **do**
  for each *customer* **in** C **do**
    bestScoredCustomer = *calculateCustomerScore(customer, C)*
    if *(bestScoredCustomer! = null)* **then**
      if *(bestScoredCustomer.isNewRoute())* **then**
        routes.add(bestScoredCustomer.getRoute())
      end if
      insertCustomer(bestScoredCustomer .getCustomer(),
        bestScoredCustomer.getInsertionData(),
        bestScoredCustomer.getRoute())
      removeCustomer(customer, C)
    end if
  end for
end while
S' ← *createNewSolution(routes, unassignedcustomers, empty)*
return S'

(R) and clustered (C) in the group, which shows the extent of the customers disparate. The second type indicates the time window and vehicle capacity size. The C1, R1, and RC1 have a smaller time window size and vehicle capacity, which can serve a small number of customers. In contrast, the C2, R2, and RC2 instance types can serve a large number of customers as it has a broader time window size and vehicle capacity. Table 1 summarizes the detail of the instances types. This proposed algorithm is developed using Java version 1.8 and runs on a desktop machine with an Intel Core i5 CPU, 2.6GHz/8G of RAM. It can be run on Window, Macintosh or Linux operating system. The parameters used are shown in Table 2.

B. RESULT ANALYSIS

To prove our proposed algorithm is better than other algorithms, we benchmark our results against four different comparisons. They are five suggested variants, four recently published hybrid algorithms, the best-known solutions, and nine leading algorithms.

1) COMPARISON WITH OTHER VARIANTS

Table 3 shows different strategies type used for constructing different variants. Each strategy type may consist of either or both ruin and recreate methods. Type 1 strategy only consists of regret insertion strategy in the recreate method. There is no strategy adopted in the ruin method. Type 2 strategy is formed by combining radial strategy in the ruin method and best insertion strategy in the recreate method. Type 3 strategy is selecting radial or random strategy randomly in the ruin method with the best insertion strategy in the recreate method, and type 4 strategy consists of random strategy in the ruin method with the best insertion method in the recreate method. Table 4 shows five different variants that are used for comparison with RRGA. Each variant may involve either or both RRGA and RR phases. During RRGA phase, the GA is used. The GA has five procedures which are performed in sequence. These procedures are the first solution generation, population initialization, selection, crossover, and mutation. Each of these procedures will call one of the strategy types. Variant 1 consists of both RRGA and RR phase. During RRGA phase, first solution generation, population initialization, crossover, and mutation will use types 1, 4, 4, and 4 strategies, respectively. In the RR phase, the type 4 strategy is used. Variant 2 has RRGA and RR phase. In the RRGA phase, types 1, 2, 2, and 2 strategies are used in first solution generation, population initialization, cross over and mutation, respectively. Variant 3, 4, and 5 have the RR phase only. The types 3, 4, and 2 strategies are used in variant 3, 4, and 5 accordingly. Table 5 compares the RRGA with five variants. In the table, ''NV'' represents the minimum number of used vehicles, ''TD'' represents the minimum total travelled distance (function cost) and ''T'' represents the minimum operation time of the proposed algorithm in seconds. The ''count'' represents the number of minimum total travelled distance in the instances. The ''Average'' represents the average of each metric for each instance type. The ''Overall Average'' represents the average of each metric for all instances. The best result or minimum value of total travelled distance in each instance is obtained after 30 independent runs.

In C1 and C2 instances, the RRGA best results are similar to other variants except for variant 2 in C109 instance. In C1 instance type, variant 1 has the minimum average operation time, and except for variant 2, all variants and the RRGA have the minimum average total travelled distance.

TABLE 5. Comparison with different variants.

Instance	RRGA			Variant 1			Variant 2		
	NV	TD	T	NV	TD	T	NV	TD	T
C101	10	**828.94**	0	10	**828.94**	0	10	**828.94**	0
C102	10	**828.94**	0	10	**828.94**	0	10	**828.94**	1.28
C103	10	**828.07**	3.27	10	**828.07**	0	10	**828.07**	22.36
C104	10	**824.78**	20.4	10	**824.78**	2.73	10	**824.78**	143.22
C105	10	**828.94**	0	10	**828.94**	0	10	**828.94**	0
C106	10	**828.94**	0	10	**828.94**	0	10	**828.94**	0
C107	10	**828.94**	0	10	**828.94**	0	10	**828.94**	0.56
C108	10	**828.94**	0.15	10	**828.94**	0	10	**828.94**	5.35
C109	10	**828.94**	0.11	10	**828.94**	0.28	10	837.58	39.22
Average	10.00	**828.38**	2.66	10.00	**828.38**	0.33	10.00	829.34	23.55
C201	3	**591.56**	0	3	**591.56**	0	3	**591.56**	0
C202	3	**591.56**	0	3	**591.56**	0	3	**591.56**	0
C203	3	**591.17**	0	3	**591.17**	0	3	**591.17**	0
C204	3	**590.60**	0	3	**590.6**	0.3	3	**590.6**	0
C205	3	**588.88**	0	3	**588.88**	0	3	**588.88**	0
C206	3	**588.49**	0	3	**588.49**	0	3	**588.49**	0
C207	3	**588.29**	0	3	**588.29**	0	3	**588.29**	0
C208	3	**588.32**	0	3	**588.32**	0	3	**588.32**	0
Average	3.00	**589.86**	**0.00**	3.00	**589.86**	0.04	3.00	**589.86**	**0.00**
R101	20	**1642.88**	429.15	20	1647.58	1247.67	20	**1642.88**	365.85
R102	18	**1473.62**	74.6	18	1473.73	216.18	18	**1473.62**	44.82
R103	14	**1213.62**	49.3	14	**1213.62**	432.35	14	**1213.62**	9.68
R104	11	**976.61**	30.42	11	**976.61**	254.27	11	977.55	1.32
R105	15	**1360.78**	70.34	15	**1360.78**	40.22	15	**1360.78**	20.39
R106	13	**1239.37**	75.61	13	**1239.37**	739.89	13	**1239.37**	333.25
R107	11	**1072.12**	436.19	11	1076.44	473.51	11	1073.6	43.81
R108	10	951.22	61.25	10	944.44	56.96	10	949.04	1.22
R109	13	**1151.84**	240.13	13	**1151.84**	181.24	13	**1151.84**	4.6
R110	12	**1072.42**	143.41	12	1080.24	335.95	12	1080.04	83.03
R111	12	**1053.50**	13.92	12	**1053.5**	65.82	12	**1053.5**	0.14
R112	10	**959.97**	158.6	10	968.23	118.24	10	964.78	182.03
Average	13.25	1180.66	148.58	13.25	1182.20	346.86	13.25	1181.72	**90.85**
R201	8	**1147.80**	0.8	8	1148.09	124.44	8	**1147.8**	1.4
R202	8	**1034.35**	0.01	5	1045	17.46	7	1038.53	1.79
R203	6	**874.87**	1.34	6	**874.87**	3.05	6	**874.87**	6.87
R204	5	**735.80**	1.3	4	737.06	8.64	5	**735.8**	5.77
R205	5	**954.16**	0.68	5	**954.16**	0.36	5	**954.16**	0.17
R206	5	**884.85**	1.92	5	892.08	0.38	5	885.18	2.78
R207	4	**797.99**	9.06	4	798	1.4	4	**797.99**	6.58
R208	4	**705.33**	4.36	4	712.67	1.23	4	**705.33**	0.74
R209	5	860.46	3.57	5	860.46	2.7	5	860.46	1.48
R210	6	**904.78**	1.29	5	915.43	5.26	6	**904.78**	0.27
R211	4	764.69	1.5	4	763.93	4.05	4	765	0.03
Average	5.45	878.64	**2.35**	5.00	881.98	15.36	5.36	879.08	2.53
RC101	15	**1623.59**	13.13	15	1636.97	485.79	15	1629.75	10.33
RC102	14	**1461.23**	87.53	15	1482.77	628.26	14	**1461.23**	211.83
RC103	11	1262.98	400.06	12	1288.34	1431.03	11	**1262.02**	70.87
RC104	10	**1135.48**	106.86	11	1161.16	599.46	10	1136.27	476.88
RC105	16	**1518.58**	680.82	16	1518.77	1138.43	16	1518.6	446.98
RC106	13	**1377.35**	464.35	13	1396.27	2135.57	13	1379.22	446.14
RC107	12	1212.83	120.01	13	1245.74	1121.01	12	1228.33	140.66
RC108	11	1140.73	104.89	11	1140.73	228.51	14	1213.62	41.65
Average	12.75	1341.60	247.21	13.25	1358.84	971.01	13.13	1353.63	230.67
RC201	9	1266.11	0.78	8	1267.88	1.26	9	1266.11	5.2
RC202	8	**1095.64**	1.81	8	1098.81	1.32	8	**1095.64**	2.35
RC203	5	**926.82**	6.36	5	**926.82**	8.69	5	**926.82**	2.88
RC204	4	788.66	2.01	4	788.66	0.33	4	788.66	2.2
RC205	7	**1157.55**	0.74	7	**1157.55**	0.18	7	**1157.55**	0.1
RC206	7	**1054.61**	0.41	5	1063.53	0.56	5	1077.09	0.4
RC207	6	**966.08**	0.02	6	971.21	6.38	6	971.21	6.49
RC208	4	779.31	0.49	4	780.72	0.44	4	**778.93**	0.57
Average	6.25	**1004.35**	**1.58**	5.88	1006.90	2.40	6.00	1007.75	2.52
Overall Average	8.66	**978.12**	**68.27**	8.59	981.94	216.46	8.66	980.79	57.06
Count		**47**			27			36	

"TD" represents the total travelled distance. "NV" represents the number of used vehicles. "T" represents the minimum operation time of the proposed algorithm in seconds.

TABLE 5. *(Continued.)* **Comparison with different variants.**

Instance	Variant 3			Variant 4			Variant 5		
	NV	TD	T	NV	TD	T	NV	TD	T
C101	10	**828.94**	0	10	**828.94**	0	10	**828.94**	0.08
C102	10	**828.94**	0.17	10	**828.94**	0.15	10	**828.94**	1.49
C103	10	**828.07**	1.42	10	**828.07**	1.29	10	**828.07**	25.2
C104	10	**824.78**	6.62	10	**824.78**	5.94	10	**824.78**	386.65
C105	10	**828.94**	0.05	10	**828.94**	0.15	10	**828.94**	0.98
C106	10	**828.94**	0.18	10	**828.94**	0.14	10	**828.94**	1.49
C107	10	**828.94**	0.23	10	**828.94**	0.14	10	**828.94**	6.43
C108	10	**828.94**	0.28	10	**828.94**	0.21	10	**828.94**	47.06
C109	10	**828.94**	1.69	10	**828.94**	1.79	10	**828.94**	365.46
Average	10.00	**828.38**	1.18	10.00	**828.38**	1.09	10.00	**828.38**	92.76
C201	3	**591.56**	0	3	**591.56**	0	3	**591.56**	0
C202	3	**591.56**	0	3	**591.56**	0	3	**591.56**	0
C203	3	**591.17**	0.01	3	**591.17**	0.01	3	**591.17**	0.04
C204	3	**590.6**	0.37	3	**590.6**	0.23	3	**590.6**	0.23
C205	3	**588.88**	0.18	3	**588.88**	0.03	3	**588.88**	0.16
C206	3	**588.49**	0.14	3	**588.49**	0.03	3	**588.49**	0.42
C207	3	**588.29**	0.15	3	**588.29**	0.14	3	**588.29**	0
C208	3	**588.32**	0.13	3	**588.32**	0.08	3	**588.32**	0.41
Average	3.00	**589.86**	0.12	3.00	**589.86**	0.07	3.00	**589.86**	0.16
R101	20	**1642.88**	316.99	20	**1642.88**	166.46	20	**1642.88**	368.67
R102	18	1473.07	88.57	18	**1473.62**	519.93	18	1472.82	147.34
R103	14	**1213.62**	1138.4	14	**1213.62**	868.33	14	**1213.62**	116.4
R104	11	**976.61**	136.1	11	**976.61**	90.12	11	**976.61**	304.4
R105	15	**1360.78**	69.84	15	1361.23	487.42	15	1361.64	86.82
R106	13	**1239.37**	83.14	13	**1239.37**	510.88	13	**1239.37**	165.5
R107	11	**1074.74**	332.95	11	**1075.51**	98.15	11	**1072.92**	460.51
R108	10	**938.2**	85.65	10	**948.57**	179.82	10	**947.37**	221.67
R109	13	**1151.84**	168.4	13	**1151.84**	229.14	13	**1154.15**	193.18
R110	12	**1072.42**	205.78	12	**1072.42**	457.12	12	**1078.77**	159.92
R111	12	**1053.5**	51.06	12	**1053.5**	304.17	12	**1053.5**	642.21
R112	10	**962.87**	279.89	10	**962.7**	201.54	10	**974.29**	159.17
Average	13.25	**1179.99**	246.40	13.25	**1180.99**	342.76	13.25	**1182.33**	252.15
R201	8	**1147.8**	8.22	8	**1147.8**	37.81	8	**1147.8**	2.5
R202	6	**1034.97**	8.04	7	**1038.53**	14.74	7	**1037.23**	32.49
R203	6	**874.87**	5.03	6	**880.73**	30.85	6	**874.87**	27.81
R204	5	**735.8**	2.72	5	**735.8**	3.68	5	**735.8**	2.21
R205	5	**954.16**	2.29	5	**954.16**	3.43	5	**954.16**	3.69
R206	5	**884.85**	3.37	5	**884.85**	7.65	5	**884.85**	15.95
R207	4	**797.99**	15.96	4	**797.99**	23.56	4	**797.99**	14.77
R208	4	**705.33**	6.6	3	**706.86**	20.16	4	**705.33**	10.37
R209	5	**860.11**	4.38	5	**860.11**	10.57	5	**860.11**	4.96
R210	6	**905.21**	26.79	6	**910.93**	58.52	6	**910.15**	13.41
R211	4	**755.95**	9.89	4	**753.15**	93.8	4	**755.96**	19.37
Average	5.27	**877.91**	8.48	5.27	**879.17**	27.71	5.36	**878.57**	13.41
RC101	15	**1623.59**	86.16	16	**1629.96**	569.5	15	**1623.59**	260.32
RC102	14	**1461.23**	72.01	14	**1461.5**	848.1	14	**1461.54**	359.58
RC103	11	**1262.37**	173.61	11	**1275.49**	2349.37	12	**1295.62**	240.09
RC104	10	**1135.8**	182.26	11	**1150.84**	942.67	10	**1135.52**	299.87
RC105	16	**1518.58**	107.47	16	**1518.75**	830.17	16	**1518.58**	643.8
RC106	13	**1377.35**	450.67	13	**1378.33**	1266.6	13	**1383.99**	373.33
RC107	12	**1212.83**	217.7	12	**1213.78**	2231.16	12	**1218.2**	124.77
RC108	11	**1132.24**	78.94	11	**1134.85**	421.9	11	**1134.75**	121.58
Average	12.75	**1340.50**	171.10	13.00	**1345.44**	1182.43	12.88	**1346.47**	302.92
RC201	9	**1266.11**	56.43	9	**1265.56**	12.92	9	**1265.56**	7.35
RC202	8	**1095.64**	1.81	8	**1095.64**	4.06	8	**1095.64**	4.84
RC203	5	**926.82**	5.52	5	**932.7**	14.12	5	**926.82**	5.92
RC204	4	**796.55**	5.26	4	**786.39**	25.1	4	**788.66**	0.99
RC205	7	**1157.55**	17.99	7	**1157.55**	15.26	7	**1157.55**	17.13
RC206	5	**1063.53**	3.33	7	**1054.61**	6.15	7	**1054.61**	2.92
RC207	5	**970.78**	3.31	6	**974.94**	20.91	6	**971.21**	11.4
RC208	4	**778.93**	2.29	4	**780.72**	7.57	4	**778.93**	4.4
Average	5.88	**1006.99**	11.99	6.25	**1006.01**	13.26	6.25	**1004.87**	6.87
Overall Average	8.57	**978.06**	80.83	8.66	**979.08**	249.89	8.66	**979.24**	115.85
Count		43			37			**39**	

"TD" represents the total travelled distance. "NV" represents the number of used vehicles. "T" represents the minimum operation time of the proposed algorithm in seconds.

TABLE 6. Comparison with different variants by average.

Instances	RRGA			Variant 1			Variant 2		
	MTD	Std Dev	AT	MTD	Std Dev	AT	MTD	Std Dev	AT
C101	**828.94**	0	29.42	**828.94**	0	249.82	**828.94**	0	57.64
C102	**828.94**	0	0.89	**828.94**	0	28.68	**828.94**	0	24.34
C103	**828.07**	0	20.49	**828.07**	0	208.59	**828.7**	1.44	1019.98
C104	**824.78**	0	408.64	825.59	4.46	1.12	829.17	6.92	4052.68
C105	**828.94**	0	1.3	**828.94**	0	0.62	**828.94**	0	24.94
C106	**828.94**	0	1.38	**828.94**	0	0.67	**828.94**	0	175.56
C107	**828.94**	0	2.63	**828.94**	0	0.9	**828.94**	0	61.78
C108	**828.94**	0	3.43	**828.94**	0	19.81	829.07	0.73	2171.66
C109	**828.94**	0	1.84	**828.94**	0	82.49	864.4	23.24	1979.81
Average	**828.38**	**0**	52.22	828.47	0.5	249.82	832.89	3.59	1063.15
C201	**591.56**	0	0	**591.56**	0	0	**591.56**	0	0
C202	**591.56**	0	0	**591.56**	0	0	**591.56**	0	0.28
C203	**591.17**	0	0.32	**591.17**	0	0.09	**591.17**	0	0.93
C204	**590.6**	0	45.31	**590.6**	0	2.15	**590.6**	0	23.75
C205	**588.88**	0	0.22	**588.88**	0	32.41	**588.88**	0	0.69
C206	**588.49**	0	0.7	**588.49**	0	0.8	**588.49**	0	0.42
C207	**588.29**	0	0.38	**588.29**	0	0.53	**588.29**	0	0.43
C208	**588.32**	0	0.44	**588.32**	0	0.72	**588.32**	0	1.38
Average	**589.86**	**0**	5.92	**589.86**	**0**	4.59	**589.86**	**0**	3.49
R101	**1644.61**	3.21	2531.3	1647.58	7.47	2199.55	1645.52	3.46	2420.98
R102	1474.15	0.59	1226.71	1475.16	1.39	2137.57	**1473.9**	0.16	1500.33
R103	**1218.86**	3.73	2071.11	1220.13	5.56	6634.68	1220.09	2.81	2648.82
R104	**985.39**	8.73	1922.35	987.57	13.54	5586.91	982.78	5.6	1310.45
R105	1368.65	7.91	1732.99	**1367.59**	4.23	4408.77	1369.02	6.53	2904.15
R106	1244.42	5.58	1533.63	1245.89	7.21	6023.68	**1244.07**	5.61	2374.65
R107	1078.82	4.81	4513.96	1085.91	8.78	6972.75	**1078.28**	5.56	2460.95
R108	**952.72**	1.99	1922.89	954.73	6.4	3070.6	954.95	6.96	2205.36
R109	1156.43	5.41	3265.27	1157.21	10.1	5126.13	**1155.68**	9.05	2609.87
R110	**1086.88**	12.6	3617.43	1094.58	8.89	3536.49	1092.69	13.19	4140.04
R111	1058.89	6.76	2281.61	1057.91	4.35	2194.64	**1055.77**	5.38	1594.37
R112	**978.2**	10.81	4180.14	974.7	5.71	2599.81	982.56	15.63	4508.75
Average	**1187.34**	**6.01**	2566.62	1189.08	6.97	4207.63	1187.94	6.66	2556.56
R201	**1148.49**	1.68	394.69	1150.7	4.11	1340.91	1150.48	5.1	1903
R202	**1038.6**	2.97	1464.29	1047.17	2.36	508.33	1040.68	2.74	1358.48
R203	875.22	1.34	416.15	876.46	2.93	293.31	**875.03**	0.9	849.44
R204	736.61	2.89	337.02	737.06	0	764.8	**735.95**	0.59	492.49
R205	955.57	2.44	252.46	954.89	2.78	1064.99	**954.16**	0	734.9
R206	**886.4**	2.8	172.34	892.08	0	397.73	887.89	9.32	566.27
R207	**797.99**	0	250.61	800.09	2.61	463.74	800.18	5.78	541.86
R208	711.74	6.79	463.74	714.87	0.96	819.1	**707.15**	4.52	821.72
R209	861.15	1.86	343.05	864.27	5.96	554.99	**860.78**	1	487.37
R210	910.74	4.8	611.05	915.68	0.64	1250.87	**906.29**	3.09	776.49
R211	**764.77**	0.32	522.79	764.94	0.88	629.18	772.1	8.32	694.8
Average	**880.66**	2.54	**475.29**	883.47	**2.11**	735.27	880.97	3.76	838.8
RC101	**1629.38**	6.91	1710.98	1644.26	8.44	1765.5	1635.91	4.82	1695.61
RC102	**1478.07**	8.8	2576.63	1485.24	3.35	1135.65	1473.95	12.23	3706.2
RC103	1289.43	20.76	2544.98	1316.69	22.67	2766.04	**1286.12**	24.1	3403.92
RC104	1152.33	14.33	2117.79	1176.34	9.75	1789.2	**1149.54**	15.11	3971.67
RC105	1529.55	14.73	3073.05	1538.68	18.34	1676.72	**1522.46**	6.2	4518.92
RC106	1390.18	11.91	2958.42	1405.91	7.12	2705.52	**1390**	13.06	3782.18
RC107	**1232.84**	17.33	2636.22	1260.57	11.12	1705.62	1241.88	15.08	2495.2
RC108	**1140.13**	4.99	1944.13	1155.28	20.51	1034.63	1220.13	5.56	1962.99
Average	**1355.24**	12.47	2445.28	1372.87	12.66	**1822.36**	1365	12.02	3192.09
RC201	1270.86	5.65	340.22	**1268.76**	3.31	787.13	1271.13	6.73	1745.44
RC202	1098.23	3.85	139.33	1103.46	3.6	198.95	**1096.31**	2.09	1255.21
RC203	**926.82**	0	170.68	928.04	5.7	861.38	**926.82**	0	454.32
RC204	796	3.89	229.19	**793.25**	4.55	1097.71	795.11	5.01	1725.81
RC205	**1157.59**	0.06	299.51	1157.61	0.06	645.29	1158.73	2.99	549.3
RC206	**1059.14**	7.38	187.26	1065.19	2.96	819.27	1079.8	6.93	931.7
RC207	980.42	11.4	415.68	978.04	4.33	1593.64	**977.2**	9.57	444.38
RC208	787.17	15.89	910.95	781	1.58	458.86	**779.88**	2.22	1741.22
Average	1009.53	6.02	1452.96	**1009.42**	**3.26**	807.78	1010.62	4.44	1105.92
Overall Average	**982.64**	4.43	**1050**	986.08	**4.26**	1433.04	985.11	5.1	1497.96
Count	**5**	**3**	1	2	**3**	1	1	1	0

TABLE 6. *(Continued.)* **Comparison with different variants by average.**

Instances	Variant 3			Variant 4			Variant 5		
	MTD	Std Dev	AT	MTD	Std Dev	AT	MTD	Std Dev	AT
C101	**828.94**	0	1.11	**828.94**	0	3.26	**828.94**	0	1.02
C102	**828.94**	0	1.04	**828.94**	0	1.15	**828.94**	0	8.6
C103	**828.07**	0	16.64	**828.07**	0	34.66	828.57	1.32	963.64
C104	**824.78**	0	240.32	825.59	4.47	95.3	834.34	10.61	2176
C105	**828.94**	0	1.32	**828.94**	0	1.03	**828.94**	0	10.87
C106	**828.94**	0	0.56	**828.94**	0	0.71	**828.94**	0	32.58
C107	**828.94**	0	1.07	**828.94**	0	0.97	**828.94**	0	75.08
C108	**828.94**	0	3.66	**828.94**	0	1.08	828.96	0.14	929.18
C109	**828.94**	0	27.44	**828.94**	0	23.87	879.39	26.51	1999.1
Average	**828.38**	**0**	32.57	828.47	0.5	**18**	835.11	4.29	688.45
C201	**591.56**	0	0	**591.56**	0	0	591.56	0	0
C202	**591.56**	0	0	**591.56**	0	0.34	591.56	0	0.15
C203	**591.17**	0	0.78	**591.17**	0	83.15	591.17	0	0.86
C204	**590.6**	0	0.97	**590.6**	0	1.31	592.67	11.33	3.11
C205	**588.88**	0	0.88	**588.88**	0	128.15	588.88	0	0.81
R101	1645.52	3.98	2434.5	1,646.87	5.9	4873.91	1645.28	2.72	1903.67
R102	1473.99	0.41	1350.73	1,475.64	3.96	2830.2	1473.96	0.5	2910.86
R103	1220.26	3.09	2444.51	1,219.76	3.01	4683.71	1220.32	3.26	1781.58
R104	988.11	7.25	2318.11	1,000.47	7.9	4330.8	993.37	11.18	2117.07
R105	1371.26	8.09	2129.85	1,370.28	6.27	4177.92	1373.41	7.92	2464.4
R106	1247.03	7.83	2090.61	1,247.05	8.02	5617	1247.75	6.25	2262.14
R107	1083.37	7.22	2448.32	1,084.60	5.89	4146.09	1082.77	7.25	2521.95
R108	955.81	7.15	1738.69	959.01	8.05	3354.01	962.87	8.99	1871.26
R109	1156.41	7.49	2100.55	1,157.71	7.89	4784.14	1172.26	16.03	2019.77
R110	1094.01	11.04	2779.08	1,092.99	10.85	5758.3	1107.27	11.05	2444.1
R111	1060.8	8.03	1762.1	1,063.67	11	4189.85	1065.1	8.04	6515.37
R112	978.55	9.27	2117.51	978.84	7.97	5144.98	1001.11	17.67	2005.91
Average	1189.59	6.74	**2142.88**	1,191.41	7.23	4490.91	1195.46	8.4	2568.17
R201	1152.84	5.43	1449.99	1,160.69	7.31	2617.62	1158	9.54	571.79
R202	1045.6	6.41	584.01	1,047.86	4.85	2027.49	1046.7	6.14	631.74
R203	881.21	13.14	1024.96	902.79	16.16	1817.66	883.3	13.67	726.7
R204	746.26	6.41	163.89	751.52	10.18	2332.48	745.07	6.61	160.85
R205	962.22	11.35	446.8	970.23	12.66	3861.68	964.81	12.59	279.34
R206	896.71	10.06	873.02	898.35	8.78	1854.49	895.04	9	232.22
R207	809.54	12.14	661.08	814.33	12.66	2861.93	806.93	10.57	553.34
R208	718.59	5.65	656.86	722.61	5.45	1665.31	718.84	5.24	338.05
R209	870.71	10.72	358.65	871.76	9.13	1554.47	869.74	10.76	524.95
R210	921.08	10.35	1081.45	927.83	8.42	2560.37	922.63	8.95	516.21
R211	793.8	25.14	1093.71	805.81	18.83	3396.25	790.91	17.07	1174.08
Average	890.78	10.62	763.13	897.62	10.4	2413.61	891.09	10.01	519.02
RC101	1635.92	9.83	1899.75	1,644.54	8.7	7050.14	1640.55	13.45	1868.31
RC102	1480.98	10.07	2304.7	1,480.99	6.02	6132.79	1485.52	8.28	1972.41
RC103	1306.41	21.16	2434.59	1,316.19	22.05	8286.84	1319.97	12.51	2087.14
RC104	1159.2	13.75	2548	1,168.64	7.93	6243.03	1171.98	18.16	1764.28
RC105	1528.76	13.48	3416.28	1,546.58	16.34	6904.83	1524.47	5.92	2549.62
RC106	1394.5	12.8	2992.33	1,400.25	11.1	6790.99	1412.47	15.58	2077.23
RC107	1233.4	14.21	2581.65	1,246.53	13.13	8428.51	1254.67	18.24	2061.62
RC108	1147.52	12.42	2526.22	1,150.98	10.51	7362.07	1156.7	16.32	2842.96
Average	1360.84	13.47	2587.94	1,369.34	**11.97**	7149.9	1370.79	13.56	2152.95
RC201	1274.6	5.5	1414.67	1,277.13	7.67	3115.65	1278.66	7.24	669.24
RC202	1104.16	8.78	153.23	1,106.37	7.71	1616.7	1103.66	6.66	259.09
RC203	945.77	11.09	432.17	952.98	7.83	1443.14	941.64	11.2	249.41
RC204	799.13	1.48	432.78	799.49	4.16	1308.94	799.23	3.93	632.76
RC205	1159.84	8.1	913.09	1,170.89	17.13	2764.96	1163.48	8.52	536.94
RC206	1078.7	9.29	714.94	1,079.07	12.89	1997.49	1083.47	14.14	835.22
RC207	992.31	18.69	800.38	1,007.76	25.61	3485.85	987.02	9.96	473.87
RC208	805.75	30.79	698.75	838.82	40.04	2441.05	809.69	28.65	913.12
Average	1020.03	11.72	695	1,029.06	15.38	2271.72	1020.86	11.29	**571.21**
Overall Average	987.41	7.13	1083.4	991.66	7.58	2789.72	991.38	8.21	1152.22
Count	2	2	2	1	2	2	0	0	1

"MTD" represents the average total travelled distance (distance cost) on each instance after 30 independent runs. "Std Dev" represents the standard deviation on each instance.

TABLE 7. Comparison with the four recently published hybrid algorithms.

Instances	LGA [17]		GA-PSO [37]		M-MOEA/D [38]		ESS-PSO [32]		RRGA	
	TD	NV	TD	NV	TD	NV	TD	NV	TD	NV
C101	**828.94**	10	**828.94**	10	**828.94**	10	**828.94**	10	**828.94**	10
C102	**828.94**	10	**828.94**	10	**828.94**	10	**828.94**	10	**828.94**	10
C103	**828.06**	10	**828.06**	10	828.07	10	**828.06**	10	828.07	10
C104	**824.78**	10	**824.78**	10	**824.78**	10	**824.78**	10	**824.78**	10
C105	**828.94**	10	**828.94**	10	**828.94**	10	**828.94**	10	**828.94**	10
C106	**828.94**	10	**828.94**	10	**828.94**	10	**828.94**	10	**828.94**	10
C107	**828.94**	10	**828.94**	10	**828.94**	10	**828.94**	10	**828.94**	10
C108	**828.94**	10	**828.94**	10	**828.94**	10	**828.94**	10	**828.94**	10
C109	**828.94**	10.00	**828.94**	10.00	**828.94**	10.00	**828.94**	10.00	**828.94**	10.00
Average	**828.38**	**10.00**	**828.38**	**10.00**	**828.38**	**10.00**	**828.38**	**10.00**	**828.38**	**10.00**
C201	**591.56**	3	**591.56**	3	**591.56**	3	**591.56**	3	**591.56**	3
C202	**591.56**	3	**591.56**	3	**591.56**	3	**591.56**	3	**591.56**	3
C203	**591.17**	3	**591.17**	3	**591.17**	3	**591.17**	3	**591.17**	3
C204	**590.6**	3	**590.6**	3	594.89	3	**590.6**	3	**590.6**	3
C205	**588.88**	3	**588.88**	3	**588.88**	3	**588.88**	3	**588.88**	3
C206	**588.49**	3	**588.49**	3	**588.49**	3	**588.49**	3	**588.49**	3
C207	**588.29**	3	**588.29**	3	**588.29**	3	**588.29**	3	**588.29**	3
C208	**588.32**	3	**588.32**	3	**588.32**	3	**588.32**	3	**588.32**	3
Average	**589.86**	**3.00**	**589.86**	**3.00**	**590.40**	**3.00**	**589.86**	**3.00**	**589.86**	**3.00**
R101	1646.9	20	**1642.87**	20	1644.7	20	1642.88	20	1642.88	20
R102	1474.28	18	**1472.62**	18	1473.73	18	1472.92	18	1473.62	18
R103	1222.68	15	**1213.62**	14	**1213.62**	14	1213.73	14	**1213.62**	14
R104	989.53	11	982.01	10	991.91	11	**976.61**	11	**976.61**	11
R105	1382.78	16	1360.78	15	1366.58	15	**1360.76**	15	1360.78	15
R106	1250.11	13	1241.52	13	1249.22	13	**1239.37**	13	**1239.37**	13
R107	1083.42	12	1076.13	11	1086.22	11	1073.46	11	**1072.12**	11
R108	952.44	10	**948.57**	10	965.52	10	950.59	10	951.22	10
R109	1160.69	13	**1151.84**	13	1155.38	13	**1151.84**	13	**1151.84**	13
R110	1080.69	12	1080.36	11	1106.03	12	1073.46	12	**1072.42**	12
R111	1057.64	12	**1053.5**	12	1073.82	11	**1053.5**	12	**1053.5**	12
R112	965	10	960.68	10	981.43	10	**953.62**	10	959.97	10
Average	1188.85	13.50	1182.04	13.08	1192.35	13.17	**1180.23**	13.25	1180.66	13.25
R201	1156.29	9	1148.48	9	1185.79	6	1152.63	8	**1147.8**	8
R202	1042.25	8	1049.74	7	1049.72	5	1036.3	6	**1034.35**	8
R203	877.29	6	900.08	5	899.36	5	875.21	6	**874.87**	6
R204	736.52	4	772.33	4	743.29	5	**737.43**	4	**735.8**	5
R205	960.35	6	970.89	6	954.48	5	**954.16**	5	**954.16**	5
R206	894.19	6	898.91	5	887.9	4	**884.25**	5	884.85	5
R207	800.79	4	814.78	3	809.51	4	801.15	4	**797.99**	4
R208	706.86	3	723.61	3	711.59	3	706.86	3	**705.33**	4
R209	860.63	5	879.53	6	867.47	4	**860.11**	5	860.46	5
R210	948.82	5	932.89	7	920.06	5	912.8	5	**904.78**	6
R211	762.23	5	808.56	4	767.1	4	**757.6**	4	764.69	4
Average	886.02	5.55	899.98	5.36	890.57	4.55	879.86	5.00	**878.64**	5.45
RC101	1660.55	16	**1623.58**	15	1646.65	16	1639.75	16	1623.59	15
RC102	1494.92	15	1466.84	14	1484.48	15	1461.33	14	**1461.23**	14
RC103	1276.05	12	**1261.67**	11	1274.85	11	1277.55	12	1262.98	11
RC104	1151.63	10	**1135.48**	10	1145.79	10	1138.13	10	**1135.48**	10
RC105	1556.21	16	1618.55	16	1528.61	15	1519.46	15	**1518.58**	16
RC106	1402.25	14	**1377.35**	13	1399.17	13	1378.62	13	**1377.35**	13
RC107	**1212.83**	12	**1212.83**	12	1235.54	12	**1212.83**	12	**1212.83**	12
RC108	1133.25	11	**1117.53**	11	1138.95	11	1118.57	11	1140.73	11
Average	1360.96	13.25	1351.73	12.75	1356.76	12.88	1343.28	12.88	**1341.60**	12.75
RC201	1281.63	10	1387.55	8	1289.94	7	**1265.56**	9	1266.11	9
RC202	1103.47	8	1148.84	9	1118.66	5	1096.53	8	**1095.64**	8

TABLE 7. *(Continued.)* **Comparison with the four recently published hybrid algorithms.**

RC203	942	6	945.96	5	940.55	5	**926.82**	5	**926.82**	5
RC204	796.12	4	798.41	3	792.98	4	**786.38**	4	788.66	4
RC205	1168.89	8	1161.81	7	1187.48	6	**1157.55**	7	**1157.55**	7
RC206	1060.52	7	1059.89	7	1089.14	5	1057.83	6	**1054.61**	7
RC207	970.97	7	976.4	7	987.88	5	966.37	6	**966.08**	6
RC208	782.7	5	795.39	5	807.83	4	**779.31**	4	**779.31**	4
Average	1013.29	6.88	1034.28	6.38	1026.81	5.13	1004.54	6.13	**1004.35**	6.25
Overall Average	984.18	8.91	986.86	8.64	987.05	8.35	977.37	8.59	**976.95**	8.68
Count	18		28		16		34		**42**	

"TD" represents the total travelled distance. "NV" represents the number of used vehicles.

However, in C2 instance type, the RRGA and all variants have the minimum average total travelled distance. In R1 and R2 instances, the RRGA achieves more minimum total travelled distance (R1 - 11 instances and R2 - 9 instances) than variant 1 (R1 - 6 instances and R2 - 2 instances), 2 (R1 - 7 instances and R2 - 7 instances), 3 (R1 - 9 instances and R2 - 8 instances), 4 (R1 - 8 instances and R2 - 7 instances) and 5 (R1 - 5 instances, and R2 - 8 instances). In R1 instance type, variants 2 and 3 have the minimum average operation time and average total travelled distance, respectively. However, the variant 2 minimum average total travelled distance differences are insignificant compared to the RRGA. In R2 instance type, the RRGA and variant 3 have the total travelled distance and minimum average operation time, respectively. The difference in minimum average total travelled distance between the RRGA and variant 3 are insignificant.

In RC1 instances, the RRGA has similar number of minimum total travelled distance with variant 3 (RC1 has 5 instances), but the minimum average operation time is better than the RRGA, but the difference in minimum average total distance is insignificant. Nevertheless, in RC2 instance type, the RRGA shows better results in average total travelled distance (1004.35) and operation time (1.58 seconds) than other variants.

Overall, the RRGA shows remarkable results (47 instances) than other variants (variant 1 – 27 instances, variant 2 - 36 instances, variant 3 - 43 instances, variant 4 - 37 instances, and variant 5 - 39 instances) in overall average total travelled distance eventhough the overall average operation time (68.29 seconds) is marginally lower than variant 3 (57.06 seconds). In this comparison, the RRGA has better optimization performance and reasonable convergence speed than other variants. It is worth mentioning some instances have longer average operation time because the production machines used in this experiment are personal computers and not a server specification. The operation time can be lowered down drastically by vertically scaling the personal desktop or using a server specification. Nowadays, the cost of acquiring a better server specification is within the affordable range and the deploy speed is faster and economical through on-demand requests with the advent of cloud computing and the demise of on-premise deployment.

Table 6 compares the average of total travelled distance, standard deviation, and operation time after 30 independent runs with other variants. "MTD" represents the average of total travelled distance after 30 independent runs. This metric measures the best results on average after 30 independent runs. "Std Dev" represents the standard deviation of total travelled distance after 30 independent runs and this metric measures the magnitude of best results or a minimum average of total travelled distance differences. "AT" represents the minimum average operation time after 30 independent runs in seconds. The "count" represents the number of minimum total travelled distance in the instances. The "Average" represents the minimum average of each metric in each instance type. The "Overall Average" represents the minimum average of each metric in all instances. In C1 instances, the RRGA has more minimum average of total travelled distances (9 instances) than variant 1 (8 instances), variant 2 (6 instances), variant 4 (8 instances), and variant 5 (5 instances) except variant 3. In C1 instance type, the RRGA and variant 3 have the minimum average operation time and better standard deviation than other variants except variant 3. Although variant 4 has better average operation time (18 seconds) than the RRGA, their results on average total travelled distance and standard deviation are slightly higher than the RRGA, which makes variant 4 slightly inferior in this instance type. In C2 instances, all variants except for variant 5 have the same number of minimum total travelled distances (8 instances), and standard deviations (0 values) with the RRGA. In C2 instance type, the minimum average total travelled distance (589.86) and standard deviation (0 value) in RRGA are similar to all variants except for variant 5, but the average operation time is slightly above (5.92 seconds) the variant 3 (0.62 seconds) which makes the RRGA slightly underperformed in this instance type. In R1 instances, the RRGA has more minimum total travelled distance (6 instances) than variant 1 (1 instance), variant 2 (5 instances), variant 3 (0 instances), variant 4 (0 instances), and variant 5 (0 instances). In R1 instance type, the RRGA has the minimum total travelled distance (1187.34) and standard deviation (6.01) than other variants, even though variant 3 has minimum average operation time (2142.88). There are more instances (6 instances) in variant 2 with the number

TABLE 8. Comparison with the four recently published hybrid algorithms average results.

	LGA [17]		GA-PSO [37]		M-MOEA/D [38]		ESS-PSO [32]		RRGA	
Instances	MTD	Std Dev	MTD	Std Dev	MTD	Std Dev	MTD	Std Dev	MTD	Std Dev
C101	**828.94**	**0**	842.60	N/A	**828.94**	**0**	**828.94**	**0**	**828.94**	**0**
C102	**828.94**	**0**	828.94	N/A	**828.94**	**0**	**828.94**	**0**	**828.94**	**0**
C103	828.07	**0**	828.06	N/A	828.07	**0**	**828.06**	**0**	828.07	**0**
C104	**824.78**	**0**	**824.78**	N/A	827.93	5.57	**824.78**	**0**	**824.78**	**0**
C105	**828.94**	**0**	841.60	N/A	**828.94**	**0**	**828.94**	**0**	**828.94**	**0**
C106	**828.94**	**0**	874.62	N/A	**828.94**	**0**	**828.94**	**0**	**828.94**	**0**
C107	**828.94**	**0**	842.70	N/A	**828.94**	**0**	**828.94**	**0**	**828.94**	**0**
C108	**828.94**	**0**	841.29	N/A	**828.94**	**0**	**828.94**	**0**	**828.94**	**0**
C109	**828.94**	**0**	828.94	N/A	**828.94**	**0**	**828.94**	**0**	**828.94**	**0**
Average	**828.38**	**0**	839.28		828.73	0.62	**828.38**	**0**	**828.38**	**0**
C201	595.43	11.86	598.87	N/A	**591.56**	**0**	**591.56**	**0**	**591.56**	**0**
C202	593.23	9.14	**591.56**	N/A	**591.56**	**0**	**591.56**	**0**	**591.56**	**0**
C203	594.75	14.67	604.69	N/A	591.31	3.66	**591.17**	**0**	**591.17**	**0**
C204	591.95	6.79	628.49	N/A	603.16	10.12	592.47	3.12	**590.60**	**0**
C205	**588.88**	**0**	599.01	N/A	**588.88**	**0**	**588.88**	**0**	**588.88**	**0**
C206	**588.49**	**0**	**588.49**	N/A	**588.49**	**0**	**588.49**	**0**	**588.49**	**0**
C207	**588.29**	**0**	599.39	N/A	**588.29**	**0**	**588.29**	**0**	**588.29**	**0**
C208	**588.32**	**0**	599.79	N/A	**588.32**	**0**	**588.32**	**0**	**588.32**	**0**
Average	591.17	5.31	601.29		591.45	1.72	590.09	0.39	**589.86**	**0**
R101	1671.79	13.74	1645.83	N/A	1645.82	5.48	1650.11	7.35	**1644.61**	**3.21**
R102	1502.38	15.39	1477.14	N/A	**1463.91**	4	1478.03	6.14	1474.15	**0.59**
R103	1244.01	14.87	1239.01	N/A	1223.27	**2.41**	1222.20	4.16	**1218.86**	3.73
R104	1009.09	14.87	992.52	N/A	1002.54	**6.04**	994.87	13.05	**985.39**	8.73
R105	1405.33	15.26	1366.42	N/A	1372.01	**5.45**	**1365.98**	5.8	1368.65	7.91
R106	1278.32	20.53	1247.29	N/A	1256.45	**4.76**	1250.16	10.13	**1244.42**	5.58
R107	1106.74	12.36	1088.49	N/A	1097.41	**4.46**	1082.16	6.38	**1078.82**	4.81
R108	965.89	8.48	955.82	N/A	965.82	5.91	973.57	10.48	**952.72**	**1.99**
R109	1187.91	16.64	1164.71	N/A	1163.07	7.2	1170.23	11.31	**1156.43**	**5.41**
R110	1105.71	12.23	1098.14	N/A	1100.83	**8.51**	1098.60	12.41	**1086.88**	12.6
R111	1086.66	17.42	1069.14	N/A	1076.61	7.16	1063.01	7.1	**1058.89**	**6.76**
R112	977.82	7.9	**968.58**	N/A	978.16	**6.47**	979.18	12.33	978.20	10.81
Average	1211.80	14.141	1192.76		1195.49	**5.65**	1194.01	8.887	**1187.34**	6.011
R201	1176.43	14.46	1178.39	N/A	1178.90	4.85	1170.79	10.04	**1148.49**	**1.68**
R202	1066.69	14.21	1081.57	N/A	1053.44	7.88	1046.53	6.22	1038.60	**2.97**
R203	899.00	14.61	922.71	N/A	896.05	7.77	886.40	8.55	**875.22**	**1.34**
R204	754.83	11.15	801.47	N/A	758.13	6.5	751.36	11.71	**736.61**	**2.89**
R205	985.74	10.43	977.95	N/A	975.83	7.61	971.60	12.3	**955.57**	**2.44**
R206	911.14	11.94	903.89	N/A	908.18	6.76	894.39	8.35	**886.40**	**2.8**
R207	827.36	13.83	836.67	N/A	826.74	11.31	821.15	13.58	**797.99**	**0**
R208	721.71	8.35	725.50	N/A	732.31	6.99	725.99	**5.01**	**711.74**	6.79
R209	884.21	11.73	891.82	N/A	882.22	6.71	875.71	9.72	**861.15**	**1.86**
R210	984.82	9.37	937.89	N/A	938.63	9.31	930.26	10.9	**910.74**	**4.8**
R211	774.44	8.85	824.99	N/A	781.23	7.61	788.82	24.25	**764.77**	**0.32**
Average	907.85	11.721	916.62		902.88	7.573	896.64	10.97	**880.66**	**2.54**
RC101	1698.53	22.77	1645.18	N/A	1656.01	6.93	1660.48	7.2	**1629.38**	**6.91**
RC102	1519.53	16.42	1487.10	N/A	1488.79	**7.29**	1486.56	7.4	**1478.07**	8.8
RC103	1330.47	21.8	**1262.04**	N/A	1298.32	18.5	1314.67	**12**	1289.43	20.76
RC104	1164.73	12.52	**1135.49**	N/A	1168.13	**5.43**	1155.05	7.81	1152.33	14.33
RC105	1581.81	12.82	1621.16	N/A	1554.79	**8.5**	1566.24	15.72	**1529.55**	14.73
RC106	1432.26	15.15	1392.80	N/A	1413.38	**7.41**	1406.79	16.83	**1390.18**	11.91
RC107	1252.72	20.88	1226.01	N/A	1256.98	**11.9**	1236.86	16.25	**1232.84**	17.33
RC108	1141.46	9.08	**1126.40**	N/A	1149.65	9.76	1140.28	7.16	1140.13	**4.99**
Average	1390.19	16.43	1362.02		1373.26	**9.47**	1370.87	11.29	**1355.24**	12.5
RC201	1296.62	7.56	1391.43	N/A	1284.59	5.71	1285.16	9.05	**1270.86**	**5.65**
RC202	1124.85	10.75	1173.34	N/A	1122.97	5.25	1111.22	9.33	**1098.23**	**3.85**
RC203	960.11	11.78	990.67	N/A	951.30	6.33	944.69	7.28	**926.82**	**0**
RC204	808.39	6.59	798.67	N/A	809.09	8.13	797.47	7.16	**796.00**	**3.89**
RC205	1183.86	8.15	1187.56	N/A	1172.80	16.34	1173.06	12.21	**1157.59**	**0.06**
RC206	1086.19	10.5	1092.22	N/A	1082.93	8.4	1082.95	9.86	**1059.14**	**7.38**
RC207	991.02	9.75	995.65	N/A	998.46	9.82	985.59	**9.64**	**980.42**	11.4
RC208	804.90	11.79	807.27	N/A	809.23	**8.65**	808.06	23.62	**787.17**	15.89
Average	1031.99	9.61	1054.60	N/A	1028.92	8.58	1023.53	11.02	**1009.53**	**6.02**
Overall Average	1001.61	9.81	1001.66	N/A	994.38	5.62	991.47	7.30	**982.64**	4.43

"MTD" represents the average total travelled distance (distance cost) on each instance after 30 independent runs. "Std Dev" represents the standard deviation in each instance.

TABLE 9. The Wilcoxon signed-ranks non-parametric test.

Instance	RRGA vs LGA [17]		RRGA vs GA-PSO [37]		RRGA vs M-MOEA/D [38]		RRGA vs ESS-PSO [32]	
C1	1		0.046	+	0.317		0.317	
C2	0.068		0.028	+	0.18		0.317	
R1	0.003	+	0.023	+	0.012	+	0.004	+
R2	0.003	+	0.003	+	0.003	+	0.003	+
RC1	0.012	+	0.89		0.012	+	0.012	+
RC2	0.012	+	0.012	+	0.012	+	0.012	+

The notation "+" represents the null hypothesis is rejected, and most of the instances types in the RRGA are statistically better than comparing algorithms.

of minimum average total travelled distance than the RRGA (5 instances) in R2 instances. However, the RRGA is still leading when it comes to the minimal average total travelled distance (880.66) and average operation time (475.29 seconds) than other variants in this instance type. This result means the RRGA outperforms other variants in R1 and R2 instance type. In RC1 instance type, the RRGA outperform other variants, but in RC2 instance type, the RRGA is underperformed but the difference is insignificant. In summary, the RRGA still has the best overall average total travelled distance (982.64) and overall average operation time (1050 seconds) than other variants. These results indicate that RRGA has superior average optimization results and average convergence speed.

2) COMPARISON WITH THE FOUR RECENTLY PUBLISHED HYBRID ALGORITHMS

Table 7 compares the RRGA with four recently published hybrid algorithms. Our results show that RRGA has high number of minimum total travelled distances (42 instances or 75% of total instances) than other hybrid algorithms (ESS-PSO – 34 instances, M-MOEA/D – 16 instances, GA-PSO – 28 instances, and LGA – 18 instances). The RRGA also outperforms other hybrid algorithms by instances (R1 - 7 instances, R2 - 8 instances, RC1 - 5 instances, and RC2 - 6 instances). These results show that the RRGA has superior optimization performance than the four recently published hybrid algorithms.

Table 8 compares the average total travelled distance and standard deviation with the four recently published hybrid algorithms. The "N/A" represents that the result is not available. In instances comparison, our proposed algorithm has the highest number of average total travelled distances in C1 (9 instances), C2 (8 instances), R1 (9 instances), R2 (10 instances), RC1 (5 instances), and RC2 (8 instances) than the recently published hybrid algorithms. In addition, if we compare the overall average, the RRGA also has the minimum average total travelled distance (C2 - 589.86, R1 - 1187.34, R2 - 880.66, RC1 - 1355.24, and RC2 - 1009.53) and standard deviation (C1 – 0, C2 - 0, R2 - 2.54, and RC2 - 6.02) which outperforms four recently published hybrid algorithms. This result explains that RRGA has better average optimization performance than the four recently published hybrid algorithms even after 30 independent runs.

Table 9 shows that the Wilcoxon signed-ranks is a non-parametric test that is used to determine whether two dependents of data are different. The p-value is the probability of obtaining the observed difference between the two groups are based on chance. If the p-value is very low (<0.05), the null hypothesis will be rejected, and the result is significant; otherwise, the null hypothesis will be accepted. Table 10 shows the p-value resolve by the Wilcoxon test with significance level $\alpha = 0.05$. This measurement is used to compare the RRGA with the four recently published hybrid algorithm papers, whether they are statistically significant or not. The notation "+" represents the null hypothesis is rejected, and the RRGA is statistically better performance for that particular instance. Except in LGA [17], M-MOEA/D [38], and ESS-PSO [36] in C1 and C2 instance type and GA-PSO [37] in RC1 instance type, our proposed algorithm is statistically better in most of the instances types.

3) COMPARISON WITH THE BEST-KNOWN SOLUTIONS

Table 10 compares with the best-known solutions [36]. In this table, "BKS" is the best-known solution, and "TD" is the total travelled distance. "Gap %" is the percentage difference between the proposed algorithm and the best-known result. "Average Gap %" is the average percentage difference between the proposed algorithm and the best-known solution. Our results show that 33 instances which have similar results compare to the best-known solutions, and 7 instances which outperform the best-known solutions. This result is equivalent to 71.42% of all the instances, which is similar or better results than best-known solutions. If we compare with the minimum average total travelled distance, 50% (28 instances) of the instances outperform the best-known solutions, 23 instances which do not have the information available, and 4 instances which have results inferior to the best-known solutions, but the differences are relatively insignificant (0.55%). In this case, we can conclude that the RRGA can produce a lot of best-known solutions in many instances and outperform the best-known average total travelled distance in most of the instances.

4) COMPARISON WITH THE NINE LEADING ALGORITHMS

Table 11 shows nine leading algorithms are selected for comparison. Their instance type results are compared with one another, and the best results are highlighted in bold. The RRGA achieves the best results in most of the instance types except for RC1 instance type. However, the result difference

TABLE 10. Comparison with the best-known solutions.

	Best-known					RRGA					
Instances	NV	TD	MNV	MTD	Ref	NV	TD	MNV	MTD	Gap %	Gap Average %
C101	10	828.94	N/A	N/A	[39]	10	**828.94**	10.00	828.94	0.00	N/A
C102	10	828.94	N/A	N/A	[39]	10	**828.94**	10.00	828.94	0.00	N/A
C103	10	828.07	N/A	N/A	[39]	10	**828.07**	10.00	828.07	0.00	N/A
C104	10	824.78	N/A	N/A	[39]	10	**824.78**	10.00	824.78	0.00	N/A
C105	10	828.94	N/A	N/A	[39]	10	**828.94**	10.00	828.94	0.00	N/A
C106	10	828.94	N/A	N/A	[39]	10	**828.94**	10.00	828.94	0.00	N/A
C107	10	828.94	N/A	N/A	[39]	10	**828.94**	10.00	828.94	0.00	N/A
C108	10	828.94	N/A	N/A	[39]	10	**828.94**	10.00	828.94	0.00	N/A
C109	10	828.94	N/A	N/A	[39]	10	**828.94**	10.00	828.94	0.00	N/A
C201	3	591.56	N/A	N/A	[39]	3	**591.56**	3.00	591.56	0.00	N/A
C202	3	591.56	N/A	N/A	[39]	3	**591.56**	3.00	591.56	0.00	N/A
C203	3	591.17	N/A	N/A	[39]	3	**591.17**	3.00	591.17	0.00	N/A
C204	3	590.60	N/A	N/A	[39]	3	**590.60**	3.00	590.60	0.00	N/A
C205	3	588.88	N/A	N/A	[39]	3	**588.88**	3.00	588.88	0.00	N/A
C206	3	588.49	N/A	N/A	[39]	3	**588.49**	3.00	588.49	0.00	N/A
C207	3	588.29	N/A	N/A	[39]	3	**588.29**	3.00	588.29	0.00	N/A
C208	3	588.32	N/A	N/A	[39]	3	**588.32**	3.00	588.32	0.00	N/A
R101	18	1613.59	N/A	1674.75	[29]	20	1642.88	19.93	**1644.61***	1.82	-1.80
R102	18	1454.68	N/A	1527.11	[29]	18	1473.62	18.00	**1474.15***	1.30	-3.47
R103	14	1213.62	N/A	N/A	[39]	14	**1213.62**	14.00	1218.86	0.00	N/A
R104	10	974.24	N/A	1019.37	[29]	11	976.61	11.00	**985.39***	0.24	-3.33
R105	15	1360.78	15.91	1371.52	[40]	15	**1360.78**	15.13	**1368.65***	0.00	-0.21
R106	13	1240.47	13.59	1252.44	[40]	13	**1239.37***	13.03	**1244.42***	N/A	-0.64
R107	11	1073.34	11.73	1083.10	[40]	11	**1072.12***	11.20	**1078.82***	N/A	-0.40
R108	10	947.55	10.74	959.65	[40]	10	951.22	10.17	**952.72***	0.39	-0.72
R109	13	1151.64	12.97	1157.27	[40]	13	1151.84	12.83	**1156.43***	0.02	-0.07
R110	12	1072.41	12	1082.72	[40]	12	1072.42	11.97	1086.88	0.00	0.38
R111	12	1053.50	12	1063.21	[40]	12	**1053.50**	11.90	**1058.89***	0.00	-0.41
R112	10	953.63	N/A	N/A	[39]	10	959.97	10.47	978.20	0.66	N/A
R201	9	1144.48	N/A	N/A	[41]	8	1147.80	7.97	1148.49	0.29	N/A
R202	8	1034.35	7.4	1038.40	[40]	8	**1034.35**	7.03	1038.60	0.00	0.02
R203	6	874.87	6	875.87	[40]	6	**874.87**	5.93	**875.22***	0.00	-0.07
R204	4	736.52	4.46	741.41	[40]	5	**735.80***	4.93	**736.61***	N/A	-0.65
R205	5	954.16	6.05	964.69	[42]	5	**954.16**	5.17	**955.57***	0.00	-0.95
R206	5	879.89	5.33	892.55	[40]	5	884.85	4.90	**886.40***	0.56	-0.69
R207	4	799.86	4.66	814.05	[40]	4	**797.99***	4.00	**797.99***	N/A	-1.97
R208	4	705.45	3.5	714.37	[40]	4	**705.33***	3.83	**711.74***	N/A	-0.37
R209	5	859.39	5.26	867.52	[40]	5	860.46	4.90	**861.15***	0.12	-0.73
R210	5	910.70	6.1	918.37	[40]	6	**904.78***	5.70	**910.74***	N/A	-0.83
R211	4	755.96	4.7	765.64	[40]	4	764.69	4.00	**764.77***	1.15	-0.11
RC101	15	1623.58	N/A	N/A	[39]	15	**1623.59**	15.17	1629.38	0.00	N/A
RC102	14	1461.23	14.65	1480.82	[40]	14	**1461.23**	14.50	**1478.07***	0.00	-0.19
RC103	11	1261.67	N/A	N/A	[43]	11	1262.98	11.57	1289.43	0.10	N/A
RC104	10	1135.48	N/A	N/A	[44]	10	**1135.48**	10.30	1152.33	0.00	N/A
RC105	16	1518.58	15.96	1540.66	[40]	16	**1518.58**	15.57	**1529.55***	0.00	-0.72
RC106	13	1371.69	N/A	1403.89	[29]	13	1377.35	13.23	**1390.18***	0.41	-0.98
RC107	12	1212.83	12.03	1227.81	[40]	12	**1212.83**	10.30	1232.84	0.00	0.41
RC108	11	1117.53	11	1135.81	[40]	11	**1140.73**	11.00	1140.13	2.08	0.38
RC201	6	1134.91	N/A	1337.21	[29]	9	**1266.11**	8.73	1270.86*	11.56	-4.96
RC202	8	1095.64	7.84	1101.03	[40]	8	**1095.64**	7.67	1098.23*	0.00	-0.25
RC203	5	928.51	5.29	943.81	[40]	5	**926.82***	5.00	**926.82***	N/A	-1.80
RC204	4	786.38	4.05	799.19	[40]	4	788.66	3.40	**796.00***	0.29	-0.40
RC205	7	1157.55	7.8	1164.43	[40]	7	**1157.55**	7.00	**1157.59***	0.00	-0.59
RC206	7	1054.61	6.39	1067.49	[40]	7	**1054.61**	6.70	**1059.14***	0.00	-0.78
RC207	6	966.08	6.07	975.24	[40]	6	**966.08**	5.37	980.42	0.00	0.53
RC208	4	779.31	4.98	791.35	[40]	4	**779.31**	4.00	**787.1***	0.00	-0.54
Count							**40**		**28**		

"TD" represents the total travelled distance. "NV" represents the number of used vehicles. "MTD" represents the average of total travelled distance (distance cost) on each instance. "MNV" represents the average of the number of used vehicles on each instance

is insignificantly lower than the CGH algorithm. On average, the RRGA has the best average solution (978.12) than other leading algorithms and very close to the best-known average result (974.02).

V. CONCLUSIONS

This article presents a distributed RRGA to solve the VRPTW. In this proposed algorithm, the RRGA process lifecycle is divided into RRGA phase and the RR phase.

TABLE 11. The RRGA compares with the nine-leading algorithm.

	C1	C2	R1	R2	RC1	RC2	Average
Best-known	828.38	589.86	1175.80	877.78	1337.82	987.87	974.02
HMOEA(2006)[29]	828.74	590.69	1187.35	951.74	1355.37	1068.26	1005.19
CGH(2007)[41]	828.38	589.86	1196.80	899.90	1341.70	1015.90	987.42
ACO-Tabu(2011)[22]	829.01	590.78	1196.96	951.36	1380.55	1095.84	1014.77
LGA(2011)[17]	828.38	589.86	1188.85	885.78	1360.96	1013.29	984.59
Meta-HSA(2015)[45]	838.47	605.41	1207.76	977.19	1381.96	1099.12	1018.32
GA-PSO (2015)[37]	828.38	589.86	1182.04	899.98	1351.73	1034.28	988.33
MOEA/D (2015)[38]	828.38	589.86	1192.35	889.66	1356.76	1026.81	988.16
Tabu-ABC(2017)[46]	828.38	590.39	1187.90	891.24	1361.08	1017.47	986.88
ESS-PSO(2018)[32]	828.38	589.86	1180.22	879.86	1343.28	1004.54	978.54
RRGA	828.38	589.86	1180.60	878.64	1341.57	1004.35	978.12

In RRGA phase, it has two subprocedures. In the first subprocedure, the genetic operators generate a random customer list, and in the second subprocedure, the strategy types in the RR principle create a new solution based on the generated customer list. In RR principle, part of the solution can be randomly or radially ruined and reconstructed back into a new solution using the recreate method. Therefore, the radial and random strategies were proposed in the RR principle because they can produce excellent results compared to other strategies. In the RR phase, it executes a procedure iteratively until a termination criterion is met. In these iterations, the new solution is intensely improved. In these ways, these two phases will focus and make use of exploration and exploitation traits, thus promoting global optimal.

The RRGA achieves outstanding results when compared with the best-known solution, four recently published hybrid algorithms, and nine leading hybrid algorithms. Moreover, our results are statistically better, and this shows that our proposed algorithm is highly effective and better convergence speed. Although our results are excellent, there is still a myriad of problems we can solve using our proposed algorithm, especially in VRP variants.

ACKNOWLEDGMENT

These experiments are carried out at the Universiti Tunku Abdul Rahman's (UTAR) Numerical and High-performance Computing lab, MIMOS lab, and Traffic Equipment lab. This research is supported by the Universiti Tunku Abdul Rahman, UTAR Research Fund (UTARRF), no. IPSR/RMC/UTARRF/2019-C2/M01.

REFERENCES

[1] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Manage. Sci.*, vol. 6, no. 1, pp. 80–91, Oct. 1959.
[2] P. Toth, D. Vigo, P. Toth, and D. Vigo, *Vehicle Routing: Problems, Methods, and Applications*, 2nd ed. Philadelphia, PA, USA: SIAM, 2014.
[3] M. M. Solomon, "Algorithms for the vehicle routing and scheduling problems with time window constraints," *Oper. Res.*, vol. 35, no. 2, pp. 254–265, Apr. 1987.
[4] J. K. Lenstra and A. H. G. R. Kan, "Econometric institute complexity of vehicle routing and scheduling problems," *Networks*, vol. 2, no. 2, pp. 221–227, 1981.
[5] D. Pecin, C. Contardo, G. Desaulniers, and E. Uchoa, "New enhancements for the exact solution of the vehicle routing problem with time windows," *Informs J. Comput.*, vol. 29, no. 3, pp. 489–502, Aug. 2017.
[6] K. Sörensen, M. Sevaux, and F. Glover, "A history of metaheuristics," *Handb. Heuristics*, vol. 2, pp. 791–808, 2018.
[7] O. Bräysy and M. Gendreau, "Vehicle routing problem with time windows, part II: Metaheuristics," *Transp. Sci.*, vol. 39, pp. 119–139, Feb. 2005.
[8] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MA, USA: Univ. Michigan Press, 1975.
[9] C. Darwin and D. Quammen, *On the Origin of Species: The Illustrated Edition*. Sterling Signature, 2008, p. 560.
[10] V. S. Kumar, M. R. Thansekhar, R. Saravanan, and S. M. J. Amali, "Solving multi-objective vehicle routing problem with time windows by FAGA," *Procedia Eng.*, vol. 97, pp. 2176–2185, Jan. 2014.
[11] C.-B. Cheng and K.-P. Wang, "Solving a vehicle routing problem with time windows by a decomposition technique and a genetic algorithm," *Expert Syst. Appl.*, vol. 36, no. 4, pp. 7758–7763, May 2009.
[12] R. Pérez-Rodríguez and A. Hernández-Aguirre, "A hybrid estimation of distribution algorithm for the vehicle routing problem with time windows," *Comput. Ind. Eng.*, vol. 130, pp. 75–96, Apr. 2019.
[13] J. Berger and M. Barkaoui, "A parallel hybrid genetic algorithm for the vehicle routing problem with time windows," *Comput. Oper. Res.*, vol. 31, no. 12, pp. 2037–2053, Oct. 2004.
[14] D. M. Pierre and N. Zakaria, "Stochastic partially optimized cyclic shift crossover for multi-objective genetic algorithms for the vehicle routing problem with time-windows," *Appl. Soft Comput.*, vol. 52, pp. 863–876, Mar. 2017.
[15] G. B. Alvarenga, G. R. Mateus, and G. de Tomi, "A genetic and set partitioning two-phase approach for the vehicle routing problem with time windows," *Comput. Oper. Res.*, vol. 34, no. 6, pp. 1561–1584, Jun. 2007.
[16] K. Ghoseiri and S. F. Ghannadpour, "Multi-objective vehicle routing problem with time windows using goal programming and genetic algorithm," *Appl. Soft Comput.*, vol. 10, no. 4, pp. 1096–1107, Sep. 2010.
[17] Z. Ursani, D. Essam, D. Cornforth, and R. Stocker, "Localized genetic algorithm for vehicle routing problem with time windows," *Appl. Soft Comput.*, vol. 11, no. 8, pp. 5375–5390, Dec. 2011.
[18] G. Schrimpf, J. Schneider, H. Stamm-Wilbrandt, and G. Dueck, "Record breaking optimization results using the ruin and recreate principle," *J. Comput. Phys.*, vol. 159, no. 2, pp. 139–171, Apr. 2000.
[19] E. Prescott-Gagnon, G. Desaulniers, and L.-M. Rousseau, "A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows," *Networks*, vol. 54, no. 4, pp. 190–204, Dec. 2009.
[20] D. Pisinger and S. Ropke, "A general heuristic for vehicle routing problems," *Comput. Oper. Res.*, vol. 34, no. 8, pp. 2403–2435, Aug. 2007.
[21] Z. Wang, Y. Li, and X. Hu, "A heuristic approach and a tabu search for the heterogeneous multi-type fleet vehicle routing problem with time windows and an incompatible loading constraint," *Comput. Ind. Eng.*, vol. 89, pp. 162–176, 2014.
[22] B. Yu, Z. Z. Yang, and B. Z. Yao, "A hybrid algorithm for vehicle routing problem with time windows," *Expert Syst. Appl.*, vol. 38, no. 1, pp. 435–441, 2011.
[23] Y. Nagata, O. Bräysy, and W. Dullaert, "A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows," *Comput. Oper. Res.*, vol. 37, no. 4, pp. 724–737, Apr. 2010.
[24] Y. Shi, T. Boudouh, and O. Grunder, "An efficient tabu search based procedure for simultaneous delivery and pick-up problem with time window," *IFAC-PapersOnLine*, vol. 51, no. 11, pp. 241–246, 2018.
[25] H. Li and A. Lim, "Local search with annealing-like restarts to solve the VRPTW," *Eur. J. Oper. Res.*, vol. 150, no. 1, pp. 115–127, Oct. 2003.
[26] H. Ben Ticha, N. Absi, D. Feillet, and A. Quilliot, "Multigraph modeling and adaptive large neighborhood search for the vehicle routing problem with time windows," *Comput. Oper. Res.*, vol. 104, pp. 113–126, 2019.

[27] V. Pureza, R. Morabito, and M. Reimann, "Vehicle routing with multiple deliverymen: Modeling and heuristic approaches for the VRPTW," *Eur. J. Oper. Res.*, vol. 218, no. 3, pp. 636–647, May 2012.
[28] P. P. Repoussis, C. D. Tarantilis, and G. Ioannou, "Arc-guided evolutionary algorithm for the vehicle routing problem with time windows," *IEEE Trans. Evol. Comput.*, vol. 13, no. 3, pp. 624–647, Jun. 2009.
[29] K. C. Tan, Y. H. Chew, and L. H. Lee, "A hybrid multiobjective evolutionary algorithm for solving vehicle routing problem with time windows," *Comput. Optim. Appl.*, vol. 34, no. 1, pp. 115–151, May 2006.
[30] N. Labadi, C. Prins, and M. Reghioui, "A memetic algorithm for the vehicle routing problem with time windows," *RAIRO-Oper. Res.*, vol. 42, no. 3, pp. 415–431, Jul. 2008.
[31] P. Moscato, *New Ideas in Optimization*, D. Corne, M. Dorigo, F. Glover, D. Dasgupta, P. Moscato, R. Poli, and K. V. Price, Eds. Maidenhead, U.K.: McGraw-Hill, 1999, pp. 219–234.
[32] J. Zhang, F. Yang, and X. Weng, "An evolutionary scatter search particle swarm optimization algorithm for the vehicle routing problem with time windows," *IEEE Access*, vol. 6, pp. 63468–63485, 2018.
[33] V. Ghilas, E. Demir, and T. Van Woensel, "An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows and scheduled lines," *Comput. Oper. Res.*, vol. 72, pp. 12–30, 2016.
[34] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, vol. 27, no. 2, 1st ed. Boston, MA, USA: Addison-Wesley, 1989.
[35] I. Mihajlovic, Z. Zivkovic, N. Strbac, D. Zivkovic, and A.Jovanovic, "Using genetic algorithms to resolve facility layout problem," *Serbian J. Manag.*, vol. 2, no. 1, pp. 35–46, 2007.
[36] J.-Y. Potvin and J.-M. Rousseau, "A parallel route building algorithm for the vehicle routing and scheduling problem with time windows," *Eur. J. Oper. Res.*, vol. 66, no. 3, pp. 331–340, May 1993.
[37] S.-H. Xu, J.-P. Liu, F.-H. Zhang, L. Wang, and L.-J. Sun, "A combination of genetic algorithm and particle swarm optimization for vehicle routing problem with time windows," *Sensors*, vol. 15, no. 9, pp. 21033–21053, Aug. 2015.
[38] Y. Qi, Z. Hou, H. Li, J. Huang, and X. Li, "A decomposition based memetic algorithm for multi-objective vehicle routing problem with time windows," *Comput. Oper. Res.*, vol. 62, pp. 61–77, Oct. 2015.
[39] Y. Rochat and É. D. Taillard, "Probabilistic diversification and intensification in local search for vehicle routing," *J. Heuristics*, vol. 1, no. 1, pp. 147–167, Sep. 1995.
[40] J. Sripriya, A. Ramalingam, and K. Rajeswari, "A hybrid genetic algorithm for vehicle routing problem with time windows," in *Proc. Int. Conf. Innov. Inf., Embedded Commun. Syst. (ICIIECS)*, Mar. 2015, pp. 1–4.
[41] G. B. Alvarenga, G. R. Mateus, and G. de Tomi, "A genetic and set partitioning two-phase approach for the vehicle routing problem with time windows," *Comput. Oper. Res.*, vol. 34, no. 6, pp. 1561–1584, Jun. 2007.
[42] B. Ombuki, B. J. Ross, and F. Hanshar, "Multi-objective genetic algorithms for vehicle routing problem with time windows," *Appl. Intell.*, vol. 24, no. 1, pp. 17–30, Feb. 2006.
[43] P. Shaw and S. A. Ilog, "Using constraint programming and local search methods to solve vehicle routing problems," in *Proc. Int. Conf. Princ. Pract. Constraint Program.* Berlin, Germany: Springer-Verlag, 1998, pp. 417–431.
[44] J.-F. Cordeau, G. Laporte, and A. Mercier, "A unified tabu search algorithm for vehicle routing problems with soft time windows," *J. Oper. Res. Soc.*, vol. 52, no. 8, pp. 928–936, 2001.
[45] E. T. Yassen, M. Ayob, M. Z. A. Nazri, and N. R. Sabar, "Meta-harmony search algorithm for the vehicle routing problem with time windows," *Inf. Sci.*, vol. 325, pp. 140–158, Dec. 2015.
[46] D. Zhang, S. Cai, F. Ye, Y.-W. Si, and T. T. Nguyen, "A hybrid algorithm for a vehicle routing problem with realistic constraints," *Inf. Sci.*, vols. 394–395, pp. 167–182, Jul. 2017.

THAU-SOON KHOO received the M.Sc. degree in finance from the University of Hull, Hull, U.K., in 1996, and the M.Sc. degree in advanced in IT from the University of Malaysia Sarawak (UNIMAS), Sarawak, Malaysia, in 2012. He is currently pursuing the Ph.D. degree with the University of Tunku Abdul Rahman (UTAR), Selangor, Malaysia. He has over 25 years of IT working experience in different industry. His current interests include big data analytics, artificial intelligence, optimization, software engineering, machine learning, cloud computing, and their applications.

BABRDEL BONAB MOHAMMAD (Member, IEEE) received the Ph.D. degree in philosophy from Universiti Teknologi Malaysia. He was a Postdoctoral Research Fellow with Universiti Tunku Abdul Rahman (UTAR), where he is currently an Assistant Professor of computer science with the Lee Kong Chian Faculty of Engineering and Science, Universiti Tunku Abdul Rahman, Malaysia. His research interests include artificial Intelligence, deep learning, image processing, machine learning, and evolutionary algorithms.

VOON-HEE WONG received the Ph.D. degree in philosophy from Universiti Sains Malaysia (Statistics). He is currently an Assistant Professor and the Head of Department of Mathematical and Actuarial Sciences, Universiti Tunku Abdul Rahman (UTAR).

YONG-HAUR TAY received the Ph.D. degree in philosophy from the University De Nantes. He is currently the Director of Recogine Technology. He serves as an Adjunct Associate Professor with UTAR. His research interests include artificial intelligence, pattern recognition, neural computing, handwriting recognition, machine learning, and neural networks.

MADHAVAN NAIR received the Ph.D. degree in philosophy from the Universiti Putra Malaysia (Nature Tourism). He is currently an Assistant Professor with UTAR and the Head of the Department of Internet Engineering and Computer Science. He serves as a board of moderators of data communicaion and networking, from 2009 to 2014. He is a member of the board of moderator in 2009. He was also a Technical Consultant for software on network security and data storage management, in 2018.