Noname manuscript No.
(will be inserted by the editor)

A multi-start ILS-RVND algorithm with adaptive solution acceptance for the CVRP

Osman Gokalp · Aybars Ugur

Received: date / Accepted: date

Abstract This study proposes a novel hybrid algorithm based on Iterated Local Search (ILS) and Random Variable Neighborhood Descent (RVND) metaheuristics for the purpose of solving the Capacitated Vehicle Routing Problem (CVRP). The main contribution of this work is that two new search rules have been developed for multi-starting and adaptive acceptance strategies, and applied together to enhance the power of the algorithm. A comprehensive experimental work has been conducted on two common CVRP benchmarks. Computational results demonstrate that both multi-start and adaptive acceptance strategies provide a significant improvement on the performance of pure ILS-RVND hybrid. Experimental work also shows that our algorithm is highly effective in solving CVRP and comparable with state-of-the-art.

Keywords Capacitated vehicle routing problem · Iterated local search · Random variable neighborhood descent · Adaptive acceptance function · Multi-start · Hybrid metaheuristic

1 Introduction

Transportation and logistics play an important role in the organization of many real-world operations such as supply chain management, shipment delivery and distribution of a wide variety of goods. However, a major problem with these kinds of operations is that it is a daunting task to produce an efficient and acceptable route plan due to their complex nature. By addressing this issue, Vehicle Routing Problem (VRP) has been an object of research since it was formulated by Dantzig and Ramser (1959) as the truck dispatching problem.

The goal of the VRP is to find a set of routes for a fleet of vehicles which are used for carrying items to customers. The basic version of it is called Capacitated VRP (CVRP) which implies an identical carrying capacity on vehicles that total amount of demands in one route cannot exceed a predefined limit. Many other VRP variants have been derived to meet different real-world requirements. For example, when time window constraints are applied to the delivery schedules, the problem turns into VRP with Time Windows. Moreover, when pickup operations are carried out in addition to deliveries, the problem turns into VRP with Pickup and Delivery. Regardless of their properties, these variants are basically built on top of CVRP. Therefore, developing new efficient methods for CVRP is also beneficial for others.

CVRP is an NP-hard optimization problem, and it is one of the best-known problems in the field of combinatorial optimization. For larger problem sizes, it takes a very long time to produce an optimal solution by exact algorithms. Therefore, as in other hard optimization problems, heuristic and metaheuristic methods are preferred to obtain good results within a reasonable time. Metaheuristics offer a generic framework to the overall optimization process and usually require and manage several low-level heuristics that are specialized for the problem being solved. Although many studies that use

O. Gokalp
Department of Computer Engineering, Ege University, Bornova, Izmir, 35100, Turkey
Tel.: +90-232-3115331
Fax: +90-232-3399405
E-mail: osman.gokalp@ege.edu.tr

A. Ugur
Department of Computer Engineering, Ege University, Bornova, Izmir, 35100, Turkey

various types of metaheuristics for VRP, new contributions in this subject are still highly valued.

Iterated Local Search (ILS) (Stützle 1998) is one of the simple yet efficient metaheuristics for solving combinatorial optimization problems and it has been successfully applied to VRP (see Section 2 for the relevant literature). However, the performance of ILS is highly dependent on the local search method used. Considering CVRP, several local search algorithms having different characteristics are available, and it is better to combine more than one methods to benefit from them all. On the other hand, managing such multiple algorithms requires a higher level search strategy; so it is preferable to use a metaheuristic like Variable Neighborhood Descent (Hansen and Mladenović 2001) (VND). When there is no significant order between the managed algorithms, the randomized version of VND, or RVND, can be used. Due to the many sub-algorithms it contains, RVND exhibits very high exploitation behavior and it is likely to be trapped in local optima. Contrarily, ILS offers more randomized search with its perturbation phase and it has high exploration capability. Together these two algorithms are complementary and hybridization of the algorithms is expected to outperform their individual performances.

In this study, we propose a novel ILS-RVND hybrid metaheuristic that local search part of the ILS is managed by RVND. RVND itself consists of 6 low-level local search algorithms that are based on an efficient and fast search method called sequential search (Irnich et al. 2006). As the main contribution of this work, the performance of the ILS-RVND hybrid is further improved by adapting both the multi-starting rule and adaptive acceptance function to CVRP for the first time in the literature. These two strategies contribute to the exploration behavior of the general search process and help algorithm escape from local optima.

The rest of this paper is organized in the following way. Section 2 summarizes the relevant literature of this study. Section 3 first defines the CVRP problem and then gives the definition of ILS and RVND which are the two main algorithms that have been used in this work. The proposed algorithm and its components are detailed in Section 4. Then Section 5 presents the experimental work that has been conducted to analyze the performance of the proposed algorithm using common benchmark problems. Finally, Section 6 concludes the paper.

2 Related work

Dantzig and Ramser (1959) were first formulated Vehicle Routing Problem (VRP) as Truck Dispatch Problem. Since then many variants of this problem have been solved such as VRP with time windows (Nalepa and Blocho 2016), VRP with stochastic demands (Gee et al. 2016), VRP with simultaneous delivery and pick-up (Dethloff 2001), heterogeneous fleet VRP (Penna et al. 2013), multi-depot VRP (Agrawal et al. 2017) and open VRP (Fu et al. 2005).

Since VRP is an NP-hard problem (Lenstra and Kan 1981), no exact algorithm can solve it in polynomial time. Hence, metaheuristic algorithms are often applied to produce high-quality solutions in a reasonable time.

In recent years, there has been an increasing amount of literature on multi-start metaheuristic solutions for the VRP. In one of these studies, Subramanian et al. (2013) proposed a hybrid algorithm combining the Set Partitioning (SP) formulation with an ILS-RVND for a class of Vehicle Routing Problems with homogeneous fleet. Michallet et al. (2014) developed a multi-start ILS algorithm for the periodic vehicle routing problem with time window. The other two studies that employed multi-start ILS were carried out by Rivera et al. (2015) and Sassi et al. (2015). The former addressed the multi-trip cumulative capacitated VRP while the latter dealt with the VRP with mixed fleet of conventional and heterogeneous electric vehicles. Using Particle Swarm Optimization, Psychas et al. (2017) developed a Parallel Multi-Start algorithm to solve the Multi-objective Route-based Fuel Consumption Vehicle Routing problem. Another example is (Molina et al. 2018) in which the authors designed a Multi-start metaheuristic algorithm with an intelligent neighborhood selection method for the multi-objective humanitarian vehicle routing problems.

As for adaptive acceptance methods, there has been a limited amount of research that addresses the VRP. Tarantilis et al. (2002) proposed an adaptive algorithm based on Threshold Accepting (TA) method of Dueck and Scheuer (1990), which is a simpler variant of well-known Simulated Annealing metaheuristic (Kirkpatrick et al. 1983). Alabas-Uslu and Dengiz (2011) introduced a self-adaptive local search algorithm based on adaptive acceptance parameter, which was adjusted by the number of iterations passed and the amount of improvement over the starting solution. This strategy was later modified and adopted by Avci and Topaloglu (2015) as a part of the VND based algorithm to solve the VRP with simultaneous pick-up and delivery.

Together, these studies outline that multi-start and adaptive acceptance search strategies are preferable for solving the VRP. However, to the best of our knowledge, this is the first paper that combines both strategies to improve the overall solution quality. The main reason

why we address the CVRP in this work is that it is the basic version of the problem, so the developed methods will have a great potential to be adapted for the other VRP variants.

3 The background

3.1 Problem definition

The definition of the CVRP can be given as follows: Let $G = (V, E)$ is an undirected and complete graph, where, $V = \{0, 1, 2, ..., n\}$ is the vertex set and $E = \{(i, j) \mid i, j \in V \wedge i \neq j\}$ is the edge set. In set V, the vertex 0 represents the depot, and the vertices that are numbered from 1 to n represent the customers. Each customer has a demand q and each vehicle has an identical capacity Q. A route $R = (0, v_1, v_2, \cdots, v_l, 0)$ starts from and end at the depot by visiting the l customers $v_1, v_2, \cdots, v_l$ in order, and it is capacity feasible if $\sum_{i=1}^{l} q_{v_i} \leq Q$. The weight of an edge $(i, j) \in E$ is calculated as a Euclidean distance between the vertices i and j, and is denoted by $d_{i,j}$. Hence, the total distance of a route is calculated by $D = d_{0,v_1} + \sum_{i=1}^{l-1}(d_{v_i,v_{i+1}}) + d_{v_l,v_0}$.

A CVRP solution consists of $k \geq 1$ feasible routes that each customer is visited once in one of these routes. The objective is to minimize the total route lengths of a solution. Moreover, in some of the problem instances, a predefined service time s is considered during customer visits. In this case, it is expected that the summation of route distance and service time does not exceed a predefined limit L, so, the inequality $D + \sum_{i=1}^{l} s \leq L$ is satisfied for each route in a solution.

3.2 Iterated local search

Iterated Local Search (ILS) (Stützle 1998) is one of the main metaheuristic algorithms for optimization problems. As given in Algorithm 1, it is based on local search and perturbation steps that are applied repetitively one after another until the termination condition is satisfied. Local search procedure uses one or more neighborhood structures, which are dependent on the problem to be solved, for modifying the current solution systematically to find a better one. On the other hand, the aim of the perturbation step is to escape from local optimum by changing current solution in a random way so that the search process continues from a new point in the solution space. Generally, a random move for a perturbation is generated using a larger neighborhood structure than that of used in a local search.

Algorithm 1: ILS

```
   output: S*
1  S ← constructInitialSolution();
2  S* ← localSearch(S);
3  while termination condition is not satisfied do
4  |  S' ← perturb(S*);
5  |  S'' ← localSearch(S');
6  |  if S'' is better than S* then
7  |  |  S* ← S'';
8  |  end
9  |  if acceptance(S'') then
10 |  |  S ← S'';
11 |  end
12 end
```

3.3 Random variable neighborhood descent

Variable neighborhood descent (VND) is a deterministic metaheuristic which is used for local search. The classical version of this algorithm (Hansen and Mladenović 2001) uses a set of neighborhood structures $N = \{N_1, \cdots, N_k\}$ sequentially until the solution S becomes locally optimum for each of them. When the order of neighborhoods are decided in a random way, the algorithm is called RVND (see Algorithm 2). From the algorithm, we can see that the counter n is increased by one to try next neighbor if the current solution is not improved. Otherwise, the better solution is found, and n is set to 1 so that the cycle restarts for the newly accepted solution. When none of the neighbors make an improvement in succession, which implies $n > k$, the algorithm returns the current solution. The only parameter of the algorithm is *bi* which determines whether the best improving solution is searched for a given neighborhood.

Algorithm 2: RVND

```
   input     : S, N
   output    : S
   parameter: bi
1  {i_j}_{j=1}^{k} ← randomIndexes(1, k);
2  n ← 1;
3  while n ≤ k do
4  |  if bi=true then
5  |  |  S' ← bestImprovingSolution(N_{i_n}(S));
6  |  else
7  |  |  S' ← firstImprovingSolution(N_{i_n}(S));
8  |  end
9  |  if S' is better than S then
10 |  |  S ← S';
11 |  |  n ← 1;
12 |  else
13 |  |  n ← n + 1;
14 |  end
15 end
```

4 The proposed algorithm

In the proposed algorithm, called MA_ILS-RVND, we hybridize multi-start ILS and RVND metaheuristics in such a way that the local search part of ILS is managed by the RVND. For the acceptance function, we set an adaptive threshold that allows unimproved solutions to be accepted to some degree. As for the multi-start strategy, we set a restart limit that the search process starts from scratch when the number of successive unimproved solutions reaches it.

The details of the MA_ILS-RVND can be seen in Algorithm 3. In the initialization stage, restart limit ($rstLimit$) and perturbation size ($pSize$) values are given as to be proportional to the n and $\sqrt{n}$, respectively, where n is the number of customers. After the starting solution (S_z) is constructed using the parallel version of Clarke and Wright (1964) heuristic, it is assigned as an initial value of the current solution (S), best solution after restart (S^*) and the overall best solution (S_{best}). Then the three counters, which are the iteration counter from the beginning ($iterCounter$), the iteration count after the latest restart (i), the successive unimproved solutions after the latest restart, are set to zero.

The main loop of the algorithm continues until the maximum number of iterations is reached. In each iteration, firstly, the current solution S is randomly changed by the *perturb* procedure, and the S' is obtained. Secondly, the local search procedure takes S' as an input and outputs the solution S''. Then, three counters are increased by one, and the algorithm moves on the acceptance stage.

If the fitness of the S'' is smaller than the S, it is directly accepted as a new current solution. If S'' is also smaller than S^*, it is updated, and the $rstCount$ is reset. Moreover, if S'' improves the overall best solution since the beginning of the algorithm, S_{best} is also updated. On the other hand, if the fitness of the S'' is not smaller than the S, it can be accepted in case of its fitness is equal to or below the threshold which is calculated as the multiplication of θ and the $f(S^*)$. The value of θ is dependent on α_1 and α_2 that the former increases as the S^* is improved and the latter decreases as the i increases. This adaptive threshold based solution acceptance mechanism is taken from SALS algorithm of Alabas-Uslu and Dengiz (2011) and adapted for our algorithm.

At the last stage of each iteration, it is checked whether the number of successive unimproved solutions reaches its limit. In case of limit exceed, S and S^* are reset back to the starting solution. Also the counters i and $rstCounter$ are set back to the zero. Therefore, the algorithm restarts by the next iteration.

Algorithm 3: Multi-start ILS-RVND algorithm with adaptive acceptance (MA_ILS-RVND)

```
   input     : MAX_ITER
   output    : S_best
   parameter: pFactor,rstFactor,k,bi
 1 pSize ← ⌊pFactor * √n⌋;
 2 rstLimit ← ⌊rstFactor * n⌋;
 3 S_z ← initialSolution() ; ▷ using Clarke and Wright heuristic
 4 S ← S* ← S_best ← S_z;
 5 iterCounter ← 0;
 6 i ← 0;
 7 rstCounter ← 0;
 8 while iterCounter < MAX_ITER do
 9     S' ← perturbation(S, pSize) ;      ▷ using random CROSS-Exchange
10     S'' ← localSearch(S', k, bi) ;      ▷ using RVND
11     rstCounter ← rstCounter + 1;
12     iterCounter ← iterCounter + 1;
13     i ← i + 1;
14     if f(S'') < f(S) then
15         S ← S'';
16         if f(S'') < f(S*) then
17             S* ← S'';
18             rstCounter ← 0;
19             if f(S'') < f(S_best) then
20                 S_best ← S'';
21             end
22         end
23     else
24         α1 ← f(S*)/f(S_z);
25         α2 ← 1/i;
26         θ ← 1 + α1 * α2;
27         if f(S'') ≤ θ * f(S*) then      ▷ accept unimproved solution
28             S ← S'';
29         end
30     end
31     if rstCounter = rstLimit then      ▷ restart search
32         S ← S_z;
33         S* ← S_z;
34         i ← 0;
35         rstCounter ← 0;
36     end
37 end
```

The parameters of the MA_ILS-RVND are listed below. The first two parameters are not directly used in the algorithm; instead, they are designed as factors. The rationale behind this idea is that $pSize$ and $rstLimit$ are dependent on the size of a problem. In the preliminary experiments we have seen that the $pSize$ and $rstLimit$ are dependent on $\sqrt{n}$ and n, respectively.

- **pFactor** is used to determine the perturbation size ($pSize$) which is dependent on the number of customers.

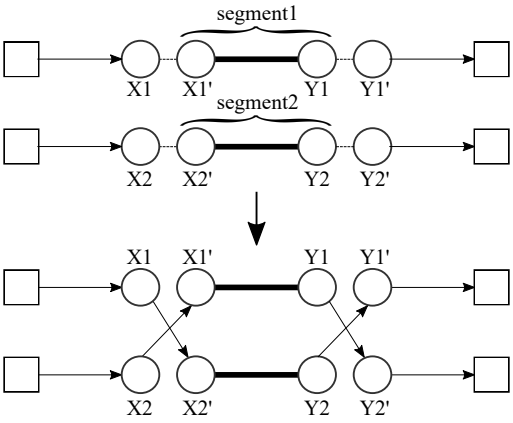

Fig. 1 The CROSS-Exchange move. Rectangle denotes the depot node, whereas circle denotes a customer node.

- **rstFactor** is used to determine the restart limit ($rstLimit$) which is dependent on the number of customers.
- **k** is used to limit segment lengths in local search for $Or-opt$ and $string-exchange$ neighborhoods.
- **bi** is used in the local search to decide which search strategy is applied. If it is $true$, best improving strategy is used, whereas if it is $false$, first improving strategy is preferred.

4.1 Perturbation

We use CROSS-Exchange (CE) neighborhood structure of Taillard et al. (1997) in the perturbation step of the algorithm that two customer sequences, or segments, are exchanged between two routes. Fig. 1 illustrates a CE move that two nodes $X1$ and $Y1$ are selected from the first route, whereas two other nodes $X2$ and $Y2$ are selected from the second route. After removing the edges between these four customers and their successors, new edges $(X1, X2')$, $(X2, X1')$, $(Y1, Y2')$ and $(Y2, Y1')$ are introduced to complete exchange operation. One of the special cases of CE is Or-opt move that occurs when one of the segments is empty (e.g., $X1 = Y1$ or $X2 = Y2$). The other case is 2-opt* that occurs when both the $Y1$ and $Y2$ are located just before the depot.

In order to be used as a perturbation procedure, we designed Algorithm 4 that makes random changes on a given solution S based on CE move type. The role of $pSize$ in the algorithm is to limit the highest value that segment lengths can take. The algorithm handles 3 different cases, namely exchange of two segments between $R1$ and $R2$, movement of a segment of $R1$ to $R2$ and movement of a segment of $R2$ to $R1$, that arise depending on which nodes are selected from the routes. In the case of two empty segments, there will be no change and the algorithm goes back to step 2 to try different segments next time. After successfully applying random CE changes on a solution, the solution is checked for both capacity and distance constraints. In case of feasibility the perturbed solution S' is returned, otherwise, the algorithm returns back to step 1.

Algorithm 4: perturbation

```
1  S' ← make a copy of S;
2  {R1, R2} ← select two different routes from S randomly;
3  X1 ← select a node from R1 randomly (either depot or customer);
4  X2 ← select a node from R2 randomly (either depot or customer);
5  X1' ← successor of X1;
6  X2' ← successor of X2;
7  Set1 ← select up to pSize successors (only customers) of X1;
8  Y1 ← select a random node from {X1} ∪ Set1;
9  Set2 ← select up to pSize successors (only customers) of X2;
10 Y2 ← select a random node from {X1} ∪ Set2;
11 Y1' ← successor of Y1;
12 Y2' ← successor of Y2;
13 if X1 ≠ Y1 and X2 ≠ Y2 then                ▷ two segments exchange
14 |  remove edges (X1, X1'), (Y1, Y1'), (X2, X2') and (Y2, Y2');
15 |  add edges (X1, X2'), (X2, X1'), (Y1, Y2') and (Y2, Y1');
16 else if X1 = Y1 and X2 ≠ Y2 then          ▷ a segment move to R1
17 |  remove edges (X1, X1'), (X2, X2') and (Y2, Y2');
18 |  add edges (X2, Y2'), (X1, X2') and (Y2, X1');
19 else if X1 ≠ Y1 and X2 = Y2 then          ▷ a segment move to R2
20 |  remove edges (X1, X1'), (Y1, Y1') and (X2, X2');
21 |  add edges (X1, Y1'), (X2, X1') and (Y1, X2');
22 else                                      ▷ no changes
23 |  go to step 2;
24 end
25 if S' is both capacity feasible and distance feasible then
26 |  return S';
27 else
28 |  go to step 1;
29 end
```

4.2 Local search

Local search is an important component in optimization algorithms and plays a key role in exploitation. The aim of the local search is to improve a given solution by trying new solutions around its neighborhood, which is dependent on a problem to be solved. Considering the CVRP, it is not enough to use a single local search algorithm, instead, several ones are employed together to find better solutions.

In this study, we use several improvement heuristics based on the *sequential search* that has been presented by Irnich et al. (2006) as a generic local search technique. This method eliminates the unpromising part of neighborhoods, so, compared to the lexicographic search, it can provide significant time savings especially for large size problem instances. An implementation of the algorithms that are designed by this technique is not a trivial task and requires some specialized data structures to be used. So, we followed the guidelines in the original paper and made use of a *giant tour* for the representation of CVRP solutions. An example CVRP solution, which consists of $k = 3$ routes and $n = 8$ customers and its *giant tour* representation are given in Fig. 2. For the giant tour representation, all the routes are linked in a circular way by adding k copies of the depot between them. In order to distinguish between the copies of the depot, the numbers $(n+1, n+2, \cdots, n+k)$ are assigned to each of them.

We list below the neighborhoods that were employed in the local search procedure. These neighborhoods are managed by RVND (see Section 2 for details) that uses the *sequential search* to find improving neighbors. Illustrations of move type for each neighborhood structure are given in Fig. 3.

- **2-opt**: inverts a node sequence *giant tour* by removing and adding two edges.
- **special 2-opt***: splits a *giant tour* into two sub tours by removing and adding two edges.
- **Or-opt**: relocates a node sequence of length at most k.
- **swap**: swaps two nodes.
- **string-exchange**: exchanges two customer sequences of length at most k by inverting their sequence. In this implementation, depots are not allowed to be in any sequences.

5 Experimental work

5.1 Experimental setting

The proposed MA_ILS-RVND algorithm was implemented in Java, and ran on a computer with the configuration of Intel® Core™ i7 6700 3.40 GHz CPU using a single core.

We used Christofides et al. (1979) dataset for preliminary experiments because it is one of the most used benchmarks in the literature and has moderate size problem instances. The other computational experiments and literature comparison were done using Uchoa et al. (2017) dataset due to its comprehensive structure.

Table 1 The tuned parameter values of MA_ILS-RVND per benchmark set using irace

Parameter Name	Tuning Interval	Tuned for Christofides et al.	Tuned for Uchoa et al.
pFactor	[0.25, 2.0]	0.54	1.04
rstFactor	[1, 16]	8	9
k	[2, 5]	4	4
bi	(true, false)	false	false

For the purpose of parameter tuning, we used the irace Package (López-Ibáñez et al. 2016), which is an automated configuration tool for optimization algorithms. To set up irace environment, we used default configurations except for the maximum tuning budget which is set to 1000 iterations. In addition, the maximum iteration count of MA_ILS-RVND is fixed as 20,000 to achieve parameter tuning in a reasonable time. As for training and test problems, we divided a dataset into two pieces that training part consists of odd-numbered instances while test part consists of even-numbered instances. The values that were obtained after parameter tuning are given in Table 1.

Following the literature, the performances of the algorithms were measured by the average gap in percentage for each algorithm-dataset pairs as follows:

$$Avg.\ Gap = \frac{\sum_{i=1}^{n}(\frac{f_i - f_i^*}{f_i^*} \times 100)}{n}$$

where n is the number of problem instances in a dataset, i is the number of a problem, f is the minimum cost value obtained by an algorithm, and f^* is the cost value of the best-known solution. In addition to the average gaps, elapsed CPU times were also measured for each run of the proposed algorithm.

It is worth mentioning that two different rules are applied when calculating distances, or costs, in CVRP literature: (i) using rounded integer numbers, and (ii) using double-precision real numbers. Generally the former is preferred by exact algorithms, whereas the latter is used in (meta)heuristic algorithms. Following the conventions, we used double-precision costs for Christofides et al. dataset. Nevertheless, we used rounded integer costs for Uchoa et al. as its authors prefer to follow the TSPLIB convention.

5.2 Preliminary experiments on Christofides et al. dataset

This part of the paper investigates the effect of multi-start and adaptive acceptance strategies on the performance of the algorithm. Also, we analyze time and

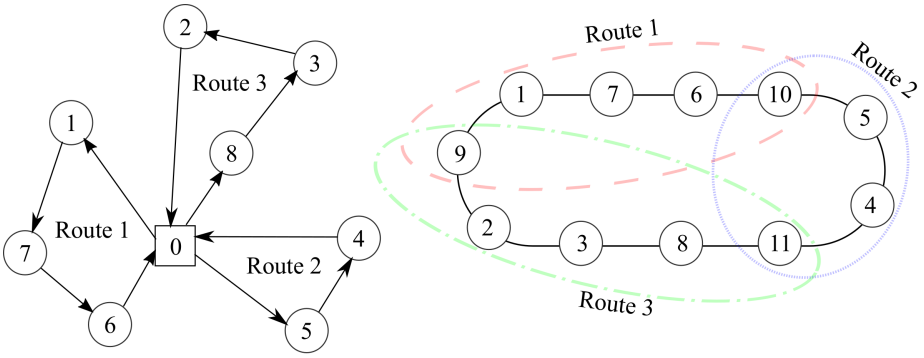

Fig. 2 An example CVRP solution (left) and its giant tour representation (right). Nodes 9, 10 and 11 are the artificial depots that are added to separate routes.

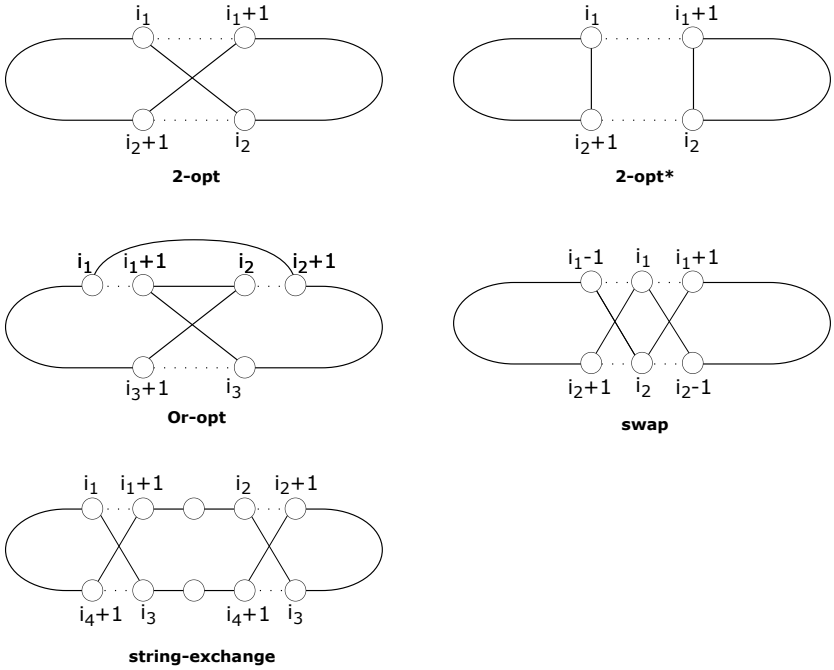

Fig. 3 Illustrations of move type for each neighborhood structure that is used in local search.

cost performances of the algorithm under different maximum iteration budgets.

In preliminary experiments, we used Christofides et al. (1979) dataset includes 14 problem instances that vary in customer size between 50 and 199. The dataset can be divided into two groups by the distribution of customers that instances 1-10 have randomly distributed customers, whereas instances 11-14 have clustered customer positions. Moreover, instances 6-10 and 13-14 require service times and have maximum route duration constraint.

5.2.1 Effect of multi-start and adaptive acceptance strategies

This part of the experimental work analyzes the effect of multi-start and adaptive acceptance strategies that have been used in the proposed algorithm. To do so, we compare the following algorithms:

- **ILS-RVND**: Simple ILS-RVND hybrid without multi-start and adaptive acceptance strategies.
- **M_ILS-RVND**: ILS-RVND hybrid with multi-start strategy.
- **A_ILS-RVND**: ILS-RVND hybrid with adaptive acceptance strategy.
- **MA_ILS-RVND**: ILS-RVND with multi-start and adaptive acceptance strategies (the proposed algorithm).

Table 2 lists the computational results of the algorithms using different combinations of multi-start and adaptive acceptance strategies. Each algorithm was run for 4×10^4 iterations. Reported cost values are based on 10 runs per instance. Avg. time is the average time

elapsed between the start and termination of an algorithm per problem instance. The results show that the proposed MA_ILS-RVND algorithm achieves the best average gap by 0.13% and outperforms the others. It can also be seen that both M_ILS-RVND and A_ILS-RVND achieves lower average gaps than that of simple ILS-RVND hybrid. Considering average times that algorithms spent, the additional cost of multi-start strategy is a bit larger than that of adaptive acceptance. However, these additional time cost seems to be acceptable compared to the gain in average gaps.

In addition, we performed a statistical significance test whether the reported average gaps of these algorithms are statistically significant. To this end, we used two-sided Wilcoxon signed rank test between the proposed algorithm and the other three. The test has shown that all the obtained p values are smaller than the significance level of $\alpha = 0.05$. Therefore, we can conclude that using both of the multi-start and adaptive acceptance strategies makes a contribution to the performance of the MA_ILS-RVND algorithm.

5.2.2 Effect of maximum number of iterations

In metaheuristic optimization, there is a trade-off between the quality of a solution and the time spent to find that solution. That is, the more the algorithm runs, the more the solution is improved. In our algorithm, the execution time is determined by the maximum number iterations (MAX_ITER) that each iteration covers one perturbation and one local search steps of the ILS.

This subsection analyzes the effect of the maximum number of iterations on the performance of the proposed algorithm. The algorithm was run for 1×10^4, 2×10^4, 4×10^4, 8×10^4 and 16×10^4 iterations, and the computational results are listed in Table 3. As expected, the average gap performance of the MA_ILS-RVND improves as the number of iterations increases, and it reaches to 0.07% for the 16×10^4 iterations. However, this improvement comes at a price that the computational time increases to 9.94 minutes.

The behavior of the algorithm under increasing maximum number of iterations in terms of the average gap and average time are illustrated together in Fig 4. It can be seen that the average time increases almost linearly with the number of iterations. On the other hand, the average gap decreases by a decreasing rate as the algorithm converges. Hence, applying the same amount of increase on the MAX_ITER provides more improvement for smaller iterations than those of larger ones.

5.3 Computational results on Uchoa et al. dataset

The final computational experiment is conducted on Uchoa et al. (2017) dataset, which is a relatively new CVRP benchmark that includes 100 instances ranging from 100 to 1000 customers. Since the dataset has been generated by considering several attributes such as depot positioning, customer positioning, demand distribution, and average route size, it is highly comprehensive and powerful for measuring the performance of CVRP optimization algorithms. For the comparison of the proposed algorithm with the literature, we have selected two state-of-the-art metaheuristics that have used this dataset: ILS-SP of Subramanian et al. (2013) and UHGS of Vidal et al. (2012).

Due to the size of Uchoa et al. dataset, computational results of the proposed MA_ILS-RVND algorithm are divided into two parts that Table 4 reports the results for the first 50 instances (100-330 customers), and Table 5 reports the results for the remaining instances (335-1000 customers). All the experiments for the MA_ILS-RVND algorithm are based on 10 runs and 8×10^4 maximum number of iterations. In addition, the reported time for each instance is the time elapsed between the start and termination of the algorithm using a single CPU core, and it is presented in minutes. As for the two algorithms used in the comparison, numerical results were obtained from (Uchoa et al. 2017).

The computational results show that our MA_ILS-RVND algorithm could achieve average gaps of 0.54% in terms of average cost values, and is highly effective for solving CVRP. As for best cost values, the algorithm could achieve the average gap of 0.37%. Moreover, it could reach 13 best-known solutions in terms of the best costs. When the average gaps are compared with those of the other two state-of-the-art algorithms, it is seen that the proposed algorithm takes second place after the UHGS (0.27%) and before the ILS-SP (0.61%). To compare the time performance of the algorithms, we used estimated GFlop performances of the CPUs and obtained scaled times with respect to the computer that was used in this study. The results reveal that in order to reach the reported average gaps, our algorithm consumes the least average time compared to others. Overall, these results suggest that the proposed algorithm is comparable with the state-of-the-art.

6 Conclusion

This study has presented a new metaheuristic algorithm, called MA_ILS-RVND, to solve the CVRP. The proposed algorithm is based on the hybridization of

Table 2 Performance comparison of the ILS-RVND algorithms using different combinations of multi-start and adaptive acceptance strategies

Instance(n)	ILS-RVND	A_ILS-RVND	M_ILS-RVND	MA_ILS-RVND	BKS
C1(50)	**524.61**	**524.61**	**524.61**	**524.61**	524.61
C2(75)	837.74	839.75	835.61	**835.26**	835.26
C3(100)	827.39	827.63	827.27	826.27	826.14
C4(150)	1031.46	1030.65	1029.30	1029.24	1028.42
C5(199)	1305.12	1306.63	1300.12	1299.59	1291.29
C6(50)	558.59	557.56	**555.43**	**555.43**	555.43
C7(75)	920.80	914.56	912.76	912.83	909.68
C8(100)	869.42	869.61	**865.94**	**865.94**	865.94
C9(150)	1174.67	1170.76	1164.71	1163.75	1162.55
C10(199)	1410.79	1405.83	1407.97	1402.23	1395.85
C11(120)	**1042.11**	**1042.11**	**1042.11**	**1042.11**	1042.11
C12(100)	**819.56**	**819.56**	**819.56**	**819.56**	819.56
C13(120)	1543.92	1547.39	1543.13	1542.88	1541.14
C14(100)	**866.37**	**866.37**	**866.37**	**866.37**	866.37
Avg. Gap	0.45%	0.38%	0.18%	0.13%	
Avg. Time	2.52 min.	2.65 min.	2.81 min.	2.82 min.	
Wilcox. S. R. (vs. MA_ILS-RVND)	p =0.002	p =0.002	p =0.023		

The number of customers and best-known solutions are denoted by n and BKS, respectively. Values in boldface indicate that the cost of a best-known solution is reached.

Table 3 Performance comparison of the MA_ILS-RVND algorithm running with different maximum number of iterations

Instance(n)	1×10^4 iter.	2×10^4 iter.	4×10^4 iter.	8×10^4 iter.	16×10^4 iter.	BKS
C1(50)	**524.61**	**524.61**	**524.61**	**524.61**	**524.61**	524.61
C2(75)	836.34	835.49	**835.26**	**835.26**	**835.26**	835.26
C3(100)	826.71	826.39	826.27	**826.14**	**826.14**	826.14
C4(150)	1030.41	1028.75	1029.24	1029.04	1028.95	1028.42
C5(199)	1303.43	1301.88	1299.59	1295.38	1293.76	1291.29
C6(50)	**555.43**	**555.43**	**555.43**	**555.43**	**555.43**	555.43
C7(75)	912.89	912.89	912.83	912.51	911.48	909.68
C8(100)	**865.94**	**865.94**	**865.94**	**865.94**	**865.94**	865.94
C9(150)	1169.56	1168.56	1163.75	1163.20	1162.68	1162.55
C10(199)	1408.26	1404.93	1402.23	1401.89	1401.31	1395.85
C11(120)	**1042.11**	**1042.11**	**1042.11**	**1042.11**	**1042.11**	1042.11
C12(100)	**819.56**	**819.56**	**819.56**	**819.56**	**819.56**	819.56
C13(120)	1543.30	1543.06	1542.88	1542.86	1542.86	1541.14
C14(100)	**866.37**	**866.37**	**866.37**	**866.37**	**866.37**	866.37
Avg. Gap	0.24%	0.18%	0.13%	0.09%	0.07%	
Avg. Time	0.66 min.	1.28 min.	2.82 min.	5.37 min.	9.94 min.	

The number of customers and best-known solutions are denoted by n and BKS, respectively. Values in boldface indicate that the cost of a best-known solution is reached.

ILS and RVND metaheuristics and make use of multi-start and adaptive acceptance search strategies. RVND takes part in the local search of the ILS and manages 5 low-level neighborhood structures using the sequential search strategy. An acceptance function accepts not only improving solutions but also unimproved ones, in order to prevent the premature convergence of the algorithm. Here, the acceptance level is determined by an adaptive threshold value, which is maintained during the search process. The multi-starting strategy restarts the algorithm in case of local optima, which is decided when the number of rejected solutions reaches a certain limit.

Comprehensive experiments have been conducted on common CVRP benchmarks to analyze the performance of the proposed algorithm. Preliminary ex-

Table 4 Computational results of the MA_ILS-RVND algorithm for Uchoa et al. dataset and comparison with current state-of-the-art algorithms (part I)

Instance	ILS-SP			UHGS			MA_ILS-RVND			BKS
(#)Name	Avg.	Best	Time	Avg.	Best	Time	Avg.	Best	Time	
(1)X-n101-k25	27,591.0	27,591	0.13	27,591.0	27,591	1.43	27,793.0	27,793	1.69	27,591
(2)X-n106-k14	26,375.9	26,362	2.01	26,381.8	26,378	4.04	26,381.8	26,381	4.35	26,362
(3)X-n110-k13	14,971.0	14,971	0.20	14,971.0	14,971	1.58	14,971.0	14,971	1.51	14,971
(4)X-n115-k10	12,747.0	12,747	0.18	12,747.0	12,747	1.81	12,747.0	12,747	2.00	12,747
(5)X-n120-k6	13,337.6	13,332	1.69	13,332.0	13,332	2.31	13,332.0	13,332	4.67	13,332
(6)X-n125-k30	55,673.8	55,539	1.43	55,542.1	55,539	2.66	55,981.2	55,960	3.96	55,539
(7)X-n129-k18	28,998.0	28,948	1.92	28,948.5	28,940	2.71	28,947.0	28,940	3.98	28,940
(8)X-n134-k13	10,947.4	10,916	2.07	10,934.9	10,916	3.32	10,922.7	10,916	4.40	10,916
(9)X-n139-k10	13,603.1	13,590	1.60	13,590.0	13,590	2.28	13,590.0	13,590	2.42	13,590
(10)X-n143-k7	15,745.2	15,726	1.64	15,700.2	15,700	3.10	15,726.2	15,726	5.51	15,700
(11)X-n148-k46	43,452.1	43,448	0.84	43,448.0	43,448	3.18	43,457.3	43,448	3.48	43,448
(12)X-n153-k22	21,400.0	21,340	0.49	21,226.3	21,220	5.47	21,385.8	21,366	7.95	21,220
(13)X-n157-k13	16,876.0	16,876	0.76	16,876.0	16,876	3.19	16,876.0	16,876	8.36	16,876
(14)X-n162-k11	14,160.1	14,138	0.54	14,141.3	14,138	3.32	14,168.3	14,147	3.60	14,138
(15)X-n167-k10	20,608.7	20,562	0.86	20,563.2	20,557	3.73	20,602.9	20,562	7.41	20,557
(16)X-n172-k51	45,616.1	45,607	0.64	45,607.0	45,607	3.83	45,621.7	45,607	4.55	45,607
(17)X-n176-k26	48,249.8	48,140	1.11	47,957.2	47,812	7.56	48,702.7	48,398	13.05	47,812
(18)X-n181-k23	25,571.5	25,569	1.59	25,591.1	25,569	6.28	25,575.6	25,569	9.05	25,569
(19)X-n186-k15	24,186.0	24,145	1.72	24,147.2	24,145	5.92	24,155.2	24,147	7.66	24,145
(20)X-n190-k8	17,143.1	17,085	2.10	16,987.9	16,980	12.08	17,008.3	16,989	19.91	16,980
(21)X-n195-k51	44,234.3	44,225	0.87	44,244.1	44,225	6.10	44,293.0	44,264	4.49	44,225
(22)X-n200-k36	58,697.2	58,626	7.48	58,626.4	58,578	7.97	58,868.1	58,729	10.62	58,578
(23)X-n204-k19	19,625.2	19,570	1.08	19,571.5	19,565	5.35	19,642.4	19,584	5.67	19,565
(24)X-n209-k16	30,765.4	30,667	3.80	30,680.4	30,656	8.62	30,721.5	30,698	11.78	30,656
(25)X-n214-k11	11,126.9	10,985	2.26	10,877.4	10,856	10.22	10,922.6	10,898	8.55	10,856
(26)X-n219-k73	117,595.0	117,595	0.85	117,604.9	117,595	7.73	117,597.2	117,595	16.72	117,595
(27)X-n223-k34	40,533.5	40,471	8.48	40,499.0	40,437	8.26	40,588.7	40,524	7.40	40,437
(28)X-n228-k23	25,795.8	25,743	2.40	25,779.3	25,742	9.80	25,998.1	25,879	11.62	25,742
(29)X-n233-k16	19,336.7	19,266	3.01	19,288.4	19,230	6.84	19,313.5	19,276	6.97	19,230
(30)X-n237-k14	27,078.8	27,042	3.46	27,067.3	27,042	8.90	27,068.3	27,042	13.88	27,042
(31)X-n242-k48	82,874.2	82,774	17.83	82,948.7	82,804	12.42	82,996.6	82,934	13.70	82,751
(32)X-n247-k50	37,507.2	37,289	2.06	37,284.4	37,274	20.41	37,837.7	37,673	21.57	37,274
(33)X-n251-k28	38,840.0	38,727	10.77	38,796.4	38,699	11.69	38,860.9	38,828	12.94	38,684
(34)X-n256-k16	18,883.9	18,880	2.02	18,880.0	18,880	6.52	18,884.8	18,880	8.87	18,839
(35)X-n261-k13	26,869.0	26,706	6.67	26,629.6	26,558	12.67	26,718.9	26,660	19.67	26,558
(36)X-n266-k58	75,563.3	75,478	10.03	75,759.3	75,517	21.36	75,817.8	75,712	18.46	75,478
(37)X-n270-k35	35,363.4	35,324	9.07	35,367.2	35,303	11.25	35,406.1	35,358	9.97	35,291
(38)X-n275-k28	21,256.0	21,245	3.59	21,280.6	21,245	12.04	21,274.1	21,250	13.81	21,245
(39)X-n280-k17	33,769.4	33,624	9.62	33,605.8	33,505	19.09	33,669.9	33,583	16.87	33,503
(40)X-n284-k15	20,448.5	20,295	8.64	20,286.4	20,227	19.91	20,391.4	20,331	16.27	20,226
(41)X-n289-k60	95,450.6	95,315	16.11	95,469.5	95,244	21.28	95,614.8	95,507	21.51	95,151
(42)X-n294-k50	47,254.7	47,190	12.42	47,259.0	47,171	14.70	47,335.3	47,274	10.32	47,161
(43)X-n298-k31	34,356.0	34,239	6.92	34,292.1	34,231	10.93	34,416.7	34,317	12.04	34,231
(44)X-n303-k21	21,895.8	21,812	14.15	21,850.9	21,748	17.28	21,890.8	21,845	15.08	21,744
(45)X-n308-k13	26,101.1	25,901	9.53	25,895.4	25,859	15.31	26,091.5	25,949	28.10	25,859
(46)X-n313-k71	94,297.3	94,192	17.50	94,265.2	94,093	22.41	94,477.4	94,388	22.81	94,044
(47)X-n317-k53	78,356.0	78,355	8.56	78,387.8	78,355	22.37	78,363.0	78,355	43.19	78,355
(48)X-n322-k28	29,991.3	29,877	14.68	29,956.1	29,870	15.16	30,025.5	29,971	11.56	29,834
(49)X-n327-k20	27,812.4	27,599	19.13	27,628.2	27,564	18.19	27,773.2	27,706	21.58	27,532
(50)X-n331-k15	31,235.5	31,105	15.70	31,159.6	31,103	24.43	31,165.9	31,105	30.27	31,102

Table 5 Computational results of the MA_ILS-RVND algorithm for Uchoa et al. dataset and comparison with current state-of-the-art algorithms (part II)

Instance	ILS-SP			UHGS			MA_ILS-RVND			BKS
(#)Name	Avg.	Best	Time	Avg.	Best	Time	Avg.	Best	Time	
(51)X-n336-k84	139,461.0	139,197	21.41	139,534.9	139,210	37.96	139,760.6	139,640	25.84	139,135
(52)X-n344-k43	42,284.0	42,146	22.58	42,208.8	42,099	21.67	42,342.8	42,273	15.00	42,056
(53)X-n351-k40	26,150.3	26,021	25.21	26,014.0	25,946	33.73	26,124.5	26,057	18.11	25,928
(54)X-n359-k29	52,076.5	51,706	48.86	51,721.7	51,509	34.85	51,870.4	51,787	36.45	51,505
(55)X-n367-k17	23,003.2	22,902	13.13	22,838.4	22,814	22.02	22,943.0	22,888	24.75	22,814
(56)X-n376-k94	147,713.0	147,713	7.10	147,750.2	147,717	28.26	147,722.3	147,714	59.12	147,713
(57)X-n384-k52	66,372.5	66,116	34.47	66,270.2	66,081	40.20	66,360.4	66,320	27.66	65,943
(58)X-n393-k38	38,457.4	38,298	20.82	38,374.9	38,269	28.65	38,439.2	38,389	22.28	38,260
(59)X-n401-k29	66,715.1	66,453	60.36	66,365.4	66,243	49.52	66,515.1	66,428	61.91	66,187
(60)X-n411-k19	19,954.9	19,792	23.76	19,743.8	19,718	34.71	19,950.2	19,801	37.72	19,718
(61)X-n420-k130	107,838.0	107,798	22.19	107,924.1	107,798	53.19	108,109.4	108,021	24.79	107,798
(62)X-n429-k61	65,746.6	65,563	38.22	65,648.5	65,501	41.45	65,773.4	65,727	24.60	65,483
(63)X-n439-k37	36,441.6	36,395	39.63	36,451.1	36,395	34.55	36,457.6	36,428	26.22	36,391
(64)X-n449-k29	56,204.9	55,761	59.94	55,553.1	55,378	64.92	55,760.7	55,645	37.02	55,269
(65)X-n459-k26	24,462.4	24,209	60.59	24,272.6	24,181	42.80	24,347.9	24,308	35.54	24,145
(66)X-n469-k138	222,182.0	221,909	36.32	222,617.1	222,070	86.65	223,029.3	222,722	55.34	221,909
(67)X-n480-k70	89,871.2	89,694	50.40	89,760.1	89,535	66.96	89,800.8	89,725	43.58	89,458
(68)X-n491-k59	67,226.7	66,965	52.23	66,898.0	66,633	71.94	66,941.7	66,788	38.88	66,510
(69)X-n502-k39	69,346.8	69,284	80.75	69,328.8	69,253	63.61	69,288.1	69,244	92.83	69,230
(70)X-n513-k21	24,434.0	24,332	35.04	24,296.6	24,201	33.09	24,372.0	24,273	29.24	24,201
(71)X-n524-k153	155,005.0	154,709	27.27	154,979.5	154,774	80.70	155,857.3	155,369	90.10	154,593
(72)X-n536-k96	95,700.7	95,524	62.07	95,330.6	95,122	107.53	95,644.8	95,512	59.60	94,988
(73)X-n548-k50	86,874.1	86,710	63.95	86,998.5	86,822	84.24	86,863.0	86,799	112.48	86,701
(74)X-n561-k42	43,131.3	42,952	68.86	42,866.4	42,756	60.60	43,010.2	42,933	34.48	42,722
(75)X-n573-k30	51,173.0	51,092	112.03	50,915.1	50,780	188.15	50,957.4	50,882	115.89	50,718
(76)X-n586-k159	190,919.0	190,612	78.54	190,838.0	190,543	175.29	191,215.6	191,043	79.36	190,423
(77)X-n599-k92	109,384.0	109,056	72.96	109,064.2	108,813	125.91	109,159.8	108,985	65.89	108,489
(78)X-n613-k62	60,444.2	60,229	74.80	59,960.0	59,778	117.31	60,133.5	60,004	40.95	59,556
(79)X-n627-k43	62,905.6	62,783	162.67	62,524.1	62,366	239.68	62,617.3	62,517	95.00	62,210
(80)X-n641-k35	64,606.1	64,462	140.42	64,192.0	63,839	158.81	64,411.9	64,315	93.56	63,737
(81)X-n655-k131	106,782.0	106,780	47.24	106,899.1	106,829	150.48	106,827.6	106,810	212.86	106,780
(82)X-n670-k130	147,676.0	147,045	61.24	147,222.7	146,705	264.10	149,348.4	148,290	151.21	146,477
(83)X-n685-k75	68,988.2	68,646	73.85	68,654.1	68,425	156.71	68,811.6	68,684	61.45	68,261
(84)X-n701-k44	83,042.2	82,888	210.08	82,487.4	82,293	253.17	82,656.2	82,545	120.04	81,934
(85)X-n716-k35	44,171.6	44,021	225.79	43,641.4	43,525	264.28	43,917.8	43,726	110.78	43,414
(86)X-n733-k159	137,045.0	136,832	111.56	136,587.6	136,366	244.53	136,774.9	136,635	74.00	136,250
(87)X-n749-k98	78,275.9	77,952	127.24	77,864.9	77,715	313.88	77,964.1	77,855	92.03	77,365
(88)X-n766-k71	115,738.0	115,443	242.11	115,147.9	114,683	382.99	115,780.6	115,555	162.12	114,525
(89)X-n783-k48	73,722.9	73,447	235.48	73,009.6	72,781	269.70	73,193.0	73,035	114.52	72,445
(90)X-n801-k40	74,005.7	73,830	432.64	73,731.0	73,587	289.24	73,723.0	73,621	201.06	73,331
(91)X-n819-k171	159,425.0	159,164	148.91	158,899.3	158,611	374.28	159,219.9	158,987	122.97	158,265
(92)X-n837-k142	195,027.0	194,804	173.17	194,476.5	194,266	463.36	194,629.0	194,401	181.02	193,810
(93)X-n856-k95	89,277.6	89,060	153.65	89,238.7	89,118	288.43	89,183.1	89,038	153.92	89,002
(94)X-n876-k59	100,417.0	100,177	409.31	99,884.1	99,715	495.38	100,095.9	99,958	207.18	99,331
(95)X-n895-k37	54,958.5	54,713	410.17	54,439.8	54,172	321.89	54,664.9	54,461	141.62	53,946
(96)X-n916-k207	330,948.0	330,639	226.08	330,198.3	329,836	560.81	330,386.0	330,132	288.43	329,247
(97)X-n936-k151	134,530.0	133,592	202.50	133,512.9	133,140	531.50	135,572.9	134,890	199.36	132,923
(98)X-n957-k87	85,936.6	85,697	311.20	85,822.6	85,672	432.90	85,728.6	85,607	243.52	85,478
(99)X-n979-k58	120,253.0	119,994	687.22	119,502.1	119,194	553.96	120,354.6	119,821	321.29	119,008
(100)X-n1001-k43	73,985.4	73,776	792.75	72,956.0	72,742	549.03	73,476.7	73,318	207.55	72,403
Avg. Gap	0.61%	0.33%		0.27%	0.09%		0.54%	0.37%		
Avg. Time	71.71 min.			98.79 min.			54.77 min.			
CPU	Xeon 3.07 GHz			Xeon 3.07 GHz			Core i7 3.40 GHz			
$\sim$ GFlops/Core	3.63			3.63			4.01			
Scale Factor	0.91			0.91			1.00			
Scaled Time	65,26 min.			89,90 min.			54,77 min.			

BKS denotes the best-known solutions that are taken from `http://vrp.atd-lab.inf.puc-rio.br/`. All the time values are given in minutes. Estimated GFlop values for CPUs are taken from `https://asteroidsathome.net/boinc/cpu_list.php/`

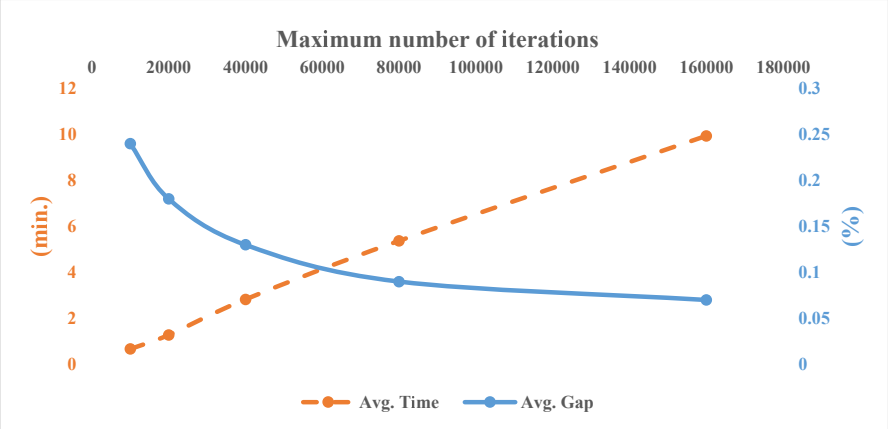

Fig. 4 Impact of the maximum number of iterations on the performance of MA_ILS-RVND algorithm

periments for Christofides et al. (1979) dataset have proved that using multi-start and adaptive acceptance strategies together contribute the performance of pure ILS-RVND algorithm. It has also shown that CPU time of the algorithm increases linearly with the number of iterations. Final experiments for Uchoa et al. (2017) showed that our algorithm is highly effective for solving CVRP and comparable with state-of-the-art algorithms.

Further research might investigate how parameter control techniques can be applied to the two most critical parameters of MA_ILS-RVND, which are $pFactor$ and $rstFactor$. Therefore, the algorithm could adapt itself to each problem instance, and improve its overall performance. Another possible area of future research would be extending and applying MA_ILS-RVND to other common VRP variants.

Acknowledgements Author Osman Gokalp acknowledges the support of Scientific and Technological Research Council of Turkey (TUBITAK) 2211 National Graduate Scholarship Program.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

Agrawal V, Lightner C, Lightner-Laws C, Wagner N (2017) A bi-criteria evolutionary algorithm for a constrained multi-depot vehicle routing problem. Soft Computing 21(17):5159-5178

Alabas-Uslu C, Dengiz B (2011) A self-adaptive local search algorithm for the classical vehicle routing problem. Expert Systems with Applications 38(7):8990-8998

Avci M, Topaloglu S (2015) An adaptive local search algorithm for vehicle routing problem with simultaneous and mixed pickups and deliveries. Computers & Industrial Engineering 83:15-29

Christofides N, Mingozzi A, Toth P (1979) The vehicle routing problem. In: Christofides N, Mingozzi A, Toth P, Sandi C. (eds) Combinatorial Optimization. Wiley, Chichester, pp 315-338

Clarke G, Wright JW (1964) Scheduling of vehicles from a central depot to a number of delivery points. Operations research 12(4):568-581

Dantzig GB,Ramser JH (1959) The truck dispatching problem. Management science 6(1):80-91

Dethloff J (2001) Vehicle routing and reverse logistics: the vehicle routing problem with simultaneous delivery and pick-up. OR-Spektrum 23(1):79-96

Dueck G, Scheuer T (1990) Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. Journal of computational physics 90(1):161-175

Fu Z, Eglese R, Li LYO (2005) A new tabu search heuristic for the open vehicle routing problem. Journal of the Operational Research Society 56(3):267-274

Gee SB, Arokiasami WA, Jiang J, Tan KC (2016) Decomposition-based multi-objective evolutionary algorithm for vehicle routing problem with stochastic demands. Soft Computing 20(9):3443-3453

Hansen P, Mladenović N (2001) Variable neighborhood search: Principles and applications. European journal of operational research 130(3):449-467

Irnich S, Funke B, Grünert T (2006) Sequential search and its application to vehicle-routing problems. Com-

puters & Operations Research 33(8):2405-2429

Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. science 220(4598):671-680

Lenstra JK, Kan AR (1981) Complexity of vehicle routing and scheduling problems. Networks 11(2):221-227

López-Ibáñez M, Dubois-Lacoste J, Cáceres LP, Birattari M, Stützle T (2016) The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives 3:43-58

Michallet J, Prins C, Amodeo L, Yalaoui F, Vitry G (2014) Multi-start iterated local search for the periodic vehicle routing problem with time windows and time spread constraints on services. Computers & operations research 41:196-207

Molina J, López-Sánchez A, Hernández-Díaz AG, Martínez-Salazar I (2018) A multi-start algorithm with intelligent neighborhood selection for solving multi-objective humanitarian vehicle routing problems. Journal of Heuristics 24(2):111-133

Nalepa J, Blocho M (2016) Adaptive memetic algorithm for minimizing distance in the vehicle routing problem with time windows. Soft Computing 20(6):2309-2327

Penna PHV, Subramanian A, Ochi LS (2013) An iterated local search heuristic for the heterogeneous fleet vehicle routing problem. Journal of Heuristics 19(2):201-232

Psychas ID, Marinaki M, Marinakis Y, Migdalas A (2017) Parallel multi-start nondominated sorting particle swarm optimization algorithms for the minimization of the route-based fuel consumption of multiobjective vehicle routing problems. In Optimization Methods and Applications, Springer, pp 425-456

Rivera JC, Afsar HM, Prins C (2015) A multistart iterated local search for the multitrip cumulative capacitated vehicle routing problem. Computational Optimization and Applications 61(1):159-187

Sassi O, Cherif-Khettaf WR, Oulamara A (2015) Multi-start iterated local search for the mixed fleet vehicle routing problem with heterogenous electric vehicles. In: European Conference on Evolutionary Computation in Combinatorial Optimization, Springer, pp 138-149

Stützle T (1998) Local search algorithms for combinatorial problems. Dissertation, Darmstadt University of Technology

Subramanian A, Uchoa E, Ochi LS (2013) A hybrid algorithm for a class of vehicle routing problems. Computers & Operations Research 40(10):2519-2531

Taillard E, Badeau P, Gendreau M, Guertin F, Potvin JY (1997) A tabu search heuristic for the vehicle routing problem with soft time windows. Transportation science 31(2):170-186

Tarantilis C, Kiranoudis C, Vassiliadis V (2002) A backtracking adaptive threshold accepting algorithm for the vehicle routing problem. Systems Analysis Modelling Simulation 42(5):631-664

Uchoa E, Pecin D, Pessoa A, Poggi M, Vidal T, Subramanian A (2017) New benchmark instances for the capacitated vehicle routing problem. European Journal of Operational Research 257(3):845-858

Vidal T, Crainic TG, Gendreau M, Lahrichi N, Rei W (2012) A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. Operations Research 60(3):611-624