Computer Science • 24(3) 2023 https://doi.org/10.7494/csci.2023.24.3.4367

Ha-Bang Ban
Hong-Phuong Nguyen
Dang-Hai Pham

HYBRID VARIABLE NEIGHBORHOOD SEARCH FOR SOLVING SCHOOL BUS-DRIVER PROBLEM WITH RESOURCE CONSTRAINTS

Abstract *The school bus-driver problem with resource constraints (SBDP-RC) is an optimization problem with many practical applications. In the problem, several vehicles are prepared to pick up a number of pupils in which the total resources of all vehicles are lower than a predefined value. The aim is to find a schedule that minimizes the sum of the pupils' waiting times. The problem is NP-hard in the general case. In this paper, we propose a two-phase metaheuristic to solve the problem. The first phase creates an initial solution by using an insertion heuristic. After this, the post phase improves the solution by a general variable neighborhood search (GVNS) with a random neighborhood search combined with the shaking technique. The proposed metaheuristic algorithm is tested on a benchmark to show its efficiency. The results show that the algorithm received good feasible solutions fast. In many cases, better solutions can be found compared to previous metaheuristic algorithms.*

Keywords SBDP-RC, metaheuristic, VNS

Citation Computer Science 24(3) 2023: 297–325

1. Introduction

1.1. Motivation

Some variants of SBDP-RC are the cumulative multiple traveling salesmen problem (C-TSP), multiple deliverymen problem (mDMP), and multiple traveling repairman problem (mTRP). These consider a set of vehicles to find a route that minimizes the total waiting times of all of the clients [4, 6, 8, 12, 14, 15, 18]. In mDMP and mTRP, there are no constraints. SBDP-RC has many practical applications (which can be found in [19]). There is only one vehicle whose resources are infinitive in the original school bus-driver problem (SBDP) [19]. This means it can run for as long as it wants; however, it is impossible in real situations when vehicles have strict regulations on resources such as oil, gas, etc. This paper considers two new assumptions: 1) multiple vehicles; and 2) the maximum total resources RM_{max} of all vehicles. We describe the problem: a set of vehicles at a starting depot and clients at different locations. The aim is to obtain a tour such that each client is picked up in which the total resources of all vehicles are limited and the total waiting times of all clients are minimized.

1.2. Problem statement

A complete graph K_n includes a set of n vertex $V = \{v_1, v_2, ..., v_n\}$ and a distance matrix $C = \{c(v_i, v_j) \mid i, j = 1, 2, ..., n\}$ ($c(v_i, v_j)$ that is the cost to travel from vertex v_i to v_j). A resource matrix $RM = \{r(v_i, v_j)\}$ shows the required resource consumption to travel from vertex v_i to v_j. Let $R = (1, 2, ..., k)$ be a set of k vehicles. All vehicles start at a depot $s = v_1$. Let RM_{max} be the maximum total resources of all vehicles. A route $T = (R_1, ..., R_l, ..., R_k)$ consists of a set of routes. Each route $R_l = (v_1, ..., v_h, ..., v_m, v_{m+1} = v_1)$ is created by vehicle $l - th$. The waiting time of v_h ($1 < h \leq m$) on R_l is the cost of the path from v_1 to v_h:

$$l(P(v_1, v_h)) = \sum_{i=1}^{h-1} c(v_i, v_{i+1}) \tag{1}$$

Let $W(R_l)$ be the total of the waiting times of R_l, and the resource consumption of route R_l (LR) is the total of the resource consumption on its edges.

$$W(R_l) = \sum_{h=2}^{m+1} l(P(v_1, v_h)) \tag{2}$$

$$LR(R_l) = \sum_{i=1}^{m} r(v_i, v_{i+1}) \tag{3}$$

The aim is as follows:

$$W(T) = \sum_{l=1}^{k} W(R_l) \to \min \tag{4}$$

The resource consumption of each vehicle must satisfy the following:

$$\sum_{l=1}^{k} LR(R_l) \leq RM_{max} \tag{5}$$

SBDP-RC requires a solution that begins at v_1 and visits each vertex exactly once such that the waiting times of the route are minimized. In this problem, we are interested in a Hamiltonian cycle; this means that the deliverymen return to the vertex from which they began their routes. Consider the example of the small graph that is shown in Figure 1.

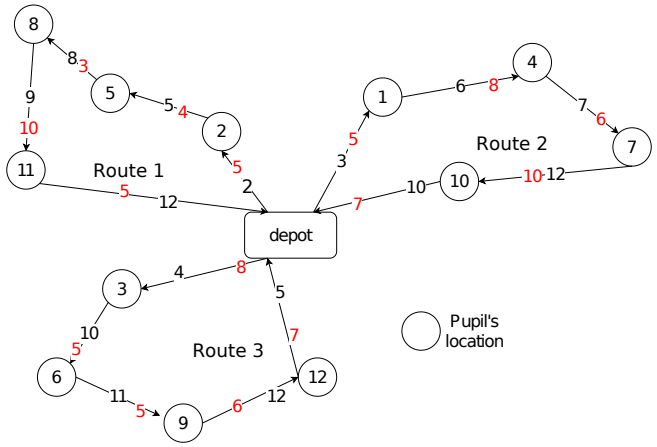

Figure 1. Example of how SBDP-RC is represented in drawing

Assume that we have complete graph $K_{12} = 0 \cup \{1, 2, 3, ..., 12\}$. All vehicles start at the main depot (vertex 0), and each pupil's location corresponds to a vertex in the graph. The cost values to travel between two vertices are highlighted in black, while the resource consumption values are highlighted in red. We have route $T = (R_1 = (v_0, v_2, v_5, v_8, v_{11}, v_0), R_2 = (v_0, v_1, v_4, v_7, v_{10}, v_0), R_3 = (v_0, v_3, v_6, v_9, v_{12}, v_0))$. Assume that the value of RM_{max} is 100; the waiting times for each route are calculated as follows:

$$\begin{aligned} W(R_1) \quad = \quad & c(v_0, v_2) + c(v_0, v_2) + c(v_2, v_5) + c(v_0, v_2) \\ & + c(v_2, v_5) + c(v_5, v_8) + c(v_0, v_2) \\ & + c(v_2, v_5) + c(v_5, v_8) + c(v_8, v_{11}) \\ & + c(v_5, v_8) + c(v_0, v_2) + c(v_2, v_5) \\ & + c(v_5, v_8) + c(v_8, v_{11}) + c(v_{11}, v_0) \\ & = 2 + (2 + 5) + (2 + 5 + 8) \\ & + (2 + 5 + 8 + 9) + (2 + 5 + 8 + 9 + 12) = 84 \end{aligned}$$

$$\begin{aligned} W(R_2) &= c(v_0,v_1)+c(v_0,v_1)+c(v_1,v_4)+c(v_0,v_1) \\ &\quad +c(v_1,v_4)+c(v_4,v_7)+c(v_0,v_1) \\ &\quad +c(v_1,v_4)+c(v_4,v_7)+c(v_7,v_{10}) \\ &\quad +c(v_0,v_1)+c(v_1,v_4)+c(v_4,v_7) \\ &\quad +c(v_7,v_{10})+c(v_{10},v_0) \\ &\quad =3+(3+6)+(3+6+7) \\ &\quad +(3+6+7+12)+(3+6+7+12+10)=94 \end{aligned}$$

$$\begin{aligned} W(R_3) &= c(v_0,v_3)+c(v_0,v_3)+c(v_3,v_6) \\ &\quad +c(v_0,v_3)+c(v_3,v_6)+c(v_6,v_9) \\ &\quad +c(v_0,v_3)+c(v_3,v_6)+c(v_6,v_9) \\ &\quad +c(v_9,v_{12})+c(v_0,v_3)+c(v_3,v_6) \\ &\quad +c(v_6,v_9)+c(v_9,v_{12})+c(v_{12},v_0) \\ &\quad =4+(4+10)+(4+10+11) \\ &\quad +(4+10+11+12) \\ &\quad +(4+10+11+12+5)=122 \end{aligned}$$

The waiting times for the route are as follows:

$$W(T)=84+94+122=300$$

The resource consumption of each route is as follows:

$$\begin{aligned} LR(R_1) &= r(v_0,v_2)+r(v_2,v_5)+r(v_5,v_8) \\ &\quad +r(v_8,v_{11}+r(v_{11},v_0) \\ &\quad =5+4+3+10+5=27 \end{aligned}$$

$$\begin{aligned} LR(R_2) &= r(v_0,v_1)+r(v_1,v_4)+r(v_4,v_7) \\ &\quad +r(v_7,v_{10})+r(v_{10},v_0) \\ &\quad 5+8+6+10+7=36 \end{aligned}$$

$$\begin{aligned} LR(R_3) &= r(v_0,v_3)+r(v_3,v_6) \\ &\quad +r(v_6,v_9)+r(v_9v_{12})+r(v_{12},v_0) \\ &\quad =8+5+5+6+7=31 \end{aligned}$$

$$LR(T)=27+36+31=94$$

The solution is feasible because the total resource consumption of all routes $LR(R_i)$ $(i=1,...,3)$ is less than RM_{max}.

1.3. Literature review

As we know, SBDP-RC has not been studied much. In the literature, several variants of the problem have been proposed; we describe these as follows: 1) mDMP or mTRP is the case when the resources are infinitive. Several metaheuristics for solving the problem were proposed in [4,13,17]. The experimental results showed that several algorithms [4,13,17] gave good solutions fast for large instances of up to 500 customers; 2) mTRP with distance constraint (mTRP-DC) is the case where the maximum duration of each vehicle is lower a predetermined value. The two metaheuristic algorithms in [2,11] can be applied well to the problem in a reasonable amount of time; 3) Capacitated mTRP [9,19] is the case where the vehicle's capacity does not exceed the permitted limit. The metaheuristic in [2] also receives good feasible solutions fast; 4) mTRP with profits (mTRPP) aims to find a solution to maximize the total revenue. In this case, some vertices may not be visited. The metaheuristic algorithm in [10,18] produced good instances with up to 200 vertices; 5) The deliveryman problem (DMP) with (without) time windows is a special case of mTRP where there is a only vehicle to run. Numerous metaheuristic algorithms [3,5,6,12] for the problem have also been developed. The experimental results showed their expressive performance for large instances; 6) Recently, a new variant of $mTRP$ post-disaster was introduced in [1,7]. In this case, an additional cost for a road-clearance operator is involved in the function cost. They tested their algorithms on the Istanbul data set.

To our knowledge, the above algorithms are the best algorithms for the problem's several variants. However, resource constraints are not involved; therefore, these algorithms are not easily adapted to SBDP-RC.

1.4. Our algorithm and contribution

The problem can be solved by exact and heuristic (or metaheuristic) algorithms. An exact algorithm obtains an optimal solution, but it consumes much time. Heuristic approaches include the classical heuristic and metaheuristic algorithms: the former finds a solution fast, but the solution's quality may not be good; on the other hand, the latter reaches a near-optimal solution in a short amount of computation time. Therefore, metaheuristic is a suitable approach for solving large-scale problems; however, its efficiency is mainly evaluated through experiments.

A good metaheuristic needs to maintain a balance between exploration and exploitation strategies. The main contributions of this work can be summarized as follows:

- From an algorithmic perspective, the proposed metaheuristic consists of two phases: 1) in the first phase (the construction phase), an initial solution is created based on the insertion heuristic scheme. This step aims to obtain a good-enough solution; 2) the post-phase (the improvement phase) improves the solution created from the previous one. Starting from a good-enough solution helps the algorithm to increase the chance of improving the solution's quality. In this phase, we use the randomized variable neighborhood search scheme (RVNS) to

investigate various neighboring solutions to find good solutions. RVNS aims to exploit a good solution space that is explored. Two additional characteristics are integrated into the proposed algorithm. First, according to a distance metric, the algorithm accepts a solution that is worse than the current solution if it is far from it; this enhances the exploration of far-away valleys. Second, the search is allowed to move to unfeasible solution spaces by using a penalty method. When a constraint is violated, the value of the parameter increases to drive the search toward feasible regions. This means that the algorithm tries to exploit the feasible regions that are explored. After this, we enlarge the search space by decreasing it if no better solutions can be found. By doing this, the algorithm has a higher chance of finding better solutions. When the algorithm still fails in finding better solutions, the shaking technique is applied to move the search toward a completely new solution space that is unexplored.

- From the computational perspective, our algorithm obtains good feasible solutions fast for instances with large sizes. Additionally, the algorithm receives better solutions as compared to the previous algorithms in many cases.

The rest of this paper is organized as follows. Section 2 introduces our algorithm; then, the experiments are described in Section 3. Sections 4 and 5 discuss and conclude the article, respectively.

2. Proposed algorithm

2.1. Variants of VNS

We describe VNS, GVNS [13], and shaking [12], respectively.

- VNS is described in [13]. It is divided into two main steps: 1) shaking, and a local search step. In the step, shaking implements the move to a random solution. The second phase consists of applying a local search to the solution and selecting the best one in a neighborhood set.
- Randomized VNS (RVNS) [13] is a variant of VNS. In RVNS, the search procedure is performed randomly to generate neighbor solutions.
- GVNS [13] is a variant of VNS. GVNS is a version of VNS in which VNS is applied as the improvement procedure. In this article, we use GVNS with a random neighborhood search.
- Skewed-GVNS [13] is an extension of basic GVNS that explores solution spaces that are far from the incumbent solution. Therefore, we can accept worse solutions if they are different from the incumbent.

2.2. Neighborhood investigation

Several neighborhoods [8, 14] in the literature are applied to exploit the search region. Let $N_k(k = 1, ..., k_m)$ be a set of neighborhood structures. Now, let $T = (R_1, R_2, ..., R_l)$ be a tour with l routes, we then introduce a novel neighborhood structure.

For inter-route: it optimizes a route.

- **Forward** (N_1) pushes a vertex forward one position. This neighborhood of R is defined as a set $N_1(R) = \{R_i = (v_1, v_2, ..., v_{i-1}, v_{i+1}, v_i, ..., v_m) : i = 2, 3, ..., m-1\}$. The complexity time is $O(n)$.
- **Backward** (N_2) pushes a vertex backward one position. This neighborhood of R is defined as a set $N_2(R) = \{R_i = (v_1, v_2, ..., v_i, v_{i-1}, v_{i+1}, ..., v_m) : i = 2, 3, ..., m-1\}$. The complexity time is $O(n)$.
- **Exchange-adjacent** (N_3) exchanges each pair of adjacent vertices. This neighborhood of R is defined as a set $N_3(R) = \{R_i = (v_1, v_2, ..., v_{i-2}, v_i, v_{i-1}, v_{i+1}, ..., v_m) : i = 3, 4, ..., m-1\}$. The complexity time is $O(n)$.
- **Exchange** (N_4) exchanges the positions of each pair of vertices. This neighborhood of R is defined as a set $N_4(R) = \{R_{ij} = (v_1, v_2, ..., v_{i-1}, v_j, v_{i+1}, ..., v_{j-1}, v_i, v_{j+1}, ..., v_m) : i = 2, 3, ..., m-3; j = i+3, ..., m\}$. The complexity time is $O(n^2)$.
- **2-opt** (N_5) removes each pair of edges from the tour and reconnects them. This neighborhood of T is defined as a set $N_5(T) = \{T_{ij} = (v_1, v_2, ..., v_i, v_j, v_{j-1}, ..., v_{i+2}, v_{i+1}, v_{j+1}, ..., v_m) : i = 1, ..., n-4; j = i+4, ..., m\}$. The complexity time is $O(n^2)$.
- **3-opt** (N_6) reallocates three vertices to another position. This neighborhood of R is defined as a set $N_6(R) = \{R_i = (v_1, v_2, ..., v_{i-1}, v_i, v_{j+1}, ..., v_k, v_{i+1}, ..., v_j, v_{k+1}...., v_m) : i = 2, 3, ..., m-5, j = 4, ..., m-3, k = 6, ..., m-1\}$. The complexity time is $O(n^3)$.

For intra-route: Intra-route is used to swap or exchange vertices between two different routes.

- **Exchange-2-routes** $N_7(R)$ exchanges two vertices from different routes. The swap-2-route neighborhood of R_l and R_h is defined as a set $N_8(T) = \{T_i = (R_1, ..., R_2, ..., R_l = (v_{1l}, v_{2l}, ..., v_{ih}, v_{il+1}, ..., v_{ml}), ..., R_h = (v_{1h}, v_{2h}, ..., v_{il}, v_{ih+1}, ..., v_{mh}), ..., R_k) : il = 2, 3, ..., ml-1, ih = 2, 3, ..., mh-1\}$. The complexity time is $O(n^2)$
- **Insert-2-routes** $N_8(R)$ removes a vertex in turn and inserts it at the best possible position in the other. An insert-2-route neighborhood of R_l and R_h is defined as a set $N_8(T) = \{T_i = (R_1, ..., R_2, ..., R_l = (v_{1l}, v_{2l}, ..., v_{ih-1}, v_{ih}, v_{il+1}, ..., v_{ml}), ..., R_h = (v_{1h}, v_{2h}, ..., v_{ih-1}, v_{ih+1}, ..., v_{mh}), ..., R_k) : il = 2, 3, ..., ml-1, ih = 2, 3, ..., mh-1\}$. The complexity time is $O(n^2)$.

2.3. Restricted infeasible solution space

Infeasible solutions are penalized by a value. With route T, let$VS(T), LR(R_i)$ be a penalty value and the resource consumption of route R_i. Penalty value $VS(T)$ is computed as follows: $\max\{(\sum_{i=1}^{k} LR(R_i) - RM_{max}), 0\}$. The solutions are then

calculated in accordance with $W' = W + PV \times VS(T)$, in which PV is the penalty factor. If the solution is feasible, then $LR \leq RM_{max}$ and $W' = W$.

To make the algorithm's structure more readable, a flowchart of the proposed algorithm is described in Figure 2. The proposed algorithm consists of two phases. Algorithm 1 depicts the whole process in pseudocode.

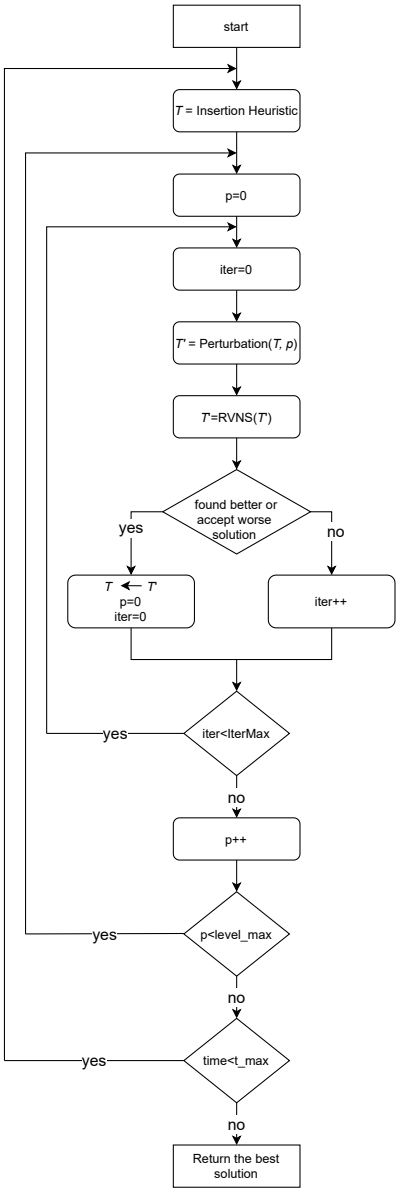

Figure 2. Flowchart of skewed GVNS algorithm

Algorithm 1 Skewed GVNS

Input: $T, IterMax$, $lvel\_max$, and t_{max} are a starting solution, the number of iterations, the strength of the perturbation procedure, and the maximum time to run, respectively.
Output: the best-found solution T^*.

repeat
  {Step 1: construction step}
  $T \leftarrow$ **Construction**(v_1, V);{T is an initial solution. It can be feasible or infeasible}
  $p = 1$;
  {Step 2: improvement step}
  while $(p < lvel\_max)$ **do**
    $iter = 0$;
    while $(iter < IterMax)$ **do**
      $T^{'} \leftarrow T$;
      {driving the search to a new promising solution space}
      $T^{'} \leftarrow$ Perturbation(T,p);
      {implement RVNS to exploit good solution space}
      $T^{'} =$ RVNS($T^{'}$);
      {accepting the worse solution}
      if $(W(T^{'}) < W(T) \times (1 + \beta \times d(T^{'}, T))$ or $(W(T^{'}) < W(T^*))$ **then**
        $T \leftarrow T^{'}$;
        p=0;
        $iter$=0;
        {update best solution}
        if $((W(T^{'}) < W(T^*))$ and (T is feasible)) **then**
          $T^* \leftarrow T^{'}$;
        end if
      else
        $iter++$;
      end if
    end while
    $p++$;
  end while
until $time < t_{max}$
return T^*;

2.4. Construction

Algorithm 2 shows the constructive procedure. Assume that we have a partial solution and $V^{'}$ is a list of unvisited vertices ($V^{'} \subseteq V$). To complete the partial solution, a vertex in $V^{'}$ needs to be inserted. We need to select a vertex and the position to insert it into the solution. We use a greedy scheme to pick a vertex so that its insertion causes the solution with the lowest cost. A solution is generated when all of the vertices of K_n are routed. The procedure then returns the feasible solution (if any). Otherwise, for added randomness in routing, it tries to generate n solutions; then, the one with the minimum fitness value will be returned.

Algorithm 2 Construction(v_1, K_n)

Input: v_1, K_n are a main depot and the graph, respectively.
Output: A starting solution T.

$S = \emptyset$; {S is the list of infeasible routes}
$FOUND$=False;
$T = \phi$; {Initially, T is empty}
for $(l = 1; l \leq k; l++)$ **do**
  $R_l \leftarrow R_l \cup v_1$; {$k$ routes start at depot}
end for
repeat
  repeat
    Select a random route $R_l (R_l \in R)$;
    Randomly pick a new vertex v and an inserted position $j < |R_l|$ so that the cost of $R_l^{'}$ after inserting is minimal; {$|R_l|$ is the number of vertices inR_l}
    Update R_l by $R_l^{'}$;
  until all vertices are visited
  for $(i = 1; i \leq k; i++)$ **do**
    $T \leftarrow T \cup R_i$;{update all routes in the tour}
  end for
  if (T is feasible) **then**
    return T;
  else
    $S \leftarrow S \cup T$;
  end if
until $|S| < n$
if $FOUND$=False **then**
  $T \longleftarrow$ solution with minimum cost W' in set S;
end if
return T;

2.5. Improvement

In the second step, it tries to improve the feasible solution that was created by the previous phase. In this step, we use RVNS in [13] to exploit the neighborhood solutions. Whenever a given neighborhood of set N fails to improve the current best solution, RVNS randomly selects another neighborhood from the same set to continue the search. The aim of using RVNS is to exploit a good solution space that has just been explored. Preliminary experiments indicate that randomly selecting another neighborhood can find better solutions than a deterministic order. If we find a better solution, it becomes the new current solution. However, the search cannot escape from very large valleys in some cases. In this paper, we adopt a skewed VNS approach that permits moves to worse solutions to explore more valleys that are far from the current solution. The aim is to support the search for getting out of huge valleys. Here, we make a move from solution T to neighboring solution $T^{''}$ if

$$W(T^{'}) < W(T) \times (1 + \beta \times d(T, T^{'}) \tag{6}$$

Let $d(T, T^{'})$ be the metric distance between T, and $T^{'}$; this shows the difference between the two solutions. The greater and greater the metric distance is, the more and more the difference is. In mathematical respect, the distance is the minimum number of transformations from T to $T^{'}$. When there exists no polynomial operator for calculating $d(T, T^{'})$, $d(T, T^{'})$ is n (the number of vertices in the graph) minus the number of vertices that have the same position in both T and $T^{'}$.

The detail of the improvement step is described in Algorithm 3.

Algorithm 3 RVNS(T)

Input: T is a route.
Output: A new solution T.

```
Initialize neighborhood list NL;
while NL ≠ 0 do
    Choose a neighborhood N in NL at random
    T' ← arg min N(T); {Neighborhood search}
    if ((L(T') < L(T)) then
        T ← T'
        Update NL;
    else
        Remove N from the NL;
    end if
end while
```

The aim of the perturbation mechanism is to maintain exploration; it drives the search to a new promising solution space. If the mechanism implements too-small shaking moves, the search gets stuck into the local optima. Conversely, large moves in the shaking drive the search to unpromising or infeasible spaces. In an approach to fulfill the omission, we use a new shaking technique that was developed from the original double-bridge technique [12] (see Figure 3).

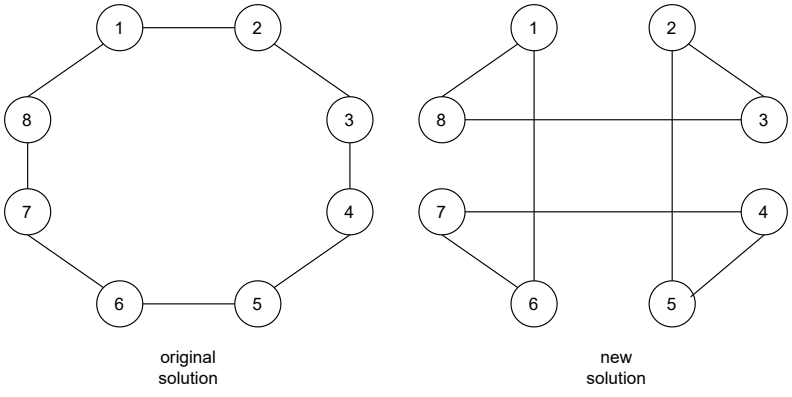

Figure 3. Double bridge

A randomly neighboring solution $T^{''}$ is generated by the double-bridge or random exchange method; then, it replaces the current solution if $(W(T^{''}) > (1-\rho) \times W(T^{*}))$ (ρ is a threshold ratio). The shaking procedure performs p times, where ρ is a parameter that is called the strength of the shake. The shaking is applied successfully in [19]. The detail is described in Algorithm 4.

Algorithm 4 Perturbation(T, p)

Input: T, T^{*}, p are the route, the best current route, and the value to control the strength of the perturbation, respectively.
Output: a route T.

```
1: i = 1;
2: while (i < p) do
3:    {Select random method to shaking}
4:    rnd=rand(2);
5:    if (rnd == 1) then
6:       T' ← Apply double-bridge in T;
7:    else
8:       T' ← Exchange randomly vertices in T;
9:    end if
10:   T'' ← arg min 3-opt-(T');
11:   if (W(T'') > (1 − ρ) × W(T*)) then
12:      T ← T''
13:      break;
14:   else
15:      i + +;
16:   end if
17: end while
18: return T;
```

The algorithm finishes after t_{max} seconds or when the best-found solution is reached.

3. Evaluations

Our algorithm is implemented on a single-threaded Pentium 4 core i7 2.50 GHz processor with 16 GB of RAM.

As we know, the parameter values quite strongly affect the quality of the solutions. The choices of parameter values were conducted in the preliminary experiments. Finding the best configuration by running all instances would be computationally too expensive, so we implemented our analysis on some selected instances. This determined configuration was tested in multiple combinations, and the one that presented the best solution was chosen. In Table 1, we define a range for each of the five parameters that yielded 1875 different parameter combinations and ran the algorithm for some selected instances of these combinations. This exhaustive search for the best parameter combinations was useful as a benchmark for evaluating the

algorithm. Looking at the parameter combinations, we found the following settings so that our algorithm could obtain the best solutions: β=5, $PV = 5$, $IterMax = 10$, $level\_max = 10$, $\rho = 0.3$, and $t\_max = 300$.

Table 1

Variable parameters

Parameters	Value ranges
PV	$2 \leq PV \leq 0$, incremented by 2
β	$2 \leq \beta \leq 10$, incremented by 2
$IterMax$	$5 \leq IterMax \leq 20$, incremented by 5
$level\_max$	$5 \leq level\_max \leq 20$, incremented by 5
t_max	$100 \leq t\_max \leq 300$ seconds incremented by 100

The experiments were implemented on benchmark instances for capacitated VRP in [15] and several random data sets. These were as follows:

- The random data set: the cost matrix elements (c_{ij}) were independent and uniformly chosen from random integers (from 0 to 500). The resource matrix elements (r_{ij}) were independent and uniformly chosen integers from 0 to 500 – c_{ij}. These cost matrices were symmetrical; moreover, these costs satisfied the triangle inequality. The maximum resource (R_{max}) was computed by using the following formula:

$$R_{max} = \lceil (1- \propto) \times \sum_{i \in V} \sum_{j \in V} r_{ij} x^c_{ij} + \propto \times \sum_{i \in V} \sum_{j \in V} r_{ij} x^r_{ij} \rceil \tag{7}$$

  The x^c_{ij} values represented the optimal solution for C-mTSP in which the cost matrix was defined by matrix c_{ij}. Similarly, the x^r_{ij} values represented the optimal solution of problem

$$\sum_{i=1}^{k} LR_i \to \min \tag{8}$$

  where the cost matrix was defined by matrix r_{ij}.
  The number of vehicles was generated randomly within a range of $[\frac{n}{5}, \frac{n}{10}]$. α was used to control the tightness of the resource constraint. Beginning with $\alpha = 1$, the resource constraint became tight; if $\alpha = 0$, the problem became an unconstrained problem. They chose $\alpha = 0.5$, 0.75, and 1 and varied the sizes of the instances (between 30 and 150 vertices) to create 300 instances. In formula R_{max}, the approximate solutions were used instead of finding the optimal solutions. The approximate solutions were computed by using the metaheuristic in [2, 16]. All of the instances were supported upon request.
- Christofides et al.: this data set consisted of instances such as CMT6, CMT7, ..., and CMT14.
- Z. Luo et al. and S. Nucamendi-Guilln et al.: 250 instances were used in the experiments; the optimal solutions of these can be obtained from [11].

3.1. Results

The performance of the proposed algorithm was compared to the initial solution from the construction phase as follows:

$$Gap_1[\%] = \frac{Best.Sol - Init.Sol}{Init.Sol} \times 100\% \tag{9}$$

The execution time of the proposed algorithm in each run was measured in seconds. Each instance was run ten times. In the Tables 2–16, *OPT*, *Init.Sol*, *Best.Sol*, *Aver.Sol*, and T corresponded to the optimal solution, the initial solution, the best solution, the average solution, and the average time by seconds over ten executions, respectively.

In Tables 2 through 16, Column 1 shows the output of the construction phase, while Columns from 2 through 5 correspond to the best solution, the average solution, *Gap*, and the running time of the proposed algorithm after ten runs, respectively. The differences in the objective function between SBDP-RC and C-mTSP on the selected instances are described in Table 17. Columns *VNS* and *GVNS* in Table 17 show the results of Ban et al.'s algorithm in [7] and the proposed algorithm, respectively. Tables 18 through 20 indicate the comparison between the proposed algorithm and several state-of-the-art metaheuristics in terms of solution quality. Each column in the two tables represents the best solution for each algorithm [2, 4, 9, 11, 16, 17].

Table 2
Our results for ins-30-x with $\alpha = 0.5$

Instances	$Init.Sol$	SKEWED-GVNS			
		$Best.Sol$	$Aver.Sol$	Gap_1	$Time$
ins-30-1	4,326.63	3,367.16	3,367.16	−22.18	0.19
ins-30-2	–	3,047.12	3,047.12	–	0.19
ins-30-3	–	3,092.38	3,092.38	–	0.2
ins-30-4	4,333.55	3,608.81	3,608.81	−16.72	0.21
ins-30-5	–	3,089.98	3,089.98	–	0.18
ins-30-6	3,878.61	3,118.82	3,118.82	−19.59	0.18
ins-30-7	–	2,890.81	2,890.81	–	0.2
ins-30-8	–	3,247.83	3,247.83	–	0.19
ins-30-9	–	3,052.64	3,052.64	–	0.19
ins-30-10	4,202.63	3,184.64	3,184.64	−24.22	0.2
ins-30-11	–	3,139.65	3,139.65	–	0.2
ins-30-12	3,789.79	3,073.56	3,073.56	−18.90	0.19
ins-30-13	–	3,075.03	3,075.03	–	0.2
ins-30-14	3,783.01	3,005.11	3,005.11	−20.56	0.19
ins-30-15	–	3,080.24	3,080.24	–	0.18
ins-30-16	–	3,134.26	3,134.26	–	0.21
ins-30-17	–	3,210.07	3,210.07	–	0.2
ins-30-18	–	3,247.09	3,247.09	–	0.18
ins-30-19	4,059.33	3,185.03	3,185.03	−21.54	0.19
ins-30-20	–	2,954.77	2,954.77	–	0.19

Table 3
Our results for ins-40-x with $\alpha = 0.5$

Instances	$Init.Sol$	SKEWED-GVNS			
		$Best.Sol$	$Aver.Sol$	Gap_1	$Time$
ins-40-1	5,061.54	4,173.1	4,173.1	−17.55	0.38
ins-40-2	–	4,338.94	4,338.94	–	0.39
ins-40-3	–	4,081.79	4,081.79	−15.17	0.41
ins-40-4	–	4,273.97	4,273.97	–	0.39
ins-40-5	–	3,987.33	3,987.33	–	0.39
ins-40-6	4,846.46	4,272.83	4,272.83	–	0.4
ins-40-7	–	4,074.3	4,074.3	–	0.38
ins-40-8	–	4,282.6	4,282.6	−10.15	0.42
ins-40-9	–	4,566.88	4,566.88	–	0.4
ins-40-10	5,403.42	4,195.86	4,195.86	–	0.41
ins-40-11	–	4,360.23	4,360.23	–	0.42
ins-40-12	4,831.03	3,894.8	3,894.8	–	0.41
ins-40-13	–	4,102.23	4,102.23	–	0.42
ins-40-14	–	4,197.87	4,197.87	–	0.39
ins-40-15	–	4,092.98	4,092.98	−11.14	0.4
ins-40-16	–	4,344.44	4,344.44	–	0.39
ins-40-17	–	4,134.12	4,134.12	–	0.39
ins-40-18	–	4,265.24	4,265.24	–	0.4
ins-40-19	5,204.03	4,108.38	4,108.38	–	0.38
ins-40-20	–	4,165.23	4,165.23	–	0.4

Table 4
Our results for ins-50-x with $\alpha = 0.5$

Instances	$Init.Sol$	SKEWED-GVNS			
		$Best.Sol$	$Aver.Sol$	Gap_1	$Time$
ins-50-1	–	5,786.72	5,786.72	–	0.65
ins-50-2	–	5,732.18	5,732.18	–	0.71
ins-50-3	–	5,500.63	5,500.63	–	0.69
ins-50-4	–	5,255.76	5,255.76	–	0.69
ins-50-5	–	5,228.71	5,228.71	–	0.69
ins-50-6	–	5,560.95	5,560.95	–	0.7
ins-50-7	–	5,634.2	5,634.2	–	0.69
ins-50-8	–	5,314.96	5,314.96	–	0.68
ins-50-9	–	5,675.45	5,675.45	–	0.66
ins-50-10	–	5,724.62	5,724.62	–	0.65
ins-50-11	–	5,635.41	5,635.41	–	0.68
ins-50-12	–	5,403.72	5,403.72	–	0.72
ins-50-13	–	5,861.04	5,861.04	–	0.71
ins-50-14	–	5,404.58	5,404.58	–	0.72
ins-50-15	–	5,920.14	5,920.14	–	0.67
ins-50-16	–	5,391.8	5,391.8	–	0.7
ins-50-17	–	5,406.67	5,406.67	–	0.71
ins-50-18	–	5,346.13	5,346.13	–	0.71
ins-50-19	–	5,530.46	5,530.46	–	0.71
ins-50-20	–	5,584.8	5,584.8	–	0.65

Table 5
Our results for ins-100-x with $\alpha = 0.5$

Instances	$Init.Sol$	SKEWED-GVNS			
		$Best.Sol$	$Aver.Sol$	Gap_1	$Time$
ins-100-1	–	9,060.56	9,086.39	–	7.55
ins-100-2	–	9,479.07	9,505.48	–	7.56
ins-100-3	–	9,318.44	9,353.17	–	7.57
ins-100-4	–	9,975.8	10,018.88	–	7.51
ins-100-5	–	8,739.14	8,795.51	–	7.59
ins-100-6	–	8,714.62	8,746.88	–	7.5
ins-100-7	–	9,475.01	9,533.76	–	7.59
ins-100-8	–	9,362.01	9,449.89	–	7.56
ins-100-9	–	9,499.02	9,554.44	–	7.57
ins-100-10	–	9,759.77	9,781.25	–	7.6
ins-100-11	–	9,544.57	9,570.97	–	7.56
ins-100-12	–	9,559.82	9,601.6	–	7.52
ins-100-13	–	9,411.15	9,484.9	–	7.53
ins-100-14	–	9,198.61	9,338.95	–	7.56
ins-100-15	–	9,469.08	9,509.66	–	7.57
ins-100-16	–	9,141.91	9,219.16	–	7.55
ins-100-17	–	9,494.98	9,543.66	–	7.5
ins-100-18	–	9,223.54	9,253.63	–	7.52
ins-100-19	–	9,734.48	9,792.45	–	7.54
ins-100-20	–	9,466.97	9,536.23	–	7.57

Table 6
Our results for ins-150-x with $\alpha = 0.5$

Instances	$Init.Sol$	SKEWED-GVNS			
		$Best.Sol$	$Aver.Sol$	Gap_1	$Time$
ins-150-1	–	12,847.44	12,952.77	–	22.26
ins-150-2	–	12,536.4	12,711.55	–	20.91
ins-150-3	–	13,177.79	13,205.59	–	21.35
ins-150-4	–	12,482.99	12,523.83	–	22.97
ins-150-5	–	13,874.35	13,925.38	–	25.25
ins-150-6	–	13,428.54	13,465.47	–	21.32
ins-150-7	–	12,436.1	12,482.01	–	20.95
ins-150-8	–	12,982.57	13,035.8	–	22.29
ins-150-9	–	12,725.95	12,795.25	–	23.05
ins-150-10	–	12,571.01	12,656.57	–	25.17
ins-150-11	–	12,668.12	12,710.78	–	22.84
ins-150-12	–	13,233.53	13,254.91	–	23.37
ins-150-13	–	13,451.53	13,518.38	–	23.06
ins-150-14	–	13,250.7	13,268.47	–	24.25
ins-150-15	–	12,616.63	12,646.25	–	22.97
ins-150-16	–	13,424.2	13,461.36	–	24.75
ins-150-17	–	12,804.89	12,852.46	–	23.7
ins-150-18	–	13,178.89	13,201.46	–	20.89
ins-150-19	–	12,796.92	12,824.38	–	21.43
ins-150-20	–	13,221.89	13,271.57	–	23.35

Table 7
Our results for ins-30-x with $\alpha = 0.75$

Instances	$Init.Sol$	SKEWED-GVNS			
		$Best.Sol$	$Aver.Sol$	Gap_1	$Time$
ins-30-1	5,422.92	4,315.47	4,315.47	–20.42	0.21
ins-30-2	–	3,650.47	3,650.47	–	0.21
ins-30-3	–	3,934.51	3,934.51	–	0.19
ins-30-4	4,562.64	4,093.33	4,093.33	–10.29	0.18
ins-30-5	–	3,639.7	3,639.7	–	0.18
ins-30-6	5,155.67	3,894.79	3,894.79	–24.46	0.19
ins-30-7	–	3,744.96	3,744.96	–	0.22
ins-30-8	–	4,002.47	4,002.47	–	0.21
ins-30-9	–	3,524.27	3,524.27	–	0.19
ins-30-10	4,655.47	4,013.16	4,013.16	–13.80	0.23
ins-30-11	–	3,866.87	3,866.87	–	0.21
ins-30-12	4,671.65	3,540.8	3,540.8	–24.21	0.22
ins-30-13	–	3,680.01	3,680.01	–	0.21
ins-30-14	4,115.54	3,473.46	3,473.46	–15.60	0.18
ins-30-15	–	3,959.72	3,959.72	–	0.19
ins-30-16	–	3,919.45	3,919.45	–	0.18
ins-30-17	–	3,893.27	3,893.27	–	0.21
ins-30-18	–	3,583.23	3,583.23	–	0.19
ins-30-19	4,096.69	3,383.12	3,383.12	–17.42	0.18
ins-30-20	–	4,011.25	4,011.25	–	0.19

Table 8
Our results for ins-40-x with $\alpha = 0.75$

Instances	$Init.Sol$	SKEWED-GVNS			
		$Best.Sol$	$Aver.Sol$	Gap_1	$Time$
ins-40-2	–	4,782.68	4,782.68	–	0.42
ins-40-3	–	5,382	5,382	–	0.41
ins-40-4	–	4,942.35	4,942.35	–	0.39
ins-40-5	–	5,009.46	5,009.46	–	0.42
ins-40-6	5,643.02	4,921.11	4,921.11	–12.79	0.38
ins-40-7	–	5,164.31	5,164.31	–	0.40
ins-40-8	–	5,464.17	5,464.17	–	0.38
ins-40-9	–	5,672.01	5,672.01	–	0.41
ins-40-10	–	5,665.7	5,665.7	–	0.42
ins-40-11	5,941.67	5,139.66	5,139.66	–13.50	0.41
ins-40-12	–	4,862.79	4,862.79	–	0.42
ins-40-13	–	5,333.96	5,333.96	–	0.4
ins-40-14	–	5,535.54	5,535.54	–	0.41
ins-40-15	–	4,781.72	4,781.72	–	0.41
ins-40-16	–	4,922.74	4,922.74	–	0.42
ins-40-17	–	6,001.77	6,001.77	–	0.43
ins-40-18	–	5,516.58	5,516.58	–	0.41
ins-40-19	6,363.44	5,464.19	5,464.19	–14.13	0.42
ins-40-20	–	5,224.37	5,224.37	–	0.39

Table 9
Our results for ins-50-x with $\alpha = 0.75$

Instances	$Init.Sol$	SKEWED-GVNS			
		$Best.Sol$	$Aver.Sol$	Gap_1	$Time$
ins-50-2	–	6,483.58	6,483.58	–	0.72
ins-50-3	–	7,317.84	7,317.84	–	0.68
ins-50-4	–	6,902.98	6,902.98	–	0.65
ins-50-5	–	6,708.91	6,708.91	–	0.68
ins-50-6	–	6,485.7	6,485.7	–	0.69
ins-50-7	–	7,293.44	7,293.44	–	0.67
ins-50-8	–	7,295.72	7,295.72	–	0.71
ins-50-9	–	6,309.42	6,309.42	–	0.66
ins-50-10	–	7,456.7	7,456.7	–	0.69
ins-50-11	–	6,419.1	6,419.1	–	0.69
ins-50-12	–	6,303.54	6,303.54	–	0.68
ins-50-13	–	6,452.13	6,452.13	–	0.66
ins-50-14	–	6,364.6	6,364.6	–	0.69
ins-50-15	–	6,591.04	6,591.04	–	0.66
ins-50-16	–	7,434.25	7,434.25	–	0.69
ins-50-17	–	6,795.34	6,795.34	–	0.71
ins-50-18	–	6,809.74	6,809.74	–	0.68
ins-50-19	–	6,730.15	6,730.15	–	0.69
ins-50-20	–	6,405.47	6,405.47	–	0.72

Table 10
Our results for ins-100-x with $\alpha = 0.75$

Instances	$Init.Sol$	SKEWED-GVNS			
		$Best.Sol$	$Aver.Sol$	Gap_1	$Time$
ins-100-2	–	11,918.03	12,255.49	–	7.58
ins-100-3	–	12,165.69	12,269.22	–	7.56
ins-100-4	–	12,933.18	13,254.95	–	7.59
ins-100-5	–	12,044.79	12,149.64	–	7.53
ins-100-6	–	12,320.63	12,395.63	–	7.58
ins-100-7	–	13,457.59	13,519.61	–	7.56
ins-100-8	–	12,257.46	12,429.89	–	7.51
ins-100-9	–	12,624.86	12,709.53	–	7.53
ins-100-10	–	12,245.88	12,459.77	–	7.58
ins-100-11	–	13,660.47	13,816.61	–	7.52
ins-100-12	–	12,190.61	12,274.93	–	7.52
ins-100-13	–	11,916.83	12,037.89	–	7.50
ins-100-14	–	11,919.56	12,117.9	–	7.54
ins-100-15	–	13,297.69	13,615.24	–	7.53
ins-100-16	–	12,624.08	12,756.94	–	7.5
ins-100-17	–	11,133.37	11,234.6	–	7.53
ins-100-18	–	13,204.53	13,411.35	–	7.51
ins-100-19	–	12,913.24	13,060.93	–	7.55
ins-100-20	–	12,327.74	12,393.17	–	7.56

Table 11
Our results for ins-150-x with $\alpha = 0.75$

Instances	$Init.Sol$	SKEWED-GVNS			
		$Best.Sol$	$Aver.Sol$	Gap_1	$Time$
ins-150-2	–	17,086.84	17,153.01	–	20.91
ins-150-3	–	18,298.93	18,464.18	–	21.35
ins-150-4	–	16,547.86	16,595.89	–	22.97
ins-150-5	–	17,248.65	17,296.95	–	25.25
ins-150-6	–	17,712.47	17,906.1	–	21.32
ins-150-7	–	17,526.23	17,623.58	–	20.95
ins-150-8	–	17,397.08	17,423.55	–	22.29
ins-150-9	–	16,464.06	16,530.44	–	23.05
ins-150-10	–	16,225.31	16,274.28	–	25.17
ins-150-11	–	17,450.13	17,552.88	–	22.84
ins-150-12	–	17,132.49	17,185.51	–	23.37
ins-150-13	–	16,654.95	16,704.26	–	23.06
ins-150-14	–	17,565.19	17,657.84	–	24.25
ins-150-15	–	17,697.85	17,831.64	–	22.97
ins-150-16	–	18,417.19	18,460.34	–	24.75
ins-150-17	–	17,755.1	17,883.79	–	23.71
ins-150-18	–	16,932.19	17,060.23	–	20.89
ins-150-19	–	16,571.58	16,682.76	–	21.43
ins-150-20	–	18,509.04	18,669.14	–	23.35

Table 12
Our results for ins-30-x with $\alpha = 1$

Instances	$Init.Sol$	SKEWED-GVNS			
		$Best.Sol$	$Aver.Sol$	Gap_1	$Time$
ins-30-1	–	7,568.362	7,568.362	–	0.19
ins-30-2	–	7,639.978	7,639.978	–	0.18
ins-30-3	–	7,253.102	7,253.102	–	0.2
ins-30-4	–	7,180.649	7,180.649	–	0.19
ins-30-5	–	7,231.949	7,231.949	–	0.19
ins-30-6	–	6,656.719	6,656.719	–	0.2
ins-30-7	–	5,784.973	5,784.973	–	0.19
ins-30-8	–	7,396.452	7,396.452	–	0.18
ins-30-9	–	7,081.527	7,081.527	–	0.19
ins-30-10	–	7,164.16	7,164.16	–	0.18
ins-30-11	–	8,250.379	8,250.379	–	0.21
ins-30-12	–	6,746.225	6,746.225	–	0.21
ins-30-13	–	7,105.159	7,105.159	–	0.18
ins-30-14	–	6,499.687	6,499.687	–	0.2
ins-30-15	–	8,739.84	8,739.84	–	0.2
ins-30-16	–	7,812.669	7,812.669	–	0.2
ins-30-17	–	9,027.127	9,027.127	–	0.21
ins-30-18	–	6,518.759	6,518.759	–	0.21
ins-30-19	–	8,521.913	8,521.913	–	0.21
ins-30-20	–	8,406.84	8,406.84	–	0.21

Table 13
Our results for ins-40-x with $\alpha = 1$

Instances	$Init.Sol$	SKEWED-GVNS			
		$Best.Sol$	$Aver.Sol$	Gap_1	$Time$
ins-40-1	–	11,730.64	11,730.64	–	0.41
ins-40-2	–	12,041.5	12,041.5	–	0.39
ins-40-3	–	11,571.29	11,571.29	–	0.42
ins-40-4	–	11,922.62	11,922.62	–	0.42
ins-40-5	–	11,296.76	11,296.76	–	0.38
ins-40-6	–	11,635.67	11,635.67	–	0.39
ins-40-7	–	11,326.47	11,326.47	–	0.38
ins-40-8	–	11,625.75	11,625.75	–	0.4
ins-40-9	–	17,797.43	17,797.43	–	0.38
ins-40-10	–	12,859.52	12,859.52	–	0.41
ins-40-11	–	14,354.51	14,354.51	–	0.4
ins-40-12	–	10,198.71	10,198.71	–	0.38
ins-40-13	–	13,391.86	13,391.86	–	0.39
ins-40-14	–	12,604.11	12,604.11	–	0.41
ins-40-15	–	13,146.88	13,146.88	–	0.39
ins-40-16	–	10,972.01	10,972.01	–	0.41
ins-40-17	–	11,334.91	11,334.91	–	0.42
ins-40-18	–	13,012.16	13,012.16	–	0.41
ins-40-19	–	11,238.14	11,238.14	–	0.39
ins-40-20	–	11,814.5	11,814.5	–	0.39

Table 14
Our results for ins-50-x with $\alpha = 1$

Instances	$Init.Sol$	SKEWED-GVNS			
		$Best.Sol$	$Aver.Sol$	Gap_1	$Time$
ins-50-1	–	14,106.51	14,106.51	–	0.66
ins-50-2	–	15,379.81	15,379.81	–	0.69
ins-50-3	–	14,673.36	14,673.36	–	0.66
ins-50-4	–	15,568.05	15,568.05	–	0.69
ins-50-5	–	18,326.42	18,326.42	–	0.7
ins-50-6	–	15,658.96	15,658.96	–	0.69
ins-50-7	–	16,049	16,049	–	0.67
ins-50-8	–	14,314.39	14,314.39	–	0.7
ins-50-9	–	17,116.64	17,116.64	–	0.68
ins-50-10	–	15,602.31	15,602.31	–	0.66
ins-50-11	–	16,023.11	16,023.11	–	0.7
ins-50-12	–	16,622.5	16,622.5	–	0.71
ins-50-13	–	13,123.83	13,123.83	–	0.68
ins-50-14	–	15,487.54	15,487.54	–	0.66
ins-50-15	–	16,834.38	16,834.38	–	0.71
ins-50-16	–	18,836.45	18,836.45	–	0.66
ins-50-17	–	15,876.96	15,876.96	–	0.65
ins-50-18	–	17,389.61	17,389.61	–	0.69
ins-50-19	–	15,744.37	15,744.37	–	0.68
ins-50-20	–	17,926.06	17,926.06	–	0.66

Table 15
Our results for ins-100-x with $\alpha = 1$

Instances	$Init.Sol$	SKEWED-GVNS			
		$Best.Sol$	$Aver.Sol$	Gap_1	$Time$
ins-100-1	–	27,057.58	27,057.58	–	7.6
ins-100-2	–	31,577.52	31,577.52	–	7.52
ins-100-3	–	31,368.54	31,865.88	–	7.53
ins-100-4	–	35,170.95	36,844.13	–	7.5
ins-100-5	–	29,465.04	29,465.04	–	7.5
ins-100-6	–	33,834.39	33,834.39	–	7.51
ins-100-7	–	31,232.81	31,564.27	–	7.53
ins-100-8	–	34,686.89	34,686.89	–	7.57
ins-100-9	–	32,528.43	32,528.43	–	7.59
ins-100-10	–	31,729.91	31,729.91	–	7.57
ins-100-11	–	28,907.83	28,907.83	–	7.51
ins-100-12	–	29,973.64	29,973.64	–	7.51
ins-100-13	–	30,783.42	31,725.25	–	7.52
ins-100-14	–	36,420.11	36,420.11	–	7.58
ins-100-15	–	30,665.7	32,796.53	–	7.6
ins-100-16	–	34,633.73	34,633.73	–	7.58
ins-100-17	–	35,676.21	35,676.21	–	7.53
ins-100-18	–	36,236.66	36,236.66	–	7.6
ins-100-19	–	27,781.99	27,781.99	–	7.56
ins-100-20	–	34,543.9	34,543.9	–	7.52

Table 16
Our results for ins-150-x with $\alpha = 1$

Instances	$Init.Sol$	SKEWED-GVNS			
		$Best.Sol$	$Aver.Sol$	Gap_1	$Time$
ins-150-1	–	51,935.16	51,935.16	–	25.21
ins-150-2	–	51,619.08	51,619.08	–	23.17
ins-150-3	–	51,128.4	51,128.4	–	20.79
ins-150-4	–	45,907.51	45,907.51	–	22.34
ins-150-5	–	49,550.98	49,550.98	–	22.43
ins-150-6	–	50,172.08	51,424.18	–	21.37
ins-150-7	–	54,480.96	54,480.96	–	20.81
ins-150-8	–	48,266.6	49,162.37	–	22.85
ins-150-9	–	52,480.74	52,480.74	–	21.51
ins-150-10	–	54,577.25	54,824.7	–	22.11
ins-150-11	–	59,166.86	59,166.86	–	21.64
ins-150-12	–	49,901.4	49,901.4	–	23.19
ins-150-13	–	52,308.1	52,308.1	–	22.79
ins-150-14	–	60,906.43	60,906.43	–	21.01
ins-150-15	–	52,200.39	52,200.39	–	24.96
ins-150-16	–	51,574.42	51,574.42	–	22.75
ins-150-17	–	49,579.06	49,579.06	–	22.41
ins-150-18	–	47,973.87	47,973.87	–	20.61
ins-150-19	–	50,491.7	51,272.83	–	24.95
ins-150-20	–	52,295.16	52,539.81	–	21.69

Table 17
Difference of objective function between two problems on selected instances with $\alpha = 1$

Instances	SBDP	SBDP-RC	diff%
ins-30-1	3,065.37±	7,568.36	59.50
ins-30-2	3,186.74±	7,639.98	58.29
ins-30-3	3,103.10±	7,253.10	57.22
ins-30-4	3,195.08±	7,180.65	55.50
ins-40-1	4,229.21±	11,730.60	63.95
ins-40-2	4,229.21±	12,041.50	64.88
ins-40-3	4,031.50±	11,571.30	65.16
ins-40-4	4,223.01±	11,922.60	64.58
ins-50-1	5,780.46±	14,106.50	59.02
ins-50-2	5,287.84±	15,379.80	65.62
ins-50-3	5,365.22±	14,673.40	63.44
ins-50-4	5,388.70±	15,568.10	65.39
ins-100-1	9,056.89±	27,057.60	66.53
ins-100-2	9,285.91±	31,577.50	70.59
ins-100-3	8,684.86±	31,368.50	72.31
ins-100-4	8,801.22±	35,171.00	74.98
ins-150-1	12,932.36±	51,935.20	75.10
ins-150-2	11,876.05±	51,619.10	76.99
ins-150-3	12,793.13±	51,128.40	74.98
ins-150-4	12,866.01±	45,907.50	71.97

± The values were the best solutions for SBDP, but they were infeasible solutions for SBDP-RC

From Tables 2 through 16, the construction step quickly gave good feasible solutions when the constraint was not tight ($\alpha = 0.5$). However, since the constraint became tight (the values of α were 0.75 and 1), the construction phase could not find feasible solutions in some cases. Even for the large instances (from 50 to 150 vertices), the construction failed to reach feasible solutions with all values of α. This implies that the construction was not good for instances with tight constraints as well as large sizes. On the other hand, the improvement phase reached feasible solutions in all cases. Moreover, in the cases where the construction phase obtained feasible solutions, the average difference of the post phase compared with the construction phase was quite obvious. The average gap value was large and significant. The proposed algorithm was the first metaheuristic algorithm for the problem; therefore, it is impossible to compare it directly to other algorithms. Nevertheless, our algorithm produces feasible solutions for instances with 150 vertices – even in the case of tight constraints. It is a significant improvement when reaching feasible solutions for large instances is NP-hard.

The difference in the objective function between SBDP-RC and C-mTSP on the selected instances is described in Table 17. The average difference was very large

when the constraint was tight. This indicates that the resource constraint also affected the solution quality strongly. Moreover, the best solutions for C-mTSP were infeasible solutions for SBDP-RC. Obviously, the good methods for C-mTSP or mTRP could not be easily applied to solve SBDP-RC; therefore, developing the algorithm for SBDP-RC is necessary. To show the efficiency of our algorithm for some variants of SBDP-RC, we implemented our algorithm on mTRP and mTRP-DC instances (Tables from 18 through 20). The experimental results in Table 14 show that our algorithm's performance was good for mTRP-DC when the optimal solutions for the instances with 80 vertices could be found in a short computation time. To the best of our knowledge, the algorithms in [2, 11] failed to reach optimal solutions in many cases. On average, our Gap_1 (1.29) was better than their Gap_1 (1.43). When compared to Ban et al.'s (VNS) [4], Ezzine et al.'s (EA) [17], and N. Guilln's (NGA) [16] algorithms for mTRP instances, our solutions outperformed VNS [4] and EA [17] in all cases while being comparable to N. Guilln's solutions (Tab. 19). Moreover, our algorithm received optimal solutions for the problem with up to 76 vertices. In the capacitiated cumulative vehicle-routing problem (CCVRP) [9], our algorithm obtained the known best solutions in most of the instances (Tab. 20). The results were comparable with the proposed algorithms for CCVRP in terms of solution quality and running time. The comparison was very valuable, as these algorithms are considered to be the best metaheuristic algorithms for mTRP, mTRP-DC, or CCVRP in the literature.

Table 18
Average experimental results for mTRP with distance constraint [2]

Instances	VNS [2]		skewed-GVNS	
	Gap_1	*Time*	Gap_1	*Time*
pr1002_40_x	0.28	0.47	0.00	0.38
brd14051_40_x	0.26	0.45	0.00	0.34
fnl4461_40_x	0.26	0.35	0.00	0.29
d15112_40_x	0.30	0.80	0.00	0.36
nrw1379_40_x	0.47	0.81	0.00	0.74
pr1002_50_x	0.54	0.77	0.00	0.77
brd14051_50_x	0.63	0.90	0.00	0.71
fnl4461_50_x	0.59	0.75	0.00	0.71
d15112_50_x	0.44	0.46	0.00	0.72
nrw1379_50_x	0.54	0.72	0.00	0.73
pr1002_60_x	4.75	1.99	0.53	1.16
brd14051_60_x	3.65	1.83	0.66	1.17
fnl4461_60_x	2.69	1.64	0.48	1.12
d15112_60_x	3.60	2.02	0.56	1.16
nrw1379_60_x	4.21	1.88	0.63	1.12
pr1002_70_x	3.67	2.82	0.88	1.16
brd14051_70_x	3.59	3.07	0.50	1.15
fnl4461_70_x	0.62	2.84	3.82	1.16
d15112_70_x	0.69	2.53	2.73	1.16

Table 18 cont.

nrw1379_70_x	0.64	2.64	4.53	1.11
pr1002_80_x	0.68	9.25	3.87	3.58
brd14051_80_x	0.57	10.47	3.32	3.56
fnl4461_80_x	0.73	9.11	4.05	3.59
d15112_80_x	0.41	8.99	2.86	3.57
nrw1379_80_x	0.95	11.04	3.06	3.53
aver	1.43	3.14	1.29	1.40

Table 19
Results of our algorithm as compared to previous algorithms in mTRP instances [4, 16, 17]

Instances	VNS [4]	EA [17]	NGA [16]	skewed-GVNS	
				Best.Sol	*Time*
E30k3	2,108.26	–	–	2,097.30	0.26
E30k4	2,623.65	–	–	2,595.11	0.25
E51k5	2,623.65	3,320.00	2,209.64*	2,209.64	0.42
E76k10	2,786.07	4,094.00	2,310.09*	2,310.09	0.84
E76k14	2,201.13	3,762.00	2,005.40*	2,005.40	0.76
E76k15	2,400.17	–	–	2,377.50	0.83
E101k8	–	6,383.00	–	4,051.47	2.57
E101k14	–	5,048.00	–	3,288.53	2.78
P40k5	1,793.14	–	1,537.79*	1,537.79	0.31
P45k5	2,336.43	–	1,912.31*	1,912.31	0.35
P50k7	1,878.81	–	1,547.89*	1,590.47	0.68

Note that: * symbol '*' is the optimal value

Table 20
Results of our algorithm as compared to previous algorithms in CCVRP instances [9, 16]

Instance	CCVRP	skewed-GVNS	
		Best.Sol	*Time*
CMT1	2,230.35	2,230.35	1.80
CMT2	2,391.63	2,391.63	6.22
CMT3	4,045.42	4,045.42	18.87
CMT4	4,987.52	4,987.52	84.14
CMT5	5,809.59	5,838.32	287.40
CMT11	7,314.55	7,314.55	21.10
CMT12	3,558.92	3,558.92	15.41

To compare their time complexity, two perspectives can be considered as follows:

- Theoretical time complexity: the time complexity of skewed-GVNS is mainly to explore neighborhoods in RVNS. In RVNS, the 3-opt consumes more time than the other neighborhoods. Assume that, when k is its maximum number of runs in skewed-GVNS, the proposed algorithm requires $O(k \times level\_max \times IterMax \times O(n^3)) \times O(n^3)$ times (n is the number of vertices). The theoretical time complexity of skewed-GVNS is $O(n^3)$ times. In [4], Ban et al. showed that their algorithm required $O(k_1 \times Tsol \times |R_l| \times |R_h|)$. In the worst case when $Tsol = O(n)$ and $|R_l|$ and $|R_h| = n$, their algorithm required $O(n^3)$. In [17], they converted k-TRP to TRP by using a k-means clustering algorithm. They then solved each sub-problem by applying an integer linear programming formulation. The approach consumed a great deal of time for large instances; therefore, the approach was not suitable for the problem with large sizes. Finally, in [16], their algorithm that was based on VNS required $O(n^2)$ for 2-opt and $Tsol = O(n)$ for calculating the cost of a neighboring solution. Their time complexity was the same as that of skewed-GVNS.
- Time complexity by CPU times: because our skewed-GVNS and the [4, 16, 17] algorithms were run on computers with different configurations, comparing their running times was done relatively. Our running time grew quite moderately when compared to NGA [16], while it was the same as with the one of VNS [4] and better than the one of EA [17].

4. Discussions

A metaheuristic is a good approach for solving optimization problems with large sizes when it can produce a good solution quickly. Skewed-GVNS [13] is a popular scheme that succeeds in solving optimization problems [2, 4, 5, 14]; it is not a new method in the literature. However, to apply it to a specific problem, we have to answer many open questions:

- How can we use and combine neighborhoods with exploring and exploiting a good solution space?
- How many neighborhoods can be used to balance solution quality with running time?
- How do we escape the local optima?
- How do we escape from very large valleys?

Therefore, many algorithms have been developed on VNS variants in the literature; however, their solution qualities are different. Applying VNS variants and popular techniques to solve the problem successfully is our contribution in this work.

A good metaheuristic must balance between exploitation and exploration. Exploration generates diverse solutions to explore a search space, while exploitation focuses on searching in a good local region by exploiting it. To do this, the proposed algorithm brings the insertion heuristic (RVNS) and perturbation schemes together.

Moreover, two characteristics have been integrated into the proposed algorithm; the role of each component is summarized as follows:

- The initial solution is created based on the insertion heuristics scheme in the construction phase.
- The post phase improves the initial solution. In this phase, RVNS ensures the exploitation by investigating various neighboring solutions. The algorithm accepts a solution that is worse than the current one, which drives the search to escape huge valleys. In addition, the search is adjusted by using the penalty function. Therefore, though the search is enlarged, it is not far from the feasible regions that are explored.
- Shaking is used to move the search to completely new solution spaces, hoping to find better solutions.

The experimental results showed the good performance of the proposed algorithm. Summarily, there are three ways to use our algorithm:

- The first option is to run the construction phase. This way is the fastest; however, it cannot reach any feasible solution when the constraint is tight in many cases. This way should be used since the constraint is not tight.
- The second option is to run the proposed algorithm with one iteration. This way is the best choice for trading off the quality of the solution and running time.
- The last option is to run the proposed algorithm with 50 iterations. The way is the best choice for solution quality; however, it consumes much time for instances with large sizes.

Moreover, the algorithm reaches optimal values for those instances with 80 customers in an acceptable amount of time. It also obtains better solutions than the algorithms in [2, 4, 17] in many cases, while it is comparable with other algorithms [9, 16]. The experimental results indicate that the algorithm can be applied well for various problems.

5. Conclusions

We propose a metaheuristic algorithm to solve the SBDP-RC problem. The construction phase creates an initial solution, while the post phase develops it. The experimental results showed that the proposed algorithm can reach optimal solutions for problems with 80 customers. Additionally, our algorithm gives better solutions than previous algorithms in many cases. Finally, we suggest three ways to use the algorithm effectively. However, the running time does not meet practical applications; therefore, this will be our focus in future research.

6. Appendix

The representation of the solution that we use here is an array of l integer strings (l is the number of vehicles). The sequence of each number in the string is the order

of visiting these customers. In Figure 1, we have a representation of a solution as follows:

$T = (R_1, R_2, R_3)$
$R_1 = (v_0, v_2, v_5, v_8, v_{11}, v_0)$
$R_2 = (v_0, v_1, v_4, v_7, v_{10}, v_0)$
$R_3 = (v_0, v_3, v_6, v_9, v_{12}, v_0)$

Here, we give three solutions for ins-30-1 with three different α values (0.5, 0.75, 1):

$\alpha = 0.5$
$W(T) = 3367.16$
$R_1 =$ 0-9-11-16-21-22-23-25-15-24-17-12-14-29-18-19
$R_2 =$ 0-2-1-27-5-6
$R_3 =$ 0-7-4-28-8-10-13-26-20-3
$\alpha = 0.75$
$W(T) = 4315.47$
$R_1 =$ 0-4-14-17-25-23-22-15-24-11-16-21-13-10-5-26-6-20-3
$R_2 =$ 0-8-27-1-2
$R_3 =$ 0-7-28-12-18-29-19-9
$\alpha = 1$
$W(T) = 7568.36$
$R_1 =$ 0-18-21-17-15-25-9-12-2-7-8-24
$R_2 =$ 0-10-26-6-27-29
$R_3 =$ 0-4-1-11-22-13-19-20-23-5-16-3-14-28

Acknowledgments

This research is funded by Hanoi University of Science and Technology (HUST) under Grant Number T2021-PC-021.

References

[1] Ajam M., Akbari V., Salman F.S.: Minimizing latency in post-disaster road clearance operations, *European Journal of Operational Research*, vol. 277(3), pp. 1098–1112, 2019. doi: 10.1016/j.ejor.2019.03.024.

[2] Ban H.B.: A GRASP+VND Algorithm for the Multiple Traveling Repairman Problem with Distance Constraints, *Journal of Computer Science and Cybernetics*, vol. 33(3), pp. 272–288, 2017. doi: 10.15625/1813-9663/33/3/10511.

[3] Ban H.B.: A RVND+ILS Metaheuristic to Solve the Delivery Man Problem with Time Windows. In: A. Tagarelli, H. Tong (eds.), *Computational Data and Social Networks*, pp. 63–69, Springer International Publishing, Cham, 2019.

[4] Ban H.B., Nguyen D.N.: An effective GRASP+VND metaheuristic for the k-Minimum Latency Problem. In: *2016 IEEE RIVF International Conference on Computing & Communication Technologies, Research, Innovation, and Vision for the Future (RIVF)*, pp. 31–36, 2016. doi: 10.1109/RIVF.2016.7800265.

[5] Ban H.B., Nguyen D.N.: A meta-heuristic algorithm combining between tabu and variable neighborhood search for the minimum latency problem, *Fundamenta Informaticae*, vol. 156(1), pp. 21–41, 2017.

[6] Ban H.B., Nguyen D.N.: Metaheuristic for the Traveling Repairman Problem with Time Window Constraints. In: *2019 IEEE-RIVF International Conference on Computing and Communication Technologies (RIVF)*, pp. 1–6, 2019. doi: 10.1109/RIVF.2019.8713655.

[7] Bruni M., Beraldi P., Khodaparasti S.: A hybrid reactive GRASP heuristic for the risk-averse k-traveling repairman problem with profits, *Computers & Operations Research*, vol. 115, 104854, 2020. doi: 10.1016/j.cor.2019.104854.

[8] Feo T.A., Resende M.G.C.: Greedy randomized adaptive search procedures, *Journal of Global Optimization*, vol. 6, pp. 109–133, 1995.

[9] Ke L., Feng Z.: A two-phase metaheuristic for the cumulative capacitated vehicle routing problem, *Computers & Operations Research*, vol. 40(2), pp. 633–638, 2013. doi: 10.1016/j.cor.2012.08.020.

[10] Lu Y., Benlic U., Wu Q., Peng B.: Memetic algorithm for the multiple traveling repairman problem with profits, *Engineering Applications of Artificial Intelligence*, vol. 80, pp. 35–47, 2019.

[11] Luo Z., Qin H., Lim A.: Branch-and-price-and-cut for the multiple traveling repairman problem with distance constraints, *European Journal of Operational Research*, vol. 234(1), pp. 49–60, 2014.

[12] Martin O., Otto S.W., Felten E.W.: Large-Step Markov Chains for the Traveling Salesman Problem, *Complex Systems*, vol. 5, pp. 299–326, 1991.

[13] Mladenović N., Hansen P.: Variable neighborhood search, *Computers & Operations Research*, vol. 24(11), pp. 1097–1100, 1997.

[14] Mladenović N., Urošević D., Hanafi S.: Variable neighborhood search for the travelling deliveryman problem, *4OR*, vol. 11, pp. 57–73, 2013.

[15] *NEO*, 2013. http://neo.lcc.uma.es/vrp/vrp-instances/capacitated-vrp-instances.

[16] Nucamendi-Guilln S., Martnez-Salazar I., Angel-Bello F., Moreno-Vega J.M.: A mixed integer formulation and an efficient metaheuristic procedure for the k-Travelling Repairmen Problem, *Journal of the Operational Research Society*, vol. 67(8), pp. 1121–1134, 2016. doi: 10.1057/jors.2015.113.

[17] Ome Ezzine I., Elloumi S.: Polynomial formulation and heuristic based approach for the k-travelling repairman problem, *International Journal of Mathematics in Operational Research*, vol. 4(5), pp. 503–514, 2012.

[18] Pei J., Mladenović N., Urošević D., Brimberg J., Liu X.: Solving the traveling repairman problem with profits: A novel variable neighborhood search approach, *Information Sciences*, vol. 507, pp. 108–123, 2020.

[19] Will T.G.: *Extremal results and algorithms for degree sequences of graphs*, University of Illinois at Urbana-Champaign, 1993.

Affiliations

Ha-Bang Ban
Hanoi University of Science and Technology, School of Information and Communication Technology, Hanoi, Vietnam, BangBH@soict.hust.edu.vn

Hong-Phuong Nguyen
Hanoi University of Science and Technology, School of Information and Communication Technology, Hanoi, Vietnam, PhuongNH@soict.hust.edu.vn

Dang-Hai Pham
Hanoi University of Science and Technology, School of Information and Communication Technology, Hanoi, Vietnam, HaiPD@soict.hust.edu.vn (Corresponding Author)

Received: 05.07.2021
Revised: 08.08.2022
Accepted: 08.09.2022