Hybrid evolutionary search for the traveling repairman problem with profits

Yongliang Lu [a], Jin-Kao Hao [b,c], Qinghua Wu [a,*]

[a] *School of Management, Huazhong University of Science and Technology, No. 1037, Luoyu Road, Wuhan, China*

[b] *LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France*

[c] *Institut Universitaire de France, 1 Rue Descartes, 75231 Paris, France*

Information Sciences, June 2019

https://doi.org/10.1016/j.ins.2019.05.075

Abstract

The Traveling Repairman Problem with Profits is a node routing problem, where a repairman visits a subset of nodes of a weighted graph in order to maximize the collected time-dependent profits. In this work, we present the first population-based hybrid evolutionary search algorithm for solving the problem that combines: (i) a randomized greedy construction method for initial solution generation, (ii) a dedicated variable neighborhood search for local optimization, (iii) two crossover operators for solution recombination with an adaptive rule for crossover selection. Computational results on six sets of 120 benchmark instances from the literature demonstrate that the proposed algorithm achieves a high performance - it improves the best-known results (new lower bounds) for 39 instances, while matching the best-known results for the remaining cases. We investigate several main algorithmic ingredients to understand their impacts on the performance of the algorithm.

Keywords: Heuristics; Traveling repairman problem; Hybrid evolutionary search; Variable neighborhood search

1 Introduction

The classic Traveling Repairman Problem (TRP) can be defined as follows [5]. Let $G = (V, E)$ be a complete undirected graph, where $V = \{0, 1, ...n\}$

* Corresponding author.

Email addresses: luyongliang@hust.edu.cn (Yongliang Lu), jin-kao.hao@univ-angers.fr (Jin-Kao Hao), qinghuawu1005@gmail.com (Qinghua Wu).

is the set of nodes (i.e., locations) and $E = \{(i,j) : i,j \in V, i \neq j\}$ is the set of edges. Each edge $(i,j) \in E$ has a weight (travel cost) t_{ij} representing the travel time between two corresponding locations, which is symmetric, i.e., $t_{ij} = t_{ji}$, and satisfies the triangle inequality rule. TRP is to determine a Hamiltonian circuit over all the nodes that minimizes the sum of the waiting times $\sum_{i=0}^{n} l(i)$, where $l(i)$ denotes the arrival time at node i. In particular, the circuit must start and end at node 0, and $l(0) = 0$. TRP is also known in the literature as the Minimum Latency Problem (MLP) [22].

The Traveling Repairman Problem with Profits (TRPP), which was recently proposed in [14], is an extension of TRP. In TRPP, each node $i \in V$ is associated with a non-negative profit p_i, and node 0 is the depot ($p_0 = 0$). A repairman starts traveling from the depot and collects a revenue $p_i - l(i)$ at each visited location i. In this problem, each node can be visited at most once, and the repairman does not need to return to the depot (not all nodes need to be visited). When $l(i)$ becomes equal to or larger than p_i for some unvisited node i, node i will not be visited. The objective of TRPP is to visit a set of nodes such that the total collected revenue is maximized. One notices that if the profits of the problem instance are set to be extremely high relative to the travel costs, all nodes of the graph will be visited like in TRP.

From the perspective of computational complexity, TRPP is known to be NP-hard [14]. As a result, TRPP is an extremely challenging problem in terms of solution methods. In addition to its significance as a typical optimization problem, TRPP is notable for its ability to formulate a number of real-life applications. A simple application arises from the humanitarian and emergency relief logistics [1,14]. For example, after an earthquake hits a region, there are a number of villages that experience an urgent need for relief. The sooner the rescue team gets to a village, the more lives can be saved. Assume that p_i persons need to be rescued at village i, and at each time instance, a person could die. Consequently, the primary goal of the rescue team is to maximize the number of persons that will be rescued, $\sum_i [p_i - l_i]^+$, where l_i is the arrival time of the rescue team at village i.

Our literature review given in Section 2 indicates that unlike the popular TRP (MLP) for which numerous solution methods are available (see the reviews [6,3,22,40,41,43]), research on algorithms for TRPP is still in its infancy with very few advanced methods. The purpose of this work is to enrich the arsenal of solution methods for TRPP by presenting an effective hybrid evolutionary search algorithm (HESA) that integrates several complementary key components, including a randomized greedy construction procedure, a dedicated variable neighborhood search procedure, and two specific crossover operators with an adaptive rule for crossover selection. For an effective exploration of the search space, the variable neighborhood search procedure relies on a variable neighborhood descent with a random neighborhood ordering. Additional

exploration capacity is ensured by a combined use of two different crossover operators. Computational results on 120 benchmark instances from the literature reveal that HESA competes very favorably with the state-of-the-art algorithms [1,14]. Specifically, it reports improved best-known solutions (new lower bounds) for 39 instances, while reaching the best-known solutions for the remaining cases.

The remainder of the paper is organized as follows. Section 2 presents a review of the related literature. Section 3 shows a mathematical model of TRPP, followed by a detailed description of the proposed algorithm in Section 4. Computational results and comparisons with state-of-the-art approaches are given in Section 5. In Section 6, we examine several key elements of the algorithm to show their impacts on the algorithm, followed by concluding comments in Section 7.

2 Literature review

This section provides a brief overview of related node routing problems and a summary of the most recent advances in solution methods for TRPP.

Several related variants of TRPP have been considered in the literature. In [7], the authors introduced a stochastic variant of TRPP under uncertain travel times, which is then formulated as a nonlinear integer model and heuristically solved by means of a beam search heuristic. The only difference with TRPP is the consideration of uncertainty in travel times, which is constant in TRPP. Recently, in [8], the stochastic variant was extended to the case where a fleet of vehicles with unlimited capacity is available for the post-disaster humanitarian relief logistics. To approximately solve the problem, the authors further provided an iterated greedy heuristic algorithm.

TRPP is strongly related to TRP which has been extensively studied. Existing solution approaches for TRP include an approximation algorithm [22], a branch and bound algorithm [6], a dynamic programming algorithm [43], a greedy randomized adaptive search procedure/variable neighborhood search (GRASP/VNS) algorithm [40], a genetic algorithm [3], and a hybrid algorithm mixing GRASP, iterated local search (ILS) and VNS [41]. Furthermore, several TRP variants which are also related to TRPP have been studied in [2,12,20,32].

Among the routing problems with profits, the profitable tour problem (PTP) [21] and the orienteering problem (OP) [42] share similarities with TRPP. In PTP, the objective is to find a circuit that maximizes the difference between the total collected profit and the total travel time. The main difference between

TRPP and PTP is the arrival time in the objective function instead of the travel time. In [9], the authors provided a risk-averse formulation for PTP assuming that the route cost is uncertain and proposed a genetic algorithm and a tabu search to solve it.

In OP, the goal is to determine a path, limited in length, that visits some locations and maximizes the sum of the collected scores (profits). TRPP differs from OP in the sense that revenues are time-dependent, and no distance constraint exists on the path. Approaches for OP include a branch-and-cut algorithm [19], a GRASP algorithm with path relinking [10], a memetic algorithm [31], and a particle swarm optimization algorithm [15]. Recently, an extension of OP was presented in [18] by considering that the profit collection at each location requires either a number of discrete passes or a continuous amount of time to be spent at the location. This problem exhibits more similarities with TRPP due to its time-dependent profits. However, unlike TRPP, profits in that problem do not depend on arrival times.

Another closely related problem is known as the maximum collection problem with time-dependent rewards (MCPTDR) [17]. In this problem, node rewards are decreasing linear functions of time and the objective is to maximize the sum of the rewards collected at the nodes visited during the tour. The main difference between MCPTDR and TRPP is the decreasing rates. In TRPP, it is assumed that the decreasing rate is 1 for every node. In [17], a branch-and-bound procedure was presented for finding optimal solutions for small instances and a penalty-based greedy heuristic algorithm was developed to handle large-sized instances. Recently, MCPTDR was extended in [16] by considering the use of multiple agents to collect linearly decreasing rewards over time and proposing a two-phase heuristic algorithm to find good-enough solutions in a reasonable timeframe.

To the best of our knowledge, TRPP has only been studied in [1,14]. As the first study on the problem, the pioneer work of [14] introduced a mathematical model for TRPP (see Section 3) and a set of 120 benchmark instances with $n \in \{10, 20, 50, 100, 200, 500\}$ (see Section 5.1). Computational results were reported including optimal solutions for small instances ($n \leq 20$) achieved by solving the mathematical model with CPLEX (version 10.1), and sub-optimal solutions (lower bounds) for larger instances ($n \geq 50$) obtained with a tabu search (TS) algorithm with multiple neighborhoods. Recently, a new mixed-integer linear formulation was introduced in [1]. Unlike the model of [14] where the number of nodes to be visited is a parameter, the new model does not require such an input. As solution method, a powerful hybrid algorithm (called GRASP-ILS) combining GRASP and ILS was investigated in [1], which achieved remarkable computational outcomes by improving 46 previous best-known results out of the 120 benchmark instances.

In this work, we enrich the TRPP literature by proposing an effective algorithm for TRPP based on the hybrid evolutionary search framework and report improved new lower bounds on 39 large benchmark instances with $n \in \{100, 200, 500\}$. For our the comparative study (Section 5), we will use the above TRPP approaches (CPLEX, TS, GRASP-ILS) of [1,14] as our references.

3 Mathematical model

To formally state the TRPP problem, we recall the mathematical model proposed in [14], which assumes the number of nodes to be visited is known in advance. The model is based on the following notations as well as those presented in the introduction section.

Sets

N_0 : set of all demand nodes and the depot, $\{0, 1, ..., n\}$

N : set of all demand nodes, $\{1, ..., n\}$

Parameters

t_{ij} : travel time from node i to j, $(i, j) \in E$

p_i : profit associated with node i, $i \in N$

k : number of nodes to be visited, $1 \leq k \leq n$

Decision Variables

$$y_{ijl} = \begin{cases} 1, & \text{if edge } (i,j) \text{ is used as the } l\text{th edge} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

Model

$$\max f = \sum_{i=0}^{n} \sum_{j=1}^{n} \sum_{l=1}^{k} (p_j - (k+1-l)t_{ij}) y_{ijl} \tag{2}$$

subject to

$$\sum_{i=0}^{n} \sum_{l=1}^{k} y_{ijl} \leq 1 \qquad \forall j \in N \tag{3}$$

$$\sum_{i=0}^{n}\sum_{j=1}^{n} y_{ijl} = 1 \qquad \forall l \in \{1,2,...,k\} \tag{4}$$

$$\sum_{i=0}^{n} y_{ijl} = \sum_{i=1}^{n} y_{jil+1} \qquad \forall j \in N, \forall l \in \{1,2,...,k-1\} \tag{5}$$

$$\sum_{j=1}^{n} y_{0j1} = 1 \tag{6}$$

$$y_{ijl} \in \{0,1\} \qquad \forall i \in N_0, \forall j \in N, \forall l \in \{1,2,...,k\} \tag{7}$$

The objective function (2) seeks to maximize the total collected revenue by summing the difference between the profit of a node and the number of times the edge preceding that node is counted in the total latency. Constraints (3) ensure that each node can be visited at most once. Constraints (4) require that k nodes must be visited. The connectivity restriction is imposed by constraints (5), and constraints (6) impose the departure of the repairman from the depot. Constraints (7) define the nature of the decision variables. Notice that k is a parameter and needs to be set before solving the model. The optimal solution of TRPP is ensured by executing the model for each value of $1 \leq k \leq n$ and then selecting the best solution with the highest revenue.

4 Hybrid evolutionary search for TRPP

4.1 Main scheme

The proposed hybrid evolutionary search algorithm (HESA) follows the general memetic framework [24,36,37], which combines the exploration power of population-based search and the exploitation capacity of local optimization. Memetic algorithms have been successfully used to tackle a number of node routing problems including vehicle routing, capacitated arc routing and general routing with nodes, edges and arcs [39].

The general architecture of our HESA algorithm for TRPP is summarized in Algorithm 1. The algorithm starts with a population of initial solutions which are first generated by a randomized greedy construction procedure (Section 4.2) and further improved with a variable neighborhood search (VNS) procedure (Section 4.3). Then it iteratively improves the initial solutions by applying a crossover operator and a local optimization procedure. Precisely, the crossover operator combines (mates) two randomly selected parent solutions to generate an offspring solution. The local optimization procedure, that is a variable neighborhood search algorithm, is then applied to improve the offspring, followed by a population updating procedure to bring up to date the population with the offspring solution (Section 4.5). The algorithm terminates

Algorithm 1 Hybrid evolutionary search algorithm for TRPP

Require: $G = (V, E)$: a TRPP instance; p: population size; γ: number of runs of the randomized greedy construction procedure
Ensure: best solution S^* found so far
// Population initialization, Section 4.2
$Pool \leftarrow \emptyset$
for $i = 1, 2, ..., \gamma$ **do**
  $S \leftarrow Randomized\_Greedy\_Procedure(G)$ /* Section 4.2 */
  if $S \notin Pool$ **then**
    Add S into $Pool$
  end if
  if $|Pool| > p$ **then**
    Delete the worst solution from $Pool$
  end if
end for
for each solution $S \in Pool$ **do**
  $S \leftarrow$ VNS(S) /* Section 4.3 */
end for
// Main search procedure
while stopping condition is not met **do**
  Randomly select S_1 and S_2 from $Pool$
  $o \leftarrow$ Crossover(S_1, S_2) /* Section 4.4 */
  $S \leftarrow$ VNS(o) /* Section 4.3 */
  $Pool \leftarrow Population\_Update(Pool, S)$ /* Section 4.5 */
end while
$S^* \leftarrow Best(Pool)$
Return S^*

when a time limit is reached. A detailed description of the algorithmic components is provided in the following sections.

4.2 Population initialization

To generate an initial solution S (i.e., a path composed of a sequence of nodes), we propose a randomized greedy construction procedure based on the nearest neighbor heuristic for routing problems that operates according to the following steps:

(1) Initialize S by including the predefined starting node and another randomly selected node that is connected to the starting node;
(2) Randomly select a node among the three nearest nodes relative to the last node of the path S and insert it at the end of the path S. This process is repeated until all the nodes are included in S;
(3) Use the RVND procedure (Section 4.3.2) to improve S until a local optimum is reached.

A particular feature of this initial construction procedure is that the travel cost t_{ij} is modified aiming to favor a route going from a node to another node with large profit. The modified travel cost d_{ij} is given as

$$d_{ij} = t_{ij} + C \times T \times \frac{P - p_j}{P},$$

where C is a parameter, the constant $T = \frac{\sum_{(i,j)\in E} t_{ij}}{|E|}$ is used to normalize each edge and $P = \sum_{i\in V} p_i$. Similar ideas are presented in [25] for the multi-commodity pickup-and-delivery traveling salesman problem.

The construction procedure is run γ times ($\gamma \geq p$, p is the population size) to obtain a set of locally optimal solutions. The best p solutions are then improved by a VNS procedure (Section 4.3) and form the initial population. This initialization procedure produces an initial population of relatively high quality.

Note that a solution generated with the construction procedure includes all the nodes with the aim of employing them in the neighborhood structures (Section 4.3.1) effectively. Nodes that do not need to be visited in the solution are easily identified, and are not considered when calculating the objective value of the solution in the HESA algorithm. These unvisited nodes are ignored in the final output of the algorithm.

4.3 Variable neighborhood search

Algorithm 2 Variable neighborhood search

Require: S_0: an initial solution; k_{max}: max allowed iterations without improvement
Ensure: the best solution S^* found

```
k ← 0
S* ← S_0
S ← S_0
while k < k_max do
    S ← RVND(S) /* Section 4.3.2 */
    if f(S) > f(S*) then
        S* ← S
        k ← 0
    else
        k ← k + 1
    end if
    S ← Shake(S, S*) /* Section 4.3.3 */
end while
return S*
```

The general variable neighborhood search (VNS) method [34] has been used to

solve several routing problems [26,28,35,40]. The HESA algorithm proposed in this work for TRPP includes a VNS procedure for improving initial solutions and offspring solutions generated by the crossover operators. Following [34,41], our VNS procedure combines the variable neighborhood descent with random neighborhood ordering (RVND) to find local optima (Section 4.3.2) and a shaking procedure to escape from local optimality traps (Section 4.3.3).

As outlined in Algorithm 2, our VNS procedure repeats the RVND procedure and the shaking procedure to explore new candidate solutions. The best-found solution S^* is updated each time a still better solution is found. If no improvement with respect to S^* is achieved during k_{max} consecutive iterations, the best-found solution S^* is returned as the final output of the VNS procedure.

4.3.1 Move operators

Our VNS procedure adopts the following five move operators, which are commonly used for solving the classical traveling salesman problem:

- **Insertion (N_1):** remove one node and insert it at the best possible position in the route;
- **Swap (N_2):** exchange two nodes;
- **Or-opt (N_3):** remove n consecutive nodes and insert them at the best possible position in the tour (where $n = \{2, 3\}$);
- **Two-opt (N_4):** remove two non-adjacent edges in the tour, and reconnect the two sub-tours (see Figure 1(a));
- **Double-bridge (N_5)**: delete four edges in the tour, reconnect the four sub-tours keeping the orientation of the four sub-tours (see Figure 1(b)).

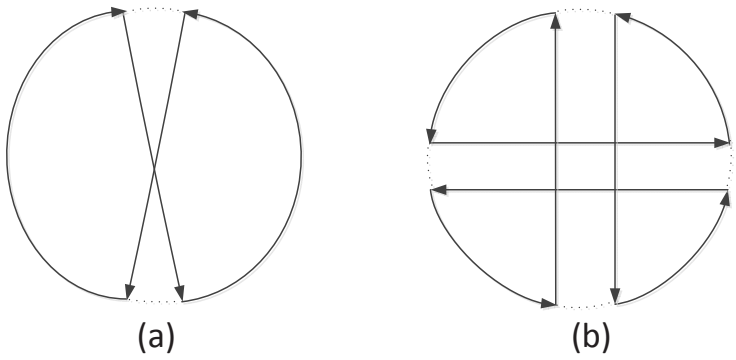


Fig. 1. (a) 2-opt move; (b) Double-bridge move

We use $N_1 - N_5$ to denote the five neighborhoods induced by these move operators. Considering an instance with n nodes (except the depot node), there are $n - 1$ possible positions for the insertion of each node. Therefore, the size of the neighborhood N_1 is clearly bounded by $O(n^2)$. Similarly, the number of pairs of nodes is $n(n-1)/2$, so the size of the neighborhood N_2 is

bounded by $O(n^2)$. As described in [38,13], both Or-opt and Two-opt operators also yield $O(n^2)$ neighboring solutions.

To evaluate the objective value of a neighboring solution, a full computation from scratch requires $O(n)$ time. However, it can be done in a more efficient way using the binary indexed tree data structures, which allows to reduce the complexity to $O(log\ n)$. Therefore, examining each of the four neighborhoods $N_1 - N_4$ requires a time of $O(n^2 log\ n)$.

The double-bridge operator is only used to perturb solutions (see Section 4.3.3), which constructs a new tour in constant time by changing a constant number of edges. However, the objective function evaluation for the new tour increases the complexity of the operator to $O(log\ n)$.

According to the analysis, since no fast and incremental evaluation of neighboring solutions is known, the objective function evaluation becomes expensive when many local search iterations are performed (which is usually the case).

The following subsections explain how these neighborhoods are used in the RVND procedure and the shaking procedure.

4.3.2 RVND procedure

Algorithm 3 RVND procedure

Require: S: current solution
Ensure: updated S
1: $NL = \{N_1, N_2, N_3, N_4\}$
2: **while** $NL \neq \emptyset$ **do**
3: Choose a neighborhood $N \in NL$ at random
4: $(S, improve\_tag) \leftarrow LocalSearch(S, N)$ /* See Algorithm 4 */
5: **if** $improve\_tag = true$ **then**
6: $NL = \{N_1, N_2, N_3, N_4\}$
7: **else**
8: Remove N from the NL
9: **end if**
10: **end while**
11: **return** S

The RVND procedure explores the neighborhoods $N_1 - N_4$ to attain a local optimum (see Algorithm 3). After initializing the neighborhood list ($NL = \{N_1, N_2, N_3, N_4\}$), RVND enters into its main loop. At each iteration, a neighborhood N is chosen at random from NL and then explored by the RVND procedure. For this, the neighboring solutions in N are visited in a random order according to the first improvement strategy to make solution transitions. If an improving solution is found in the current neighborhood, NL is re-populated with the four neighborhoods $N_1 - N_4$. Otherwise, the current

Algorithm 4 Local Search procedure

Require: S: current solution; N: current neighborhood
Ensure: updated S
1: $improve\_tag \leftarrow false$
2: Randomly shuffle all moves in neighborhood N
3: **for** each move $mv \in N$ **do**
4: $S^{'} \leftarrow S \oplus mv$
5: **if** $f(S^{'}) > f(S)$ **then**
6: $S \leftarrow S^{'}$
7: $improve\_tag \leftarrow true$
8: **end if**
9: **end for**
10: **return** $(S, improve\_tag)$

neighborhood N is removed from NL. If none of the neighboring solutions in $N_1 \cup \cdots \cup N_4$ improves on the quality of the current solution S, S is a local optimum with respect to $N_1 - N_4$ and is returned as the final solution of the RVND procedure.

4.3.3 Shaking procedure

Algorithm 5 Shaking procedure

Require: S: current solution; S^*: best-found solution; η: strength of shake; I_r: interval of threshold ratio
Ensure: S: perturbed solution
1: $w \leftarrow 0$
2: **while** $w < \eta$ **do**
3: Randomly pick a neighboring solution $S^{'} \in N_5(S)$
4: $r \leftarrow random\_select(I_r)$
5: **if** $f(S^{'}) > (1 - r) \times f(S^*)$ **then**
6: $S \leftarrow S^{'}$
7: $w \leftarrow w + 1$
8: **end if**
9: **end while**
10: **return** S

To escape from local optima, we call for a shaking procedure, in which the double-bridge operator, introduced in [30] for TSP, is used to perturb solutions. A good property of the double-bridge operator is that it is not easy for the other operators to undo the changes it performs on the tour, which means there is little chance of going back to the last local optimum. In order to control the degree of solution degradation, we use a threshold function to accept non-improving double-bridge moves. As outlined in Algorithm 5, a neighboring solution $S^{'}$ is picked at random from the neighborhood N_5 defined by the double-bridge operator and replaces the current solution S if $f(S^{'}) > (1-r) \times f(S^*)$, where S^* is the best-found solution and r is a threshold

ratio randomly chosen among the values of a given interval I_r between suitably chosen upper limits and lower limits. The shaking procedure performs η double-bridge moves, where η is a parameter called the shake strength.

4.4 Crossover operators

At each generation of our HESA algorithm, a crossover operator is applied to create a new offspring solution by recombining two parent solutions randomly selected from the population. It has been commonly recognized that the success of a population-based memetic approach crucially depends on the recombination operator [24], which should not only generate diversified solutions but also transfer meaningful components from parents to offspring. Our HESA algorithm employs two crossover operators, which are applied according to a probabilistic mechanism. As the analysis presented in Section 6.3 shows, the combined use of the two crossovers constitutes an important ingredient of the proposed algorithm.

The adopted crossover operators are inspired by those presented in [27] for a job scheduling problem and are described as follows.

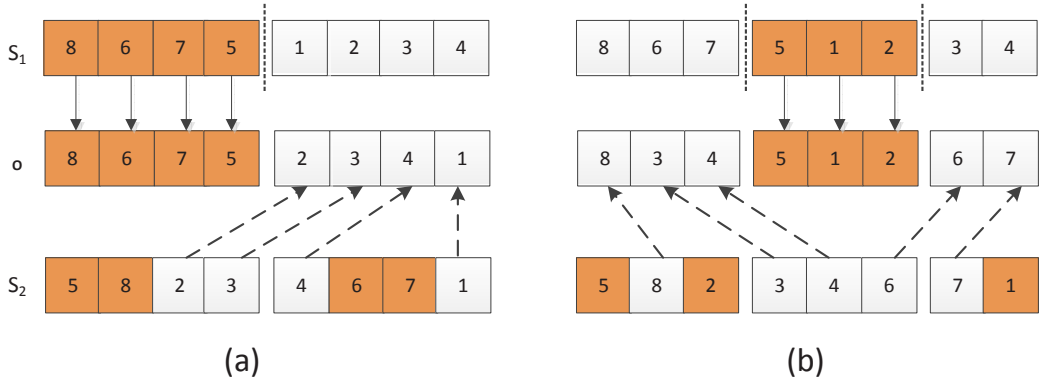


Fig. 2. (a)An example of one-point crossover; (b) An example of two-point crossover

One-point Crossover: In this operator, one cutting point is randomly selected at first. The node sub-sequence on one side of the cutting point is inherited from one parent to the offspring. The other nodes are copied to the offspring in the order they appeared in the other parent. Figure 2(a) shows an example of the one-point operation on two parent solutions S_1 and S_2 where the selected crossing point is between positions 5 and 1. The offspring o is generated by first copying the left sub-sequence of S_1, and then copying in the same order the remaining nodes from S_2.

Two-point Crossover: In this operator, instead of one cutting point, two cutting points are randomly selected to divide the parent solutions S_1 and S_2. The offspring o is obtained by first transferring the sub-sequence between

the two cutting points from S_1 to o, and then copying in the same order the remaining nodes from S_2. Figure 2(b) gives an example of the two-point crossover operator.

At each generation, HESA selects one of the two crossover operators with an adaptive probabilistic mechanism inspired by a technique from [33]. The probability of selecting any of the two crossovers is proportional to the number of times the solution produced with the given crossover is introduced into the population during the population update phase. For each of the two crossovers i, we thus maintain an counter q_i that indicates the number of times a solution created with crossover i has been introduced into the population pool. The probability of selecting a crossover i is then determined with the following formula:

$$P(i) = \frac{50 + q_i}{100 + q_1 + q_2}$$

One observes that the higher the value of q_i, the more probable the corresponding crossover is selected. Note that at the beginning of the search, the two crossovers have the same probability of being selected, and the probability calculation assumes that 100 high-quality solutions have been generated and that each crossover method has contributed to exactly 1/2 of them.

4.5 Population updating rule

For each new solution S generated by the VNS procedure, we decide whether S should be introduced into the population and which member from the population should be replaced according to the following rule. If S is distinct from any existing solution in the population and is better than the worst solution in the population, then S replaces this worst member from the population. Otherwise, the population is kept unchanged.

5 Computational experiments

In this section, we assess the performance of the proposed HESA algorithm on well-known benchmark instances, and show comparisons with the existing state-of-the-art heuristic approaches for TRPP.

5.1 Benchmark and experimental protocol

For performance assessments, we use six sets of 120 benchmark instances which were introduced in [14]. These data sets were derived from the TRP benchmark [40]. Each data set consists of 20 problem instances with 10, 20, 50, 100, 200 and 500 nodes, apart from the depot node, respectively. Besides, for each of these instances, a non-negative profit p_i was allocated to each node $i \in V \setminus \{0\}$ using the uniform distribution in the interval $(L, U]$, where $L = \min_{i \in V \setminus \{0\}}\{t_{0i}\}$, $U = \frac{n}{2} \times \max_{i \in V \setminus \{0\}}\{t_{0i}\}$, and n is the number of nodes.

Our HESA algorithm was programmed in C++ and compiled with a g++ compiler. All experiments were performed on a computer with an Intel Xeon(R) CPU E5-2695 v4 processor (2.1 GHz CPU and 2 GB RAM) running under Linux operating system. To evaluate the performance of HESA, we perform comparisons with two state-of-the-art TRPP heuristics from the literature:

- The GRASP with iterated local search (GRASP-ILS) algorithm from [1]. To our knowledge, GRASP-ILS is the current best-performing heuristic for TRPP. The results reported by GRASP-ILS were obtained after five executions per instance on a 2.20 GHz Intel core duo 2 T 7500 computer.
- The tabu search (TS) algorithm presented in [14]. The tests were carried out on a DELL Optiplex 760, Intel(R) Core(TM) 2 Duo 3.00 GHz, 4.00 GB RAM, 64-bit Operating System, but the number of runs per test instance is not specified in [14].

It should be noted that a fully fair comparative analysis with the existing TRPP algorithms from the literature is not a straight-forward task because of the differences in computing hardware, programming language, termination criterion, etc. To make the comparison as fair as possible, we use a procedure described in [11] to scale the CPU times reported for the two heuristics in the corresponding papers. The procedure is based on the assumption that the CPU speed is approximately linearly proportional to the CPU frequency. Then we adopt the scaled times used by the current best-performing GRASP-ILS algorithm [1] as the cutoff times of our HESA algorithm. Like [1], we perform five independent runs of HESA per test case using the above stopping condition.

5.2 Parameter tuning

The setting of parameters is given in Table 1. As suggested in [1], we set $k_{max} = 30$. According to our experimental experience, as the general local search algorithms (TS, ILS and VNS) are very costly for large instances with 500 nodes, we set $k_{max} = 5$ for instances with 500 nodes to save time. The

threshold ratio r is randomly chosen among the values in the given intervals I_r during the search process.

The rest of the parameters (C, p, γ, η) were tuned with Iterated F-race (IFR) [4], an automatic configuration method that is part of the IRACE package [29]. The tuning was performed on the 20 instances with 200 nodes. For each parameter, IFR requires a limited set of values as input to choose from (column "Considered values"). The total time budget for IRACE was set to 1000 executions, with the time limit per HESA execution set to the scaled CPU times used by GRASP-ILS [1]. The suggested setting of parameters by IFR is presented in Column "Final value".

Table 1
Parameter tuning results

Parameter	Section	Description	Considered values	Final value
C	4.2	a variable in the initialization phase	{1,10,20}	10
p	4.2	population size	{5,10,20}	5
γ	4.2	number of runs of the randomized construction procedure	{10,30,50}	50
k_{max}	4.3	max allowed iterations without improvement	-	{5,30}
η	4.3.3	strength of shake	{5,10,15}	10
I_r	4.3.3	interval of threshold ratio values	-	[0.1,0.3]

5.3 Computational results and comparisons with existing heuristics

We next provide computational comparisons with the two state-of-the-art TRPP heuristics (TS and GRASP-ILS) on each instance from the TRPP benchmark set.

Table 2
Results for problem instances with 10 and 20 nodes.

	n=10				n=20			
Instance	Exact	TS	GRASP-ILS	HESA	Exact	TS	GRASP-ILS	HESA
1	2520	2520	2520	2520	8772	8772	8772	8772
2	1770	1770	1770	1770	10174	10174	10174	10174
3	1737	1737	1737	1737	7917	7917	7917	7917
4	2247	2247	2247	2247	7967	7967	7967	7967
5	2396	2396	2396	2396	7985	7985	7985	7985
6	1872	1872	1872	1872	7500	7500	7500	7500
7	1360	1360	1360	1360	9439	9439	9439	9439
8	1696	1696	1696	1696	7999	7999	7999	7999
9	1465	1465	1465	1465	6952	6952	6952	6952
10	1014	1014	1014	1014	8582	8582	8582	8582
11	1355	1355	1355	1355	7257	7257	7257	7257
12	1817	1817	1817	1817	6857	6857	6857	6857
13	1585	1585	1585	1585	7043	7043	7043	7043
14	2122	2122	2122	2122	6964	6964	6964	6964
15	1747	1747	1747	1747	6270	6270	6270	6270
16	1635	1635	1635	1635	8143	8143	8143	8143
17	2025	2025	2025	2025	10226	10226	10226	10226
18	1783	1783	1783	1783	7625	7625	7625	7625
19	1797	1797	1797	1797	7982	7982	7982	7982
20	1771	1771	1771	1771	7662	7662	7662	7662

Table 2 presents the results for the two sets of 40 small instances with 10 and 20 nodes. For each instance, we indicate the known-optimal value reported in [14] (Columns 2 and 6) and the best objective values of the reference algorithms TS (Columns 3 and 7) and GRASP-ILS (Columns 4 and 8). The results of our HESA algorithm are shown in columns 5 and 9. We observe from the

table that HESA attains the same performance as TS and GRASP-ILS, by achieving the optimal solutions for all these small instances. Computational times are not reported, since they are very low (generally under one second) for these (easy) instances.

Table 3
Results for problem instances with 50 nodes.

	TS		GRASP-ILS		HESA			
Instance	f_{best}	$t(s)$	f_{best}	$t(s)$	f_{best}	f_{avg}	$t(s)$	$Gap(\%)$
1	50921	14.1	50921	11.5	50921	50921.0	11.5	0.00
2	52594	12.6	52594	8.4	52594	52594.0	8.4	0.00
3	52144	12.5	52144	9.4	52144	52144.0	9.4	0.00
4	45465	12.4	45465	7.3	45465	45465.0	7.3	0.00
5	45489	11.2	45489	8.4	45489	45489.0	8.4	0.00
6	55630	12.8	55630	7.3	55630	55630.0	7.3	0.00
7	44300	9.2	44302	9.4	44302	44302.0	9.4	0.00
8	55753	12.3	55801	10.5	55801	55801.0	10.5	0.00
9	44964	11.3	44964	10.5	44964	44964.0	10.5	0.00
10	47071	9.8	47071	9.4	47071	47071.0	9.4	0.00
11	51912	10.3	51912	9.4	51912	51912.0	9.4	0.00
12	53567	14.5	53567	8.4	53567	53567.0	8.4	0.00
13	46830	12.6	46830	9.4	46830	46830.0	9.4	0.00
14	52665	53.1	52665	8.4	52665	52665.0	8.4	0.00
15	58856	14.0	58856	10.5	58856	58856.0	10.5	0.00
16	49754	12.1	49754	12.6	49754	49754.0	12.6	0.00
17	42525	9.6	42525	8.4	42525	42525.0	8.4	0.00
18	40536	11.9	40536	9.4	40536	40536.0	9.4	0.00
19	55346	15.0	55346	10.5	55346	55346.0	10.5	0.00
20	61286	13.4	61286	9.4	61286	61286.0	9.4	0.00

Table 4
Results for problem instances with 100 nodes.

	TS		GRASP-ILS		HESA			
Instance	f_{best}	$t(s)$	f_{best}	$t(s)$	f_{best}	f_{avg}	$t(s)$	$Gap(\%)$
1	209952	119.7	209952	92.2	209952	209952.0	92.2	0.00
2	196318	101.3	196318	135.1	196318	196311.2	135.2	0.00
3	211937	126.7	211937	107.9	211937	211937.0	108.0	0.00
4	217685	112.1	217685	89.0	217685	217685.0	89.1	0.00
5	215119	169.2	215119	131.0	215119	215119.0	131.0	0.00
6	228687	144.0	228687	102.7	228687	228687.0	102.8	0.00
7	200060	347.3	200064	128.9	200064	200056.0	129.0	0.00
8	205760	117.8	205760	119.4	205760	205760.0	119.5	0.00
9	226240	93.7	226240	112.1	226240	226240.0	112.1	0.00
10	218202	162.5	218202	88.0	218202	218202.0	88.0	0.00
11	212503	126.8	212503	125.7	212503	212503.0	125.8	0.00
12	222249	148.0	222249	98.5	222249	222249.0	98.6	0.00
13	206878	145.2	206957	89.0	206957	206957.0	89.1	0.00
14	215690	126.8	215690	85.9	215690	215690.0	86.0	0.00
15	213758	157.7	213811	89.0	**214041**	214041.0	89.1	-0.11
16	214036	152.0	214036	89.0	214036	214036.0	89.0	0.00
17	223636	136.6	223636	118.4	223636	223636.0	118.5	0.00
18	192849	122.5	192849	97.4	192849	192849.0	97.4	0.00
19	206755	174.9	206755	128.9	206755	206755.0	128.9	0.00
20	198842	213.4	198908	104.8	198908	198908.0	104.8	0.00

Tables 3-6 present the results for the four other sets of benchmark instances involving 50, 100, 200 and 500 nodes. For each instance and approach, we show the best objective value (f_{best}) and the average computation time in seconds (t_s). Column 'Gap(%)' denotes the percentage gap between the best-known objective value (f_{bk}) compiled from columns 2 and 4 and the best objective value obtained with HESA, computed as $100 \times (f_{bk} - f_{best})/f_{bk}$. The entries in bold indicate the cases where HESA improves on the best-known result from the literature (i.e., when the percentage gap is negative). We do not compare the performances of the three algorithms in terms of the average solution quality, as the average results are not reported for TS in [14] and GRASP-ILS in [1].

Table 5
Results for problem instances with 200 nodes.

	TS		GRASP-ILS		HESA			
Instance	f_{best}	$t(s)$	f_{best}	$t(s)$	f_{best}	f_{avg}	$t(s)$	$Gap(\%)$
1	877523	1454.2	877530	1097.9	**877610**	877597.6	1098.9	-0.01
2	901419	1434.4	901559	1047.6	**901898**	901762.6	1048.2	-0.04
3	888082	1431.7	888393	1143.0	888393	888393.0	1144.1	0.00
4	873530	1440.1	873840	1065.4	**873910**	873757.6	1068.8	-0.01
5	849205	1432.7	849231	1110.5	**849358**	849358.0	1111.7	-0.01
6	816452	1434.7	816910	1108.4	**816923**	816904.4	1109.0	-0.00
7	783367	1448.7	783594	1082.2	**784120**	784107.8	1082.4	-0.07
8	837908	1432.7	837938	1114.7	**838023**	837828.6	1115.4	-0.01
9	891035	1439.7	891071	1043.4	**891203**	891203.0	1046.0	-0.01
10	846662	1430.2	847235	1062.3	**847303**	847177.8	1065.4	-0.01
11	803605	1430.0	803671	1082.2	**804851**	804630.2	1082.4	-0.15
12	808313	1428.8	808686	1108.4	**808966**	808963.0	1108.6	-0.03
13	861375	1433.5	861604	1072.8	**861729**	861713.0	1075.5	-0.01
14	849942	1429.4	850118	1102.1	**850507**	850330.2	1106.5	-0.05
15	847181	1455.8	847755	1088.5	**848006**	847861.2	1092.0	-0.03
16	853315	1445.0	854075	1121.0	854075	854046.6	1122.2	0.00
17	860638	1429.4	861175	1040.3	**861747**	861327.4	1042.4	-0.07
18	841922	1433.0	842283	1138.8	**842953**	842817.6	1142.2	-0.08
19	822308	1432.6	822397	1063.3	**822855**	822810.6	1065.6	-0.06
20	902948	1428.1	903772	1041.3	**904326**	904308.6	1042.0	-0.06

Table 6
Results for problem instances with 500 nodes.

	TS		GRASP-ILS		HESA			
Instance	f_{best}	$t(s)$	f_{best}	$t(s)$	f_{best}	f_{avg}	$t(s)$	$Gap(\%)$
1	6596427	2857.1	6610423	1905.6	**6630382**	6615138.8	1911.1	-0.30
2	6382793	2856.0	6409937	1995.7	**6418393**	6390801.6	2001.5	-0.13
3	6331860	2856.0	6336705	1981.0	**6383933**	6370222.8	1985.6	-0.75
4	6589154	2856.7	6620316	1992.6	**6628111**	6604666.2	1996.4	-0.12
5	6751588	2857.2	6778323	2075.3	**6780912**	6756765.6	2079.7	-0.04
6	6104076	2857.2	6123548	1908.8	**6128506**	6119310.4	1911.0	-0.08
7	6694671	2856.9	6778295	2024.0	**6778911**	6769301.2	2026.1	-0.01
8	6506056	2857.0	6518013	1870.0	**6534078**	6509147.2	1871.2	-0.25
9	6530832	2857.1	6560088	1931.8	**6581036**	6569852.2	1934.6	-0.32
10	6686395	2856.6	6713796	1917.1	**6714813**	6685038.4	1922.9	-0.02
11	6689340	2856.1	6719607	2082.7	**6739761**	6717390.6	2087.1	-0.30
12	6466482	2855.9	6505037	2005.1	**6507506**	6489918.4	2011.1	-0.04
13	6341393	2856.5	6385898	1926.6	**6386100**	6376000.4	1931.1	-0.00
14	6718962	2856.4	6720088	1907.7	**6758997**	6741078.0	1910.5	-0.58
15	6719534	2856.6	6731903	1853.2	**6759142**	6745388.6	1856.2	-0.40
16	6548694	2855.9	6606948	1954.9	**6610001**	6598459.0	1955.5	-0.05
17	6575817	2856.8	6606835	2090.0	**6608356**	6605565.8	2095.5	-0.02
18	6556239	2855.9	6563180	1977.9	**6581089**	6561545.8	1983.6	-0.27
19	6586219	2857.0	6614261	2012.5	**6617187**	6606727.0	2017.4	-0.04
20	6616931	2840.0	6636568	1848.0	**6638626**	6625809.4	1853.6	-0.03

Table 7
Summary of results.

	TS			GRASP-ILS			HESA		
n	Best-known sol.	Average Gap(%)	Average time(s)	Best-known sol.	Average Gap(%)	Average time(s)	Best-known sol.	Average Gap(%)	Average time(s)
10	20/20	0.00	< 1	20/20	0.00	< 1	20/20	0.00	< 1
20	20/20	0.00	< 1	20/20	0.00	< 1	20/20	0.00	< 1
50	20/20	0.00	14.2	20/20	0.00	9.4	20/20	0.00	9.4
100	16/20	0.01	149.9	19/20	0.01	106.6	20/20	0.00	106.7
200	0/20	0.07	1436.2	2/20	0.04	1086.7	20/20	0.00	1088.5
500	0/20	0.60	2855.8	0/20	0.19	1963.0	20/20	0.00	1967.1

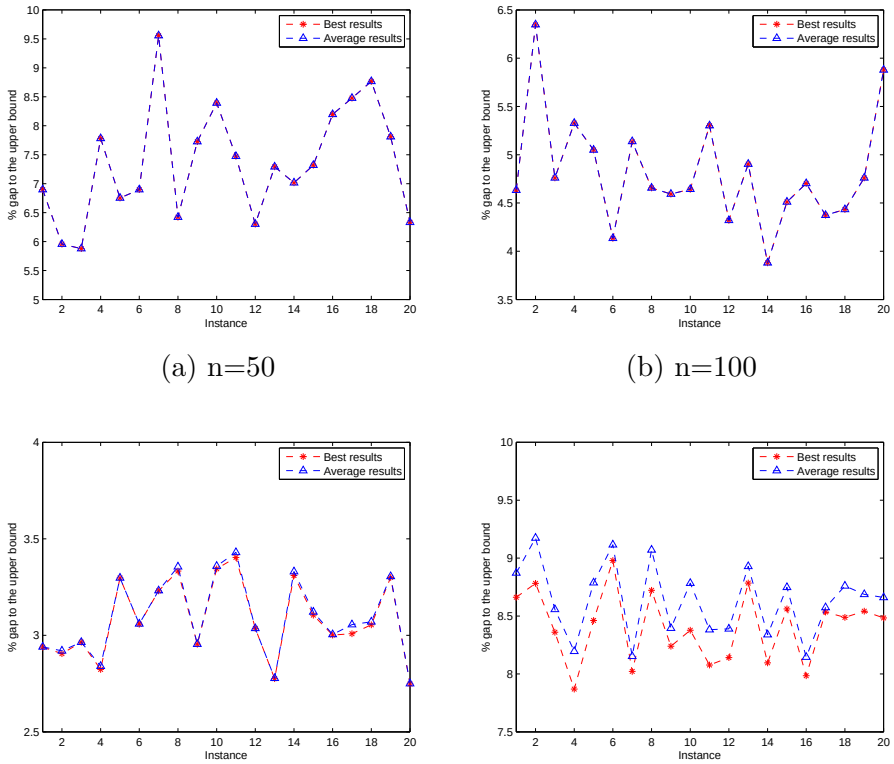
The results of Tables 3-6 show that the HESA algorithm reports improved lower bounds for 39 out of these 80 instances with unknown optima, and matches the current best-known results on the remaining instances. Moreover, we observe that 38 of these 39 improved results concern large instances with 200 and 500 nodes.

To verify the statistical significance of the observed differences, we apply the non-parametric Friedman test to compare the three algorithms on the two

sets of large instances with 200 and 500 nodes. The Friedman test in terms of best objective values reveals a p-value of 3.27e-09 ($<<$ 0.05) on the set of instances with 200 nodes, and a p-value of 2.06e-09 ($<<$ 0.05) on the set of instances with 500 nodes, confirming the statistical significance of the observed differences.

To conclude, Table 7 summarizes the results reported with the compared approaches on the six sets of 120 benchmark instances. For each set, we show the results reported by each method, including the number of cases for which the best-known result was reached or improved, the average gap between the updated best-known and the best reported result, and the average computation time in seconds required. Table 7 reveals the largest performance gap between TS/GRASP-ILS and HESA on the last two sets of medium and large size instances, which constitute the most challenging benchmark instances.

5.4 Comparison with upper bound



(a) n=50 (b) n=100

(c) n=200 (d) n=500

Fig. 3. Gap to the upper bound for each individual benchmark set

In [14], a simple and crude upper bound is presented for each instance, which can be used to contrast the quality of solutions found by a heuristic algo-

rithm (which are lower bounds). This upper bound is simply calculated by the difference between an upper bound for the collected profits and a lower bound for the total latency. To further investigate the performance of HESA, we provide a comparison with the upper bounds (provide in [1]) on the instances with unknown optima. Figure 3 shows the percentage gap between the upper bound and the best/average result reported with HESA for theses instances. The percentage gap between two results L_1 and L_2 is computed as $gap = 100 \times \frac{L_1 - L_2}{L_1}$.

We observe that the gap varies between 2.5% and 10%. Furthermore, the gap is less than 3.5% for all cases with 200 nodes. These results reveal that our HESA algorithm can lead to high-quality solutions (or near-optimal solutions). Besides, the average results are the same as the best ones on almost all of the instances with $n = 50$, 100 and 200, which further confirms the robustness of the algorithm.

5.5 Comparison with solutions reported by CPLEX

In [14] (2013), optimal solutions are reported for the 40 small instances with $n = 10$ and 20 by running the mixed integer programming solver CPLEX (version 10.1) on the mathematical model presented in Section 3. However, this approach failed to solve any larger instance with $n \geq 50$. Given that other advanced technologies has been integrated in CPLEX since 2013, one may wonder whether more instances can be optimally solved by today's CPLEX solver. For this purpose, we perform an additional experiment where we run CPLEX 12.6 with the model of [14] to try to solve the 20 instances with $n = 50$, with a maximum computation time set to three hours (10800 seconds) per instance.

Table 8 gives the detailed results of CPLEX 12.6 together with the results of our HESA algorithm. The first column indicates the instance name and the second column shows the best lower bound obtained with CPLEX 12.6, followed by the required computation time in seconds. Columns 4-7 present the computational results of HESA: the best (f_{best}) and the average objective value (f_{avg}) over 5 independent runs, the average computation time in seconds needed to attain the result, and the percentage gap between the best objective values obtained with CPLEX and HESA defined as

$$gap = 100 \times \frac{f_{CPLEX} - f_{HESA}}{f_{CPLEX}}.$$

If the gap is negative, HESA outperforms CPLEX, and HESA is outperformed otherwise.

Table 8
Computational results of CPLEX 12.6 and HESA on the 20 instances with $n = 50$

Instance	CPLEX		HESA			
	f	$t(s)$	f_{best}	f_{avg}	$t(s)$	$Gap(\%)$
1	49895	>10800	50921	50921.0	11.5	-2.06
2	51630	>10800	52594	52594.0	8.4	-1.87
3	51174	>10800	52144	52144.0	9.4	-1.90
4	44067	>10800	45465	45465.0	7.3	-3.17
5	44921	>10800	45489	45489.0	8.4	-1.26
6	54119	>10800	55630	55630.0	7.3	-2.79
7	43772	>10800	44302	44302.0	9.4	-1.21
8	54400	>10800	55801	55801.0	10.5	-2.58
9	44202	>10800	44964	44964.0	10.5	-1.72
10	46092	>10800	47071	47071.0	9.4	-2.12
11	51376	>10800	51912	51912.0	9.4	-1.04
12	53191	>10800	53567	53567.0	8.4	-0.71
13	45701	>10800	46830	46830.0	9.4	-2.47
14	51183	>10800	52665	52665.0	8.4	-2.90
15	56781	>10800	58856	58856.0	10.5	-3.65
16	48467	>10800	49754	49754.0	12.6	-2.66
17	42074	>10800	42525	42525.0	8.4	-1.07
18	40153	>10800	40536	40536.0	9.4	-0.95
19	53779	>10800	55346	55346.0	10.5	-2.91
20	60097	>10800	61286	61286.0	9.4	-1.98

We observe that CPLEX 12.6 fails to optimally solve any of the 20 instances with $n = 50$ in three hours and the best feasible solutions reported by CPLEX are always worse than the solutions found by the HESA algorithm. Furthermore, HESA requires significantly less time than CPLEX 12.6 to reach the reported results.

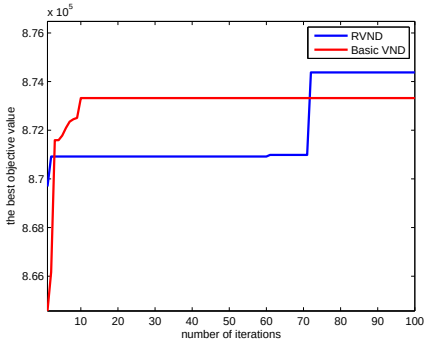
6 Discussions

We now turn our attention to an analysis of several major ingredients of the proposed algorithm: the RVND method, the neighborhoods, the adaptive selection of crossovers and the convergence ability of the proposed algorithm.

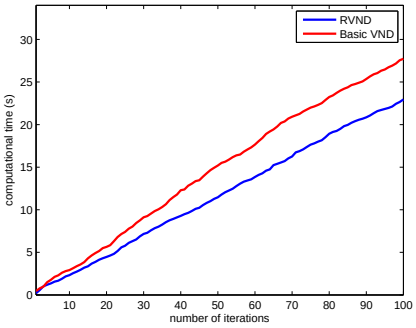
6.1 Comparison between RVND and basic VND

As described in Section 4.3, the VNS algorithm employs the RVND method as its local optimization procedure. The proposed RVND method is an extension of the basic VND method described in [23], which works in the following way. The neighborhood structures are firstly ordered in a list and then are examined one by one following the established order. As soon as an improvement of the incumbent solution in the current neighborhood occurs, the basic VND procedure resumes the search in the first neighborhood of the new incumbent solution. The whole process is stopped if the search process reaches the last neighborhood and no improving solution can be found in the last neighborhood.

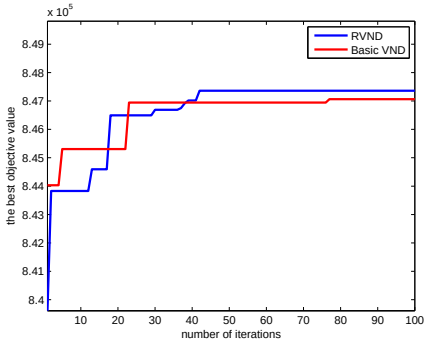
Since the RVND method is an extension of the basic VND method, we car-



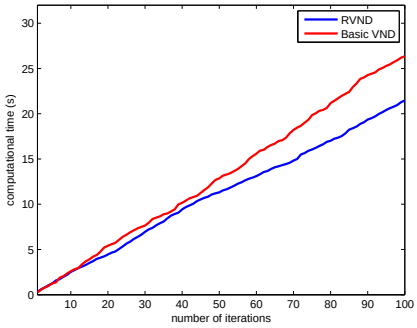
(a) Evolution of best objective value for 200.1



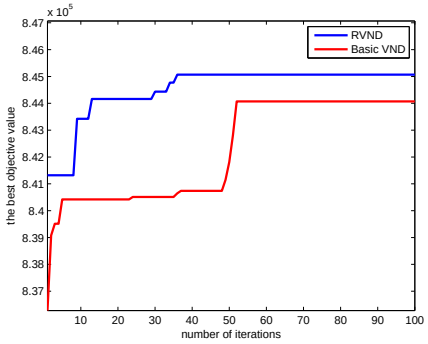
(b) Evolution of run time for 200.1



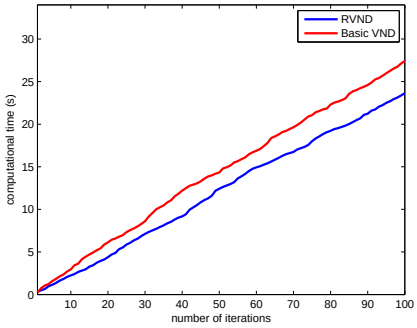
(c) Evolution of best objective value for 200.5



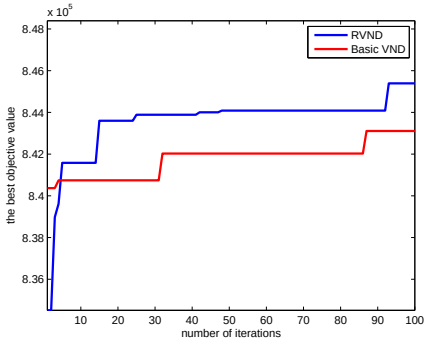
(d) Evolution of run time for 200.5



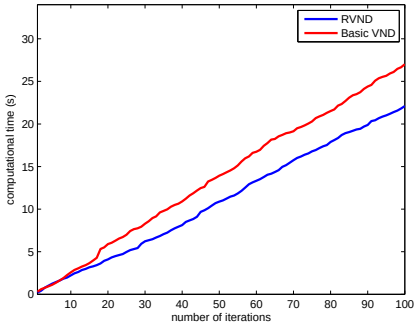
(e) Evolution of best objective value for 200.10



(f) Evolution of run time for 200.10



(g) Evolution of best objective value for 200.15



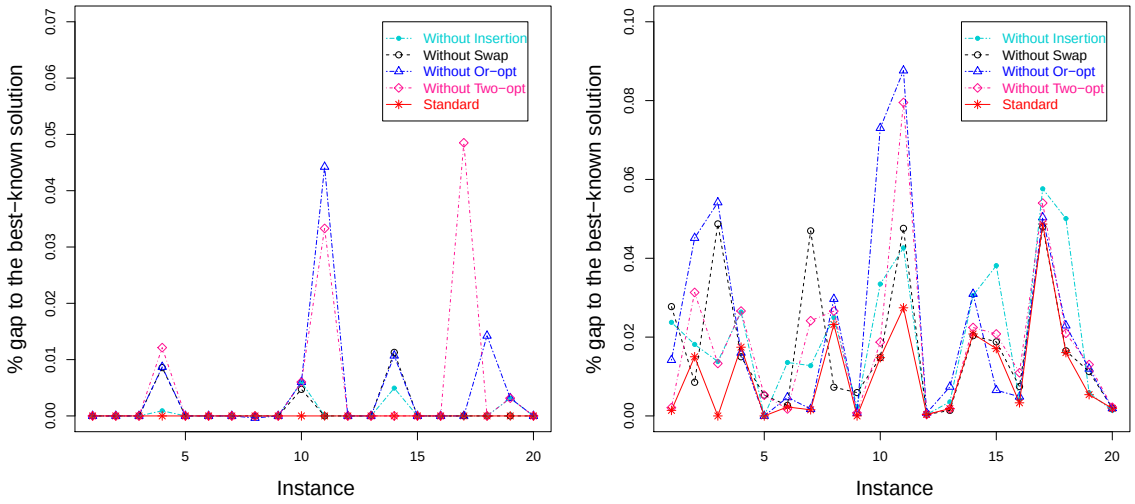
(h) Evolution of run time for 200.15

Fig. 4. Influence of the RVND and basic VND for the VNS algorithm.

ried out an experiment to compare both methods on four selected instances (*200.1*, *200.5*, *200.10* and *200.15*) with $n = 200$, under the same experimental condition as specified in Section 5.1. In this experiment, both RVND and VND were respectively employed as a local optimization procedure in the VNS algorithm, and the order of neighborhoods (i.e., $N_1 \rightarrow N_2 \rightarrow N_3 \rightarrow N_4$) is applied in the basic VND method. Specifically, for each instance, from a same initial solution generated by the construction procedure presented in Section 4.2, both methods were iteratively performed until a pre-fixed maximum number of iterations (set to 100) is reached. The evolutions of the best objective value and the run time of the iterations are plotted in Figure 4.

From Figure 4, one observes that the VNS algorithm with the RVND method performs generally better than that with the basic VND method in terms of the solution quality. Moreover, the run time increases linearly with the increase of the number of iterations, and the RVND method is faster than the basic VND method. These outcomes demonstrate the interest of the RVND method compared to the basic VND method.

6.2 Observations on neighborhood effectiveness



(a) Best results (b) Average results

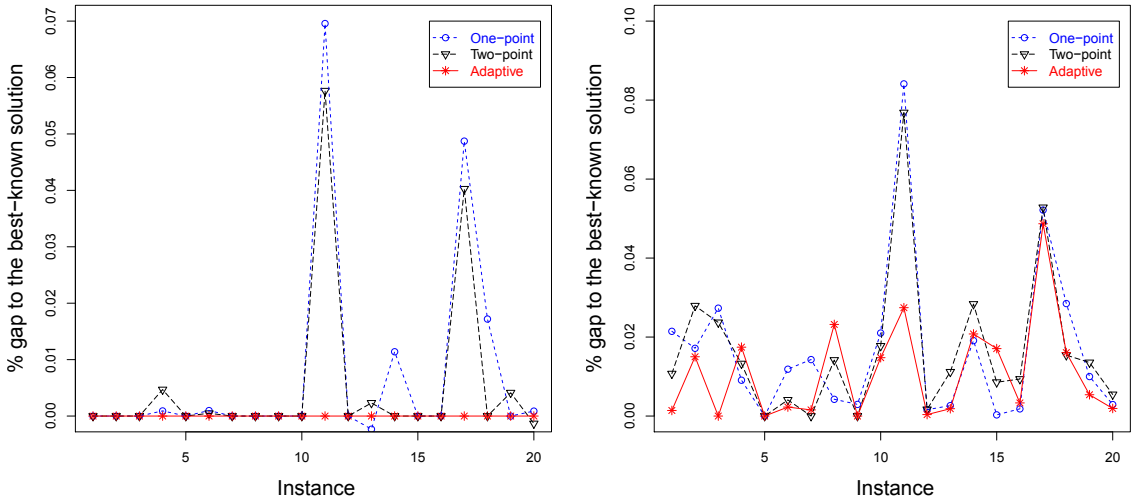
Fig. 5. Comparisons of HESA with its variants that exclude different neighborhoods.

As described in Section 4.3.2, our RVND procedure employs four dedicated neighborhood structures which are induced by the Insertion, Swap, Or-opt and Two-opt operators. In this section, we investigate the influence of each neighborhood over the performance of the HESA algorithm. For this purpose, we propose four weakened versions of HESA such that for each HESA variant,

we disable one particular neighborhood while keeping the other components unchanged. Along with the standard HESA algorithm, five HESA versions are tested on the set of instances with 200 nodes, under the same experimental condition as specified in Section 5.1. The best and the average performances are plotted in Figure 5, where the y-axis indicates the percentage gap to the updated best-known solution (see Table 5).

Figure 5 shows removing a neighborhood deteriorates the search power of HESA. This experiment confirms that all four neighborhoods contribute to the performance of the HESA algorithm. Apart from the usefulness of each individual neighborhood, their combined use within the HESA algorithm constitutes an important feature to ensure the performance of the algorithm.

6.3 Impact of the joint use of two crossover operators



(a) Best results (b) Average results

Fig. 6. Comparisons of HESA with its variants that use a single crossover operator.

As described in Section 4.4, HESA jointly applies two crossover operators by means of a probabilistic selection mechanism. To investigate the merit of this crossover combination, we compare HESA with two weakened versions of the algorithm that only use the one-point crossover and the two-point crossover respectively. For this analysis, we repeat the same experimental procedure as in the previous section. The best and the average performances are plotted in Figure 6.

In terms of the best and the average results, we observe a better performance of HESA when the two crossover operators are used, which highlights the

contribution of the combined use of the two crossover operator for HESA.

6.4 Convergence ability of the proposed algorithm

In this section, we present an experiment to assess the convergence ability of the proposed HESA algorithm. In this experiment, we ran HESA on four selected instances (*200.1*, *200.5*, *200.10* and *200.15*) with $n = 200$, under the same experimental condition as specified in Section 5.1. The stopping criterion is the number of generations (i.e., the number of applications of the recombination operation) which is set to 100.

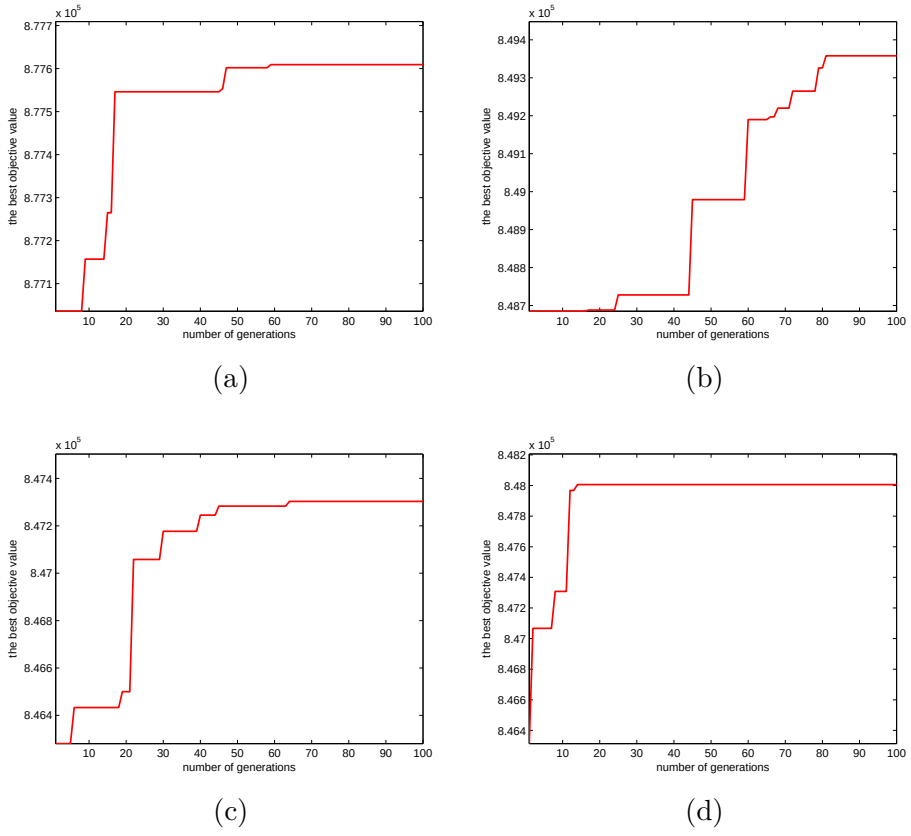


Fig. 7. Convergence graphs of HESA on four instances.

We use the running profile (convergence graph) to observe the evolution of the best objective value during the search process. Figure 7 shows the convergence graphs of HESA for the studied instances. We observe that the algorithm quickly makes significant improvements at the beginning of the search and converges to its best solution after ten to several tens of generations according to the instances considered.

7 Conclusion

The Traveling Repairman Problem with Profits is an NP-hard problem derived from several practical situations. In this paper, we presented the first population-based hybrid evolutionary search algorithm to tackle the problem. Extensive computational results conducted on six sets of 120 benchmark instances reveal that our algorithm competes very favorably with the existing heuristics from the literature. In particular, it reports 39 new lower bounds out of the 80 benchmark instances with unknown optima, while matching the best-known results of the literature for the remaining instances.

For future work, it would be interesting to investigate other crossover operators by considering more problem specific knowledge. Similarly, additional neighborhood operators could be investigated to reinforce the local optimization procedure. In addition, to efficiently examine the neighborhoods, it would be interesting to develop a fast neighborhood evaluation technique to accelerate the local search procedure. Finally, the proposed algorithm embodies rather general strategies that could be of interest for other routing problems with time-dependent profits (e.g., the Multiple Traveling Repairman Problem with Profits).

Acknowledgment

We are grateful to the anonymous referees for valuable suggestions and comments which helped us improve the paper. We thank Dr. Mustafa Avci, the first author of reference [1], for providing us with the benchmark instances tested in this work. This work is partially supported by the National Natural Science Foundation Program of China [Grant No. 71401059, 71620107002, 71531009] and the Huazhong University of Science and Technology (5001300001).

References

[1] Avci, M., & Avci, M. G. (2017). A GRASP with iterated local search for the traveling repairman problem with profits. Computers & Industrial Engineering, 113, 323-332.

[2] Bruni, M.E., Beraldi, P., & Khodaparasti, S. (2018). A heuristic Approach for the k-Traveling Repairman Problem with Profits under Uncertainty. Electronic Notes in Discrete Mathematics, 69, 221-228.

[3] Bang, B. H., & Nghia, N. D. (2010). Improved genetic algorithm for minimum latency problem. In Proceedings of the 2010 Symposium on Information and Communication Technology (pp. 9-15). ACM.

[4] Birattari, M., Yuan, Z., Balaprakash, P., & Stützle, T. (2010). F-Race and iterated F-Race: An overview. In Experimental methods for the analysis of optimization algorithms (pp. 311-336). Springer Berlin Heidelberg.

[5] Blum, A., Chalasani, P., Coppersmith, D., Pulleyblank, B., Raghavan, P., & Sudan, M. (1994). The minimum latency problem. In Proceedings of the 26th Annual ACM Symposium on Theory of Computing (pp. 163-171). ACM.

[6] Ban, H. B., Nguyen, K., Ngo, M. C., & Nguyen, D. N. (2013). An efficient exact algorithm for Minimum Latency Problem. J. PI, 1(10), 1-8.

[7] Beraldi, P., Bruni, M. E., Laganà, D., & Musmanno, R. (2018). The risk-averse traveling repairman problem with profits. Soft Computing, 1-15.

[8] Bruni, M. E., Beraldi, P., & Khodaparasti, S. (2018). A fast heuristic for routing in post-disaster humanitarian relief logistics. Transportation research procedia, 30, 304-313.

[9] Bruni, M. E., Brusco, L., Ielpa, G., & Beraldi, P. (2019). The Risk-averse Profitable Tour Problem. In Proceedings of International Conference on Operations Research and Enterprise Systems, 2019.

[10] Campos, V., Martí, R., Sánchez-Oro, J., & Duarte, A. (2014). GRASP with path relinking for the orienteering problem. Journal of the Operational Research Society, 65(12), 1800-1813.

[11] Chen, Y., Hao, J. K., & Glover, F. (2016). A hybrid metaheuristic approach for the capacitated arc routing problem. European Journal of Operational Research, 253(1), 25-39.

[12] Coene, S., & Spieksma, F. C. (2008). Profit-based latency problems on the line. Operations Research Letters, 36(3), 333-337.

[13] Carrabs, F., Cordeau, J. F., & Laporte, G. (2007). Variable neighborhood search for the pickup and delivery traveling salesman problem with LIFO loading. INFORMS Journal on Computing, 19(4), 618-632.

[14] Dewilde, T., Cattrysse, D., Coene, S., Spieksma, F. C., & Vansteenwegen, P. (2013). Heuristics for the traveling repairman problem with profits. Computers & Operations Research, 40(7), 1700-1707.

[15] Dang, D. C., Guibadj, R. N., & Moukrim, A. (2013). An effective PSO-inspired algorithm for the team orienteering problem. European Journal of Operational Research, 229(2), 332-344.

[16] Ekici, A., Retharekar, A. (2013). Multiple agents maximum collection problem with time dependent rewards. Computers & Industrial Engineering, 64(4), 1009-1018.

[17] Erkut, E., & Zhang, J. (1996). The maximum collection problem with time-dependent rewards. Naval Research Logistics, 43(5), 749-763.

[18] Erdoğan, G., & Laporte, G. (2013). The orienteering problem with variable profits. Networks, 61(2), 104-116.

[19] Fischetti, M., Gonzalez, J. J. S., & Toth, P. (1998). Solving the orienteering problem through branch-and-cut. INFORMS Journal on Computing, 10(2), 133-148.

[20] Fakcharoenphol, J., Harrelson, C., & Rao, S. (2007). The k-traveling repairmen problem. ACM Transactions on Algorithms (TALG), 3(4), 40.

[21] Feillet, D., Dejax, P., & Gendreau, M. (2005). Traveling salesman problems with profits. Transportation science, 39(2), 188-205.

[22] Goemans, M., & Kleinberg, J. (1998). An improved approximation ratio for the minimum latency problem. Mathematical Programming, 82(1-2), 111-124.

[23] Hansen, P., Mladenović, N., Todosijević, R., & Hanafi, S. (2017). Variable neighborhood search: basics and variants. EURO Journal on Computational Optimization, 5(3), 423-454.

[24] Hao, J. K. (2012). Memetic algorithms in discrete optimization. In Neri et al. (Eds.), Handbook of Memetic Algorithms (pp. 73-94). Springer Berlin Heidelberg.

[25] Hernández-Pérez, H., Rodríguez-Martín, I., & Salazar-González, J. J. (2016). A hybrid heuristic approach for the multi-commodity pickup-and-delivery traveling salesman problem. European Journal of Operational Research, 251(1), 44-52.

[26] Jarboui, B., Derbel, H., Hanafi, S., & Mladenović, N. (2013). Variable neighborhood search for location routing. Computers & Operations Research, 40(1), 47-57.

[27] Kellegöz, T., Toklu, B., & Wilson, J. (2008). Comparing efficiencies of genetic crossover operators for one machine total weighted tardiness problem. Applied Mathematics and Computation, 199(2), 590-598.

[28] Kuo, Y., & Wang, C. C. (2012). A variable neighborhood search for the multi-depot vehicle routing problem with loading cost. Expert Systems with Applications, 39(8), 6949-6954.

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

[30] Lin, S., & Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. Operations Research, 21(2), 498-516.

[31] Lu, Y., Benlic, U., & Wu, Q. (2018). A memetic algorithm for the Orienteering Problem with Mandatory Visits and Exclusionary Constraints. European Journal of Operational Research, 268(1), 54-69.

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

[33] Martí, R., Duarte, A., & Laguna, M. (2009). Advanced scatter search for the max-cut problem. INFORMS Journal on Computing, 21(1), 26-38.

[34] Mladenović, N., & Hansen, P. (1997). Variable neighborhood search. Computers & Operations Research, 24(11), 1097-1100.

[35] Mladenović, N., Urošević, D., & Ilić, A. (2012). A general variable neighborhood search for the one-commodity pickup-and-delivery travelling salesman problem. European Journal of Operational Research, 220(1), 270-285.

[36] Moscato, P. (1999). Memetic algorithms: A short introduction. In New Ideas in Optimization (pp. 219-234). McGraw-Hill Ltd., UK.

[37] Neri, F., Cotta, C. & Moscato, P. (2012). Handbook of Memetic Algorithms. Springer Berlin Heidelberg.

[38] Or, I. (1976). Traveling salesman-type combinational problems and their relation to the logistics of blood banking (PhD thesis). USA: Northwestern University.

[39] Prins, C., & Bouchenoua, S. (2005). A memetic algorithm solving the VRP, the CARP and general routing problems with nodes, edges and arcs. Recent Advances in Memetic Algorithms. Springer Berlin Heidelberg.

[40] Salehipour, A., Sörensen, K., Goos, P., & Bräysy, O. (2011). Efficient GRASP+VND and GRASP+VNS metaheuristics for the traveling repairman problem. 4OR, 9(2), 189-209.

[41] Silva, M. M., Subramanian, A., Vidal, T., & Ochi, L. S. (2012). A simple and effective metaheuristic for the minimum latency problem. European Journal of Operational Research, 221(3), 513-520.

[42] Vansteenwegen, P., Souffriau, W., & Van Oudheusden, D. (2011). The orienteering problem: A survey. European Journal of Operational Research, 209(1), 1-10.

[43] Wu, B. Y. (2000). Polynomial time algorithms for some minimum latency problems. Information Processing Letters, 75(5), 225-229.