Expert Systems With Applications 202 (2022) 117072

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Intensification-driven local search for the traveling repairman problem with profits

Jintong Ren [a,b], Jin-Kao Hao [c], Feng Wu [b], Zhang-Hua Fu [a,d,*]

[a] The Chinese University of Hong Kong, Shenzhen, 518172 Shenzhen, China
[b] University of Science and Technology of China, 230026 Hefei, China
[c] LERIA, Université d'Angers, 2 boulevard Lavoisier, 49045 Angers, France
[d] Shenzhen Institute of Artificial Intelligence and Robotics for Society, 518172 Shenzhen, China

ARTICLE INFO

Keywords:
Combinatorial optimization
Heuristics
Intensification-driven local search
Variable neighborhood search
K-exchange sampling based neighborhood
Traveling repairman problem with profits

ABSTRACT

The Traveling Repairman Problem with Profits is to select a subset of nodes (customers) in a weighted graph to maximize the collected time-dependent revenues. We introduce an intensification-driven local search algorithm for solving this challenging problem. The key feature of the algorithm is an intensification mechanism that intensively investigates bounded areas around each very-high-quality local optimum encountered. As for its underlying local optimization, the algorithm employs an extended variable neighborhood search procedure which adopts for the first time a K-exchange sampling based neighborhood and a concise perturbation procedure to obtain high-quality solutions. Experimental results on 140 benchmark instances show a high performance of the algorithm by reporting 36 improved best-known results (new lower bounds) and equal best-known results for 95 instances. Additional experiments are conducted to investigate the usefulness of the key components of the algorithm.

1. Introduction

Problem statement. The traveling repairman problem (TRP) (Blum, Chalasani, Coppersmith, Pulleyblank, Raghavan, & Sudan, 1994) is a popular combinatorial optimization problem, which is known to be $\mathcal{NP}$-hard in Afrati, Cosmadakis, Papadimitriou, Papageorgiou, and Papakostantinou (1986). Generally, the problem can be defined as follows. Given a complete weighted graph $G(V,E)$, V represents the vertex set and E is the edge set. The vertex set V is partitioned into $V = \{0\} \cup V_c$ where 0 is the depot and $V_c = \{1,2,\dots,n\}$ represents the set of n customers. Each edge $(i,j) \in E = \{(i,j) : i,j \in V, i \neq j\}$ is associated with a symmetric weight $d_{i,j} = d_{j,i}$ representing the travel time (or distance in the Euclidean plane) between the two vertices. The objective of the TRP is to find a Hamiltonian path such that the total waiting time $\sum_{i=0}^{n} l(i)$ is minimal, where $l(i)$ is the waiting time of customer i with $l(0)$ being set to 0.

The traveling repairman problem with profits (TRPP) (Dewilde, Cattrysse, Coene, Spieksma, & Vansteenwegen, 2013) generalizes the TRP by adding a non-negative profit p_i to each vertex i. A repairman starts his travel from vertex 0 (depot) and collects a revenue $p_i - l(i)$ from each visited vertex (customer). The TRPP distinguishes itself from the TRP by selecting a subset of customers to visit, which means that it is unnecessary to visit all customers and the trip stops when no positive revenue can be further collected. The objective of the TRPP is to find the open Hamiltonian circuit to maximize the total collected revenue. Formally, for a given solution $\varphi = \{x_0, x_1, x_2, \dots, x_m\}$ ($x_0 = 0$ and $x_i \in V_c$, $i = 1,2,..,m$), the objective function value is given by:

$$f(\varphi) = \sum_{i=0}^{m} \left[p_{x_i} - l(x_i)\right]^+. \tag{1}$$

where m is the number of visited customers and the revenue collected for each visited customer $\left[p_{x_i} - l(x_i)\right]^+$ is obtained as follows.

$$\left[p_{x_i} - l(x_i)\right]^+ = \begin{cases} p_{x_i} - l(x_i), & \text{if } p_{x_i} - l(x_i) \geq 0, \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Eq. (1) can be reformulated into another form if the collected revenue $p_{x_i} - l(x_i)$ is non-negative for all the customers in φ:

$$f(\varphi) = \sum_{i=0}^{m} p_{x_i} - \sum_{i=0}^{m-1} (m-i) \cdot d_{x_i,x_{i+1}}. \tag{3}$$

* Corresponding author at: The Chinese University of Hong Kong, Shenzhen, 518172 Shenzhen, China.
E-mail addresses: renjintong@cuhk.edu.cn (J. Ren), jin-kao.hao@univ-angers.fr (J.-K. Hao), wufeng02@ustc.edu.cn (F. Wu), fuzhanghua@cuhk.edu.cn (Z.-H. Fu).

https://doi.org/10.1016/j.eswa.2022.117072
Received 27 August 2021; Received in revised form 7 January 2022; Accepted 28 March 2022
Available online 21 April 2022
0957-4174/

Eq. (3) is significant to perform fast evaluations during the search process (Pei, Mladenović, Urošević, Brimberg, & Liu, 2020), which is also adopted in this study.

The TRPP can be reduced to the TRP by setting the profit of each vertex to an extremely large value, and was shown to be $\mathcal{NP}$-hard (Dewilde et al., 2013). As indicated in the literature (Avci & Avci, 2017; Dewilde et al., 2013; Lu, Hao, & Wu, 2019), the TRPP model has relevant applications in relief efforts management such as humanitarian and emergency relief logistics. For example, after an earthquake, assuming that p_i persons are in danger for each village i, a person will die at each time moment. A rescue team starts from its base and visit the damaged villages to save lives. Consequently, the goal of the rescue team is to save as many lives as possible $\sum_i [p_i - l(i)]^+$, where $l(i)$ is the arriving time of the rescue team for village i.

Literature review. In 2013, Dewilde et al. (2013) introduced a mixed 0/1 linear programming model of the TRPP. They also proposed a tabu search (TS) algorithm with multiple neighborhoods (e.g., *remove-insert, move-down, move-up, swap, 2-opt, or-opt*...) as well as a greedy initialization procedure. Six sets of 120 benchmark instances with $n = 10, 20, 50, 100, 200, 500$ were created based on various graphs of TSPLIB.[1] The TS algorithm was shown to be able to find high-quality solutions within a short time even for large instances. They also reported optimal values for small instances with $n = 10, 20$ by solving the 0/1 linear program with CPLEX.

In 2017, Avci and Avci (2017) introduced a greedy randomized adaptive search procedure with iterated local search (GRASP-ILS). In addition to its greedy randomized solution construction procedure, the proposed algorithm is characterized by its ILS procedure which combines a tabu-enhanced variable neighborhood descent algorithm with an adaptive perturbation mechanism. This algorithm improved 46 best results reported by Dewilde et al. (2013) and matched the best-known results for the remaining instances.

Later in 2019, the same authors (Avci & Avci, 2019) proposed an adaptive large neighborhood search algorithm (ALNS) for the related multiple traveling repairman problem with profits (MTRPP) and the TRPP. ALNS consists of a couple of problem-specific destroy operators and two new randomized repair operators. Tested on the benchmark instances of the TRPP, ALNS updated 36 previous best-known results.

In 2019, Lu et al. (2019) presented a population-based hybrid evolutionary search algorithm (HESA) for solving the TRPP. The algorithm employs a randomized greedy construction method to create initial solutions, two crossover operators to generate new solutions and a dedicated variable neighborhood search to improve each new solution. Computational results on six sets of 120 instances showed that this HESA was able to improve the best-known results for 39 instances and match the best-known results for the remaining instances.

In 2020, a general variable neighborhood search approach for solving the TRPP (GVNS-TRPP) was introduced in Pei et al. (2020). This algorithm integrates different neighborhoods (*Insertion*, *2-opt*, *Swap*, *Add*, *Drop*...) and auxiliary data structures to improve the efficiency of the search. They studied six different variants of the deterministic variable neighborhood descent (VND) applied to these neighborhoods according to six specific orders as well as a VND variant where the neighborhoods are applied at random (VND-R). They tested their GVNS-TRPP algorithm with VND-R on 120 instances and improved 40 best-known results. To further assess the algorithm, they also reported computational results on a new set of 20 large instances with $n = 1000$. According to the reported computational result, GVNS-TRPP can be considered to represent the state-of-the-art for solving the TRPP. As a result, this algorithm is used as the main reference algorithm in this study.

Contributions. This study aims to enrich the toolbox of practical solution methods for the TRPP and introduces an intensification-driven local search algorithm. The contributions are summarized as follows.

In terms of algorithm design, the proposed intensification-driven local search for the TRPP (IDLS-TRPP) integrates an original mechanism that examines bounded areas around each very-high-quality local optimum discovered by the underlying local optimization procedure. This mechanism uses the elite local optimum as the search center from which local optimization is repetitively launched to explore the surrounding areas to locate other high-quality local optima. The underlying local optimization procedure extends the variable neighborhood search by introducing for the first time a K-exchange sampling based neighborhood and combining it with a random exploration of four other known neighborhoods (*Swap, Insert, 2-opt and Or-opt*). IDLS-TRPP additionally adopts the first neighborhood reduction technique (using candidate sets) and integrates known streamlining techniques to ensure an efficient neighborhood evaluation.

Intensive computational evaluations on the 140 TRPP benchmark instances in the literature demonstrate a remarkable performance of the proposed algorithm. It discovers new best-known solutions (improved lower bounds) for 36 large instances and matches the best-known results for 95 other instances.

Outline. The remainder of this paper is organized as follows. Section 2 introduces the general scheme of the proposed algorithm, the greedy initialization procedure, the extended variable neighborhood search procedure as well as the concise perturbation phase. Section 3 presents computational results and comparisons with the literature. Section 4 experimentally investigates the influences of the key components of IDLS-TRPP over the performance of the algorithm. Section 5 draws conclusions and provides perspectives.

2. An intensification-driven local search for the TRPP

2.1. General scheme

The IDLS-TRPP algorithm is inspired by the Distance Guided Local Search (DGLS) framework (Porumbel & Hao, 2020), which provides an effective way to enhance the intensification capacity of an underlying local search procedure. The basic idea of DGLS is to perform intensified exploration around each very-high-quality local optimum (elite solution) φ_e in a systematic way to find other still better solutions. This is achieved by launching repetitively the underlying local search procedure starting from φ_e and each local search runs within a sphere of radius R. As such, unlike a conventional local search whose search trajectory is a continuous search path, a DGLS search trajectory is a tree-like structure, reducing thus the possibility for the search process to miss nearby high-quality solutions, which may happen with the conventional local search (Porumbel & Hao, 2020).

Based on the above idea of DGLS, the proposed algorithm for the TRPP adopts a simplified approach to explore the nearby solutions around elite local optima. Let φ^* be the best solution found so far, IDLS-TRPP repetitively runs from φ^* a underlying local search, which is composed of an extended variable neighborhood search phase and a concise perturbation phase. Each run of the local search repeats these two phases until a solution better than φ^* is encountered or the repetitions reach a search depth fixed by a parameter R (which mimics the radius parameter of DGLS). If the local search reaches the fixed search depth, a new local search is launched again starting from a slightly perturbed φ^*. During a local search run, if a new solution better than φ^* is found, IDLS-TRPP uses the new best solution to update φ^*, from which a new cycle of local search runs is performed to explore the nearby local optima around the newly discovered elite solution. Fig. 1 illustrates the tree-like search structure of the IDLS-TRPP algorithm.

The pseudo-code of the proposed algorithm is shown in Algorithm 1, which relies on three components: greedy initialization procedure (GreedyIniSol), extended variable neighborhood search procedure (EVNS) and concise perturbation procedure (CPerturb).

IDLS-TRPP starts by generating an initial solution φ with the GreedyIniSol procedure (line 7), constructing the candidate sets by

[1] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

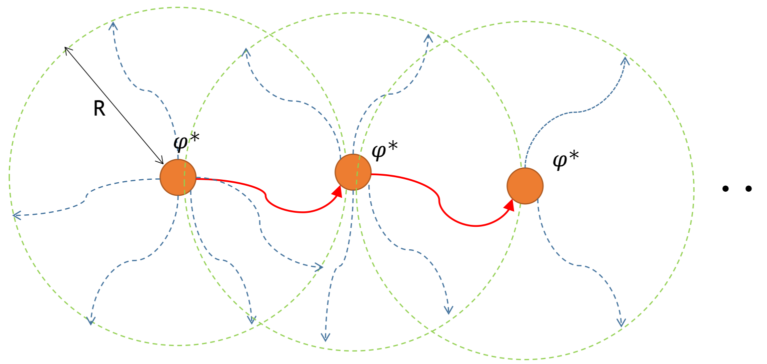


Fig. 1. Illustration of the tree-like search structure of the IDLS-TRPP algorithm. Around each new elite solution φ^*, the underlying local search procedure is repetitively run to explore nearby local optima with a search depth limited to R (blue dotted lines). Once a new improving solution is found (red lines), the best found solution φ^* is updated and a new bounded search area is created based on this new found solution (within the green dotted lines).

Algorithm 1 Intensification-driven local search for the TRPP (IDLS-TRPP)

1: **Input**: Input graph $G(V,E)$, search depth limit R, evaluation function f and cutoff-time T_{max}
2: **Output**: Best found solution φ^*
3: /* $GreedyIniSol$ is used to generate a good-quality solution. */
4: /* $IniCandidateSet$ is used to initialize the candidate sets. */
5: /* $EVNS$ is used to perform the local optimization. */
6: /* $CPerturb$ is used to modify (slightly) the input local optimum. */
7: $\varphi \leftarrow GreedyIniSol()$ // See Section 2.2
8: $IniCandidateSet()$ // See Section 2.3.1
9: $\varphi^* \leftarrow \varphi$
10: $Ct \leftarrow 0$
11: **while** T_{max} is not reached **do**
12: $\quad \varphi \leftarrow EVNS(\varphi)$ // See Section 2.3
13: $\quad$ **if** $f(\varphi) < f(\varphi^*)$ *and* $Ct < R$ **then**
14: $\quad\quad Ct \leftarrow Ct + 1$
15: $\quad\quad \varphi \leftarrow CPerturb(\varphi)$ // See Section 2.4
16: $\quad$ **else if** $f(\varphi) < f(\varphi^*)$ *and* $Ct \geq R$ **then**
17: $\quad\quad Ct \leftarrow 0$
18: $\quad\quad \varphi \leftarrow CPerturb(\varphi^*)$
19: $\quad$ **else**
20: $\quad\quad Ct \leftarrow 0$
21: $\quad\quad \varphi^* \leftarrow \varphi$
22: $\quad\quad \varphi \leftarrow CPerturb(\varphi)$
23: $\quad$ **end if**
24: **end while**
25: **return** φ^*

IniCandidateSet (line 8) and initiating the best found solution φ^* as well as the search depth counter Ct (lines 9–10). Then it enters the main loop (lines 11–24) to explore new solutions by iterating the EVNS procedure and the CPerturb procedure. For each while loop, the current solution is first improved by EVNS (Section 2.3), the CPerturb procedure is then applied either to the current or the best found solution. First, if the previous best solution φ^* is not updated and the search depth R is not reached ($Ct < R$), the counter Ct is incremented and CPerturb is operated on the current solution φ (lines 13–15). This allows EVNS to continue its trajectory from a slightly modified solution. Second, if EVNS reaches the search depth limit ($Ct \geq R$), the counter Ct is reset to 0 and CPerturb is applied to perturb the best solution φ^* (lines 16–18). This triggers a new search trajectory from φ^*. Finally, if EVNS reaches a solution φ better than the best solution φ^*, the best solution φ^* is updated, the counter Ct is reset to 0, and the perturbation is performed on the new elite solution φ^* (lines 19–23). This enables the algorithm to move definitively to the new search area centered at the newly discovered elite solution. The whole algorithm stops when the given cutoff-time T_{max} is reached and the best solution φ^* ever found is returned (line 25).

Algorithm 2 Greedy initialization procedure (GreedyIniSol)

1: **Input**: Input graph $G(V,E)$ and the maximum size of the subset q.
2: **Output**: Current solution φ.
3: /* φ is a permutation where $\varphi(k)$ denotes the customer on position k */
4: $\varphi(0) \leftarrow 0$
5: $V_r \leftarrow \{1,2,...,n\}$
6: $k \leftarrow 1$
7: **repeat**
8: $\quad V_a \leftarrow$ subset of V_r with the $min(q,n-k+1)$ customers which have the largest ratio of profit-distance with respect to the previous customer $\varphi(k-1)$
9: $\quad \varphi(k) \leftarrow$ randomly select one customer from V_a
10: $\quad V_r \leftarrow V_r \setminus \{\varphi(k)\}$
11: $\quad k \leftarrow k+1$
12: **until** All customers receive a position.
13: **return** φ

2.2. Greedy initialization procedure

In the greedy initialization procedure, the main operation is to add a customer to the current partial solution iteratively until all the customers are used to construct a complete solution φ (an array), where $\varphi(k)$ denotes the customer on position k.[2] To determine the customer for a position, we consider the profit-distance ratio of a vertex x_j with respect to another vertex x_i, given by $r_{x_i,x_j} = \frac{p_{x_j}}{d_{x_i,x_j}}$.

The pseudo-code of this procedure is presented in Algorithm 2. At first, the depot is added to the initial empty solution φ, the set of customers V_r is initialized and k is set to 1 (lines 4–6). Then the algorithm iteratively assigns a customer to each position (lines 7–12). For position k, a subset $V_a \subseteq V_r$ is generated by selecting the $min(q,n-k+1)$[3] customers with the largest profit-distance ratio with respect to the customer of the previous position $\varphi(k-1)$ (line 8). Then, a random customer from V_a is assigned to $\varphi(k)$ and removed from V_r (lines 9–10). The process is repeated until all customers are assigned a position. In our work, q (maximum size of the subset) is set to 3. The whole initialization procedure can be finished in a time complexity $O(n^2)$.

[2] The notion 'position' here represents the index in an array.
[3] $min(a,b)$ denotes the smaller value between a and b.

Algorithm 3 Extended Variable Neighborhood Search (EVNS)

```
1: Input: Evaluation function f and current solution φ
2: Output: Local best solution φ
3: /* N_1, N_2, N_3, N_4 represent Swap, Insert, 2-opt and Or-opt neighborhoods. */
4: /* N_Add, N_Drop, N_kes denote Add, Drop and K-exchange sampling neighborhoods. */
5: repeat
6:    φ_lb ← φ
7:    φ ← LocalSearch(φ, N_Add)
8:    NL ← {N_1, N_2, N_3, N_4}
9:    while NL ≠ ∅ do
10:      Randomly choose a neighborhood N ∈ NL
11:      φ ← LocalSearch(φ, N)
12:      φ ← LocalSearch(φ, N_Drop)
13:      NL ← NL \ {N}
14:   end while
15:   φ ← LocalSearch(φ, N_kes)
16:   φ ← LocalSearch(φ, N_Drop)
17: until f(φ_lb) ≥ f(φ)
18: return φ
```

2.3. Extended variable neighborhood search

The variable neighborhood search (VNS) method (Hansen & Mladenović, 2005) has been applied to a number of routing problems (Frifita, Masmoudi, & Euchi, 2017; Fu, Redi, Halim, & Jewpanya, 2020; Karakostas, Sifaleras, & Georgiadis, 2019, 2020; Mladenović, Urošević, Ilić, et al., 2012; Soylu, 2015; Xu & Cai, 2018). It has also proved to be quite successful for solving the TRPP, as illustrated in the literature (Avci & Avci, 2017, 2019; Lu et al., 2019; Pei et al., 2020). For this reason, we also adopt the VNS framework to build our underlying local optimization component and we explain the main differences between our approach and the existing approaches in Section 2.5. The proposed approach is an extended VNS procedure (EVNS) composed of two phases. The first phase uses the descent local search to explore four neighborhoods (generated by *Swap, Insert, 2-opt, Or-opt*) in a random order (See Section 2.3.2), similar to the VND-R procedure in Pei et al. (2020). The second phase employs a new K-exchange sampling based neighborhood to further improve the local optimum from the first phase (See Section 2.3.3). Both phases employ the first-improving strategy (i.e., accepting the first improving solution encountered). This is the first time that this strategy is adopted to solve the TRPP and we will assess its usefulness in Section 4.2.

The pseudo-code of EVNS is presented in Algorithm 3. At first, the current solution φ is recorded as the local best solution φ_{lb} (line 6). Then a local optimization based on the Add operator is used to add customers to the solution (line 7). The set of neighborhoods NL is initialized by N_1, N_2, N_3, N_4 which represent the $Swap$, $Insert$, 2-*opt* and *Or-opt* neighborhoods respectively (Section 2.3.2). In the inner loop (lines 9–14), these four neighborhoods are explored by the descent local search in a random order, each descent being followed by a descent with the $Drop$ neighborhood. When this local search with these four neighborhoods terminates, a local optimization based on the $K\text{-}exchange\ sampling$ neighborhood (N_{kes}) is performed, followed by a descent with the $Drop$ neighborhood (lines 15–16). This process is repeated until the local best solution φ cannot be further improved any more (line 17). At this point, the search is considered to be trapped into a deep local optimum and the concise perturbation (Section 2.4) is triggered to displace the search to a new area according to the strategy explained in Section 2.1.

2.3.1. Candidate set

To accelerate the calculation for solving the traveling salesman problem (TSP) (Flood, 1956) and the vehicle routing problem (VRP) (Dantzig & Ramser, 1959), researchers usually examine a number of most promising neighboring solutions rather than all solutions in the neighborhood. For example, the Lin–Kernighan (LK) heuristic (Lin, 1965) usually restricts the inclusion of links in the tour to the five nearest neighbors to a given vertex. This technique is realized by introducing a candidate set (candidate list) containing a limited number of candidates for a given customer. For routing problems (Bentley, 1992; Lust & Jaszkiewicz, 2010), there are several methods to construct the candidate set, such as the nearest method (Lin, 1965), the α-nearest method (Helsgaun, 2000) and the granular neighborhood method (Toth & Vigo, 2003). In this work, we employ the nearest method to generate two candidate sets S_{kes} and S_{nf}, where S_{kes} is constructed for the $K\text{-}exchange\ sampling$ neighborhood (Section 2.3.3) and S_{nf} is prepared for the other neighborhoods generated by *Swap, Insert, 2-opt, Or-opt* (Section 2.3.2). The maximum size l_{kes} and l_{nf} for the candidate sets S_{kes} and S_{nf} are determined in Section 3.2.

2.3.2. Classic neighborhoods

The six operators to generate neighborhoods were used in previous studies (Avci & Avci, 2017, 2019; Lu et al., 2019; Pei et al., 2020). However, candidate lists are also employed to generate these six neighborhoods, where $N_1 - N_4$ only change the visiting order of the selected customers and N_{Add} as well as N_{Drop} change the list of visited customers.

For a given solution $\varphi = \{x_0, x_1, \dots, x_m\}$, let m be the number of visited customers and l_{nf} represent the maximum size of the candidate set S_{nf}. These six neighborhoods are defined as follows:

(1) N_1 ($Swap$): The positions of two customers are exchanged. Exploring the $Swap$ neighborhood with respect to S_{nf} could be finished within $O(m \cdot l_{nf})$ (see below):

$$N_1(\varphi) = \{\varphi' = \varphi \oplus Swap(x_i, x_j), 0 < i \le m, 0 < j \le m, x_j \in S_{nf}(x_i)\}$$

where $\varphi' = \varphi \oplus Swap(x_i, x_j)$ denotes the solution obtained by exchanging the positions of x_i and x_j from the current solution φ.

(2) N_2 ($Insert$): A customer is removed from its position and inserted between two adjacent customers. Exploring the $Insert$ neighborhood with respect to S_{nf} requires $O(m \cdot l_{nf})$ time (Pei et al., 2020):

$$N_2(\varphi) = \{\varphi' = \varphi \oplus Insert(x_i, x_j), 0 < i \le m, 0 \le j \le m, x_j \in S_{nf}(x_i)\}$$

where $\varphi' = \varphi \oplus Insert(x_i, x_j)$ depicts the solution obtained by inserting x_i to the position between x_j and x_{j+1} from the current solution φ.

(3) N_3 (2-*opt*): Two non-adjacent edges are removed and replaced by two new edges to reconnect the circuit. Exploring the 2-*opt* neighborhood with respect to S_{nf} can be finished within $O(m \cdot l_{nf})$ (Pei et al., 2020):

$$N_3(\varphi) = \{\varphi' = \varphi \oplus 2\text{-}opt(x_i, x_j), 0 \le i < m, 0 \le j < m, |i-j| > 1, x_j \in S_{nf}(x_i)\}$$

where $\varphi' = \varphi \oplus 2\text{-}opt(x_i, x_j)$ represents the solution obtained by removing two edges ((x_i, x_{i+1}) and (x_j, x_{j+1})), as well as reconnecting two new edges ((x_i, x_j) and (x_{i+1}, x_{j+1})) from the current solution φ.

(4) N_4 (*Or-opt*): A block of h ($h = 2, 3$) consecutive customers is removed and inserted between two adjacent customers. Exploring the *Or-opt* neighborhood with respect to S_{nf} requires $O(h \cdot m \cdot l_{nf})$ time (Pei et al., 2020):

$$N_4(\varphi) = \{\varphi' = \varphi \oplus Or\text{-}opt(x_i, x_j, h), 0 < i \le m+1-h, 0 < j \le m, x_j \in S_{nf}(x_i)\}$$

where $\varphi' = \varphi \oplus Or\text{-}opt(x_i, x_j, h)$ depicts the solution obtained by inserting the sequence of $(x_i, x_{i+1}, \dots, x_{i+h-1})$ to the position between x_j and x_{j+1} from the current solution φ.

(5) N_{Add} (Add): One unselected customer is added to some position of the solution. The complexity of exploring the complete Add neighborhood is $O((n-m)\cdot m)$ (Pei et al., 2020).

(6) N_{Drop} ($Drop$): One selected customer is dropped from the solution. The complexity of exploring the complete $Drop$ neighborhood is $O(m)$ (Pei et al., 2020).

According to Pei et al. (2020), evaluating one neighboring solution of $Insert$, 2-opt, Or-opt,[4] Add and $Drop$ requires $O(1)$ time. We show here the complexity for evaluating one neighboring solution of $Swap$ neighborhood N_1 and the whole neighborhood.

Proof. Let $\varphi = \{x_0, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_{j-1}, x_j, x_{j+1}, \dots, x_m\}$ be a solution with m selected customers. Swapping x_i and x_j $(0 < i < j \le m)$ leads to a neighboring solution $\varphi' = \{x_0, \dots, x_{i-1}, x_j, x_{i+1}, \dots, x_{j-1}, x_i, x_{j+1}, \dots, x_m\}$. As the set of selected customers is not changed, we only calculate the change of the accumulated distance. By Eq. (3), the change of objective value $\Delta_f = f(\varphi') - f(\varphi)$ can be easily calculated as follows.

(1) If x_i and x_j are not adjacent, then

$$\begin{aligned}\Delta_f &= (m-i+1)\cdot(d_{x_{i-1},x_i} - d_{x_{i-1},x_j}) + (m-i)\cdot(d_{x_i,x_{i+1}} - d_{x_j,x_{i+1}})\\ &\quad + (m-j+1)\cdot(d_{x_{j-1},x_j} - d_{x_{j-1},x_i}) + (m-j)\cdot(d_{x_j,x_{j+1}} - d_{x_i,x_{j+1}})\end{aligned}$$

(2) If x_i and x_j are adjacent, then

$$\Delta_f = (m-i+1)\cdot(d_{x_{i-1},x_i} - d_{x_{i-1},x_j}) + (m-j)\cdot(d_{x_j,x_{j+1}} - d_{x_i,x_{j+1}})$$

In other words, Δ_f for any neighboring solution can be obtained in $O(1)$. As a result, the complexity of exploring the N_1 neighborhood is $O(m \cdot l_{nf})$.

Finally, there may exist several nodes in the solution whose collected revenues $p_{x_i} - l(x_i)$ are negative during the search process while Eq. (3) only considers non-negative profits. To eliminate this difficulty, we implement a local optimization based on the $Drop$ operator after applying local search with other neighborhoods (See lines 12 and 16 in Algorithm 3). It is worth noting that dropping the nodes with negative revenues always leads to a solution of better or equal quality (see Proof). This explains why applying the $Drop$ operator within the descent local search (only accepting better solutions) is able to eliminate the nodes with negative revenues efficiently.

Proof. Given a graph $G(V, E)$ in the Euclidean space, we have a feasible solution

$$\varphi = \{x_0, x_1, x_2, \dots, x_{j-1}, x_j, x_{j+1}, \dots, x_m\}$$

where the collected revenue at the node x_j is negative $(p_{x_j} - l(x_j) < 0)$. Deleting the node x_j, a new solution φ' is obtained.

$$\varphi' = \{x_0, x_1, x_2, \dots, x_{j-1}, x_{j+1}, \dots, x_m\}$$

According to Eq. (1), we get

$$f(\varphi') - f(\varphi) = \sum_{i=j+1}^{i=m} \left(\left[p_{x_i} - l(x_i) + \delta\right]^+ - \left[p_{x_i} - l(x_i)\right]^+\right) - \left[p_{x_j} - l(x_j)\right]^+ \ge 0$$

where $\left[p_{x_j} - l(x_j)\right]^+$ equals 0 (because $p_{x_j} - l(x_j) < 0$) and $\delta = w_{x_{j-1},x_j} + w_{x_j,x_{j-1}} - w_{x_{j-1},x_{j+1}}$ is non-negative due to the triangle inequality in the Euclidean space. Therefore, dropping the nodes with negative revenues leads to a solution of better or equal quality.

2.3.3. K-exchange sampling based neighborhood

This section presents a new neighborhood – the K-exchange sampling (KES) neighborhood N_{kes}, which is constructed by the solutions randomly sampled from the K-exchange neighborhood.[5] To efficiently explore N_{kes}, we also propose a corresponding KES heuristic, which is inspired by the popular LK heuristic (Lin, 1965). The main difference between KES heuristic and LK heuristic is stated as follows. With the four criterion,[6] the LK heuristic efficiently explores the complete K-exchange neighborhood to obtain the best solution in the neighborhood (K-opt). On the contrary, our KES heuristic does not target the optimality of the found solution and only samples at random a portion of the solutions from the K-exchange neighborhood.

We describe now the *KES* heuristic for exploring N_{kes}. Starting from a random node, the *KES* heuristic successively swaps pairs of edges between the nodes until an improving solution is found or the maximum number of swaps K (K is a parameter) is reached. This procedure is called one 'simulation'. The *KES* heuristic repeats the simulation until no improvement is reached during $m \cdot d_l$ successive simulations, where m is the number of selected customers and d_l is a parameter called 'exploration limit'. In our work, the nodes for an edge swap with respect to customer x_i are restricted to the candidate set $S_{kes}(x_i)$. The parameters d_l and K are determined in Section 3.2.

Fig. 2 illustrates the process of the *KES* heuristic on a 20-customer graph, with an initial solution φ_a shown in Fig. 2(a). The maximum number of swaps K is set to 3 and the parameter d_l is set to 5. A simulation starts by selecting a random node from the solution (node x_4 marked in blue in Fig. 2(a)) and the connection between x_4 and x_5 is broken. Here, we call the node to seek for new connection as the target node (x_5). The *KES* heuristic repeats the following four steps.

(1) 'Identify': We identify the candidate set of the target node. In Fig. 2(a), the candidate set for the target node x_5 is given by $S_{kes}(x_5) = \{x_8, x_{11}, x_{16}\}$ marked in orange (the candidate set is determined by the input graph).

(2) 'Choose': We randomly choose a node from the candidate set of the target node. In this example, we choose x_8.

(3) 'Swap': We swap the edges between the two pair of nodes. From Fig. 2(a) to Fig. 2(b), we break the connection between x_7 and x_8 and reconnect x_5 and x_8 as well as x_4 to x_7 to get a new solution φ_b.

(4) 'Evaluate': We evaluate the new solution to determine whether we continue this simulation. If φ_b is better than φ_a in terms of the objective value, φ_b replaces solution φ_a, this simulation is ended, and a new simulation is started with the newly obtained solution. Otherwise, we repeat the above procedure using the new target node x_7 based on the intermediate solution in Fig. 2(c).

Following the same rule, we reconnect x_7 and x_{13} (one candidate node of x_7 marked in orange in Fig. 2(c)) to get a temporary solution in Fig. 2(d). We repeat the same procedure and reconnect x_{12} and x_{18} to get the solution in Fig. 2(e). Here we performed three edge swaps and reached the maximum number K. Hence we reconnect x_4 and x_{17} and finish this simulation.

Following the step of 'Evaluate', if the solution φ_f in Fig. 2(f) is better than the original solution φ_a, it is recorded and a new simulation is stimulated based on the solution φ_f. Otherwise, we restart a new simulation from the original solution φ_a. The maximum number of simulations is equal to $m \cdot d_l = 20 \cdot 5 = 100$ where m represents the number of selected customers. The *KES* heuristic stops when there is no improvement over 100 successive simulations.

More explanations about the differences between the LK heuristic and the *KES* heuristic are given in Section 2.5.

[4] $Inter$-$Swap$ denotes the operation by exchanging a selected customer with an unselected customer. However, Pei et al. (2020) named it as $Swap$ in their work. Here, we call it $Inter$-$Swap$ to distinguish itself from $Swap$ in our work.

[5] The K-exchange neighborhood consists of the solutions by replacing at most K edges from the current solution.

[6] They are the sequential exchange criterion, the feasibility criterion, the positive gain criterion and the disjunctivity criterion (Helsgaun, 2000).

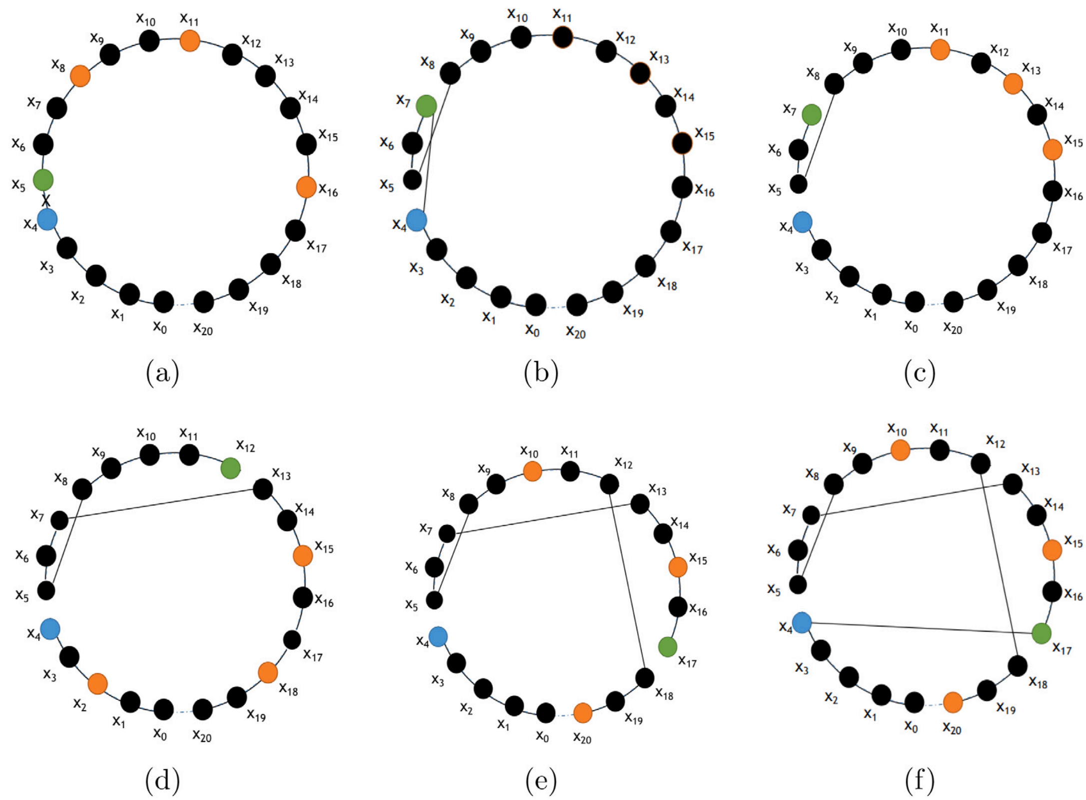


Fig. 2. Illustration of the *KES* heuristic on a 20-customer graph.

2.4. Concise perturbation

After EVNS, a concise perturbation phase is used to help the search escape from the deep local optimum. As explained in Section 2.1, the perturbation operates either on the current solution φ or the best found solution φ^* according to the dedicated rule. To perform the perturbation, we apply $Insert$ and Add to transform the chosen solution. We firstly execute the $Insert$ operation p_1 times by randomly choosing a customer x_r from the set of the visited customers and inserting it to a random position. Then we apply the Add operator $min(p_2, |V| - |V_s|)$ times by adding at each time an unselected customer $x_i \in V \setminus V_s$ to the position behind a random vertex $x_j \in V_s \cap S_{nf}(x_i)$ where V_s is the set of selected customers. p_1 and p_2 are two parameters determined in Section 3.2. We also experimented other perturbation operations, but this concise perturbation method proves to be the most suitable.

2.5. Novelties with respect to the existing algorithms

We discuss now the novelties of the proposed IDLS-TRPP algorithm with respect to the existing TRPP methods.

First, the IDLS-TRPP algorithm uses the intensification mechanism introduced in Section 2.1 to ensure an intensified exploration of every elite solution encountered during the search. This mechanism uses the latest best solution as the search center and explores multiple search directions by repetitively launching the underlying EVNS procedure from this center. This strategy enables IDLS-TRPP to find additional high-quality solutions that may be missed by conventional local search methods.

Second, like the algorithms (Avci & Avci, 2017, 2019; Dewilde et al., 2013; Lu et al., 2019; Pei et al., 2020) for solving the TRPP, our algorithm also relies on the VNS framework to perform the local optimization. The employment of the candidate lists helps our algorithm to explore the neighborhoods more efficiently compared to the main reference algorithm (Pei et al., 2020). The detailed comparisons are listed in Table 1. Besides that, our EVNS procedure enhances the exploration of four known neighborhoods (*Swap, Insert, 2-opt and Or-opt*) by a K-exchange sampling based neighborhood N_{kes}, which was never applied in the literature for solving the TRPP. This generally allows the algorithm to find still better solutions from the best local optimum generated by the other neighborhoods.

Table 1
Summary of the classical neighborhood structures as well as their complexities in Pei et al. (2020) and the proposed algorithm, where n depicts the number of customers, m is the number of selected customers, h is the number of consecutive customers in the block for *Or-opt*, and l_{nf} is the maximum size of the candidate set S_{nf}.

Neighborhood	GVNS-TRPP (Pei et al., 2020)		IDLS-TRPP	
	Employment	Complexity	Employment	Complexity
$Swap$	×	–	✓	$O(m \cdot l_{nf})$
$Insert$	✓	$O(m^2)$	✓	$O(m \cdot l_{nf})$
2-opt	✓	$O(m^2)$	✓	$O(m \cdot l_{nf})$
Or-opt	✓	$O(h \cdot m^2)$	✓	$O(h \cdot m \cdot l_{nf})$
$Inter\text{-}Swap$	✓	$O((n-m) \cdot m)$	×	–
Add	✓	$O((n-m) \cdot m)$	✓	$O((n-m) \cdot m)$
$Drop$	✓	$O(m)$	✓	$O(m)$

It is worth mentioning that, we employ the newly introduced *KES* heuristic instead of the LK heuristic to explore the K-exchange neighborhood to avoid the high computational complexity of the LK heuristic. Indeed, effective fast evaluation techniques applied in the TSP are not applicable due to the potential negative profit nodes for the TRPP. Hence, the LK heuristic has a high computational cost. On the contrary, the *KES* heuristic is computationally advantageous since it only samples partially the K-exchange neighborhood.

Finally, the TRPP algorithms in the literature explore each neighborhood completely. By contrast, our algorithm utilizes the candidate set strategy to reduce each neighborhood, which consequently increases the computational efficiency of the algorithm.

Table 2
Description and ranges of the parameters of IDLS-TRPP used for automatic parameter tuning with Irace (nez, Dubois-Lacoste, Pérez Cáceres, Birattari, & Stützle, 2016).

Parameter	Description	Type	Value range
l_{nf}	Maximum size of the candidate set S_{nf}	Categorical	{10, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 100, 200, 250}
l_{kes}	Maximum size of the candidate set S_{kes}	Categorical	{3, 5, 6, 10, 15, 20, 30, 35, 40, 45, 50, 60, 70}
K	Maximum number of switching in N_{kes}	Categorical	{3, 5, 10, 15, 20, 30, 35, 40, 45, 50, 60, 70, 100}
R	Radius of the bounded area	Integer	[0, 30]
p_1	Strength of the $Insert$ perturbation	Integer	[0, 30]
p_2	Strength of the Add perturbation	Integer	[0, 30]
d_l	Exploration limit of N_{kes}	Real	[0.0, 15.0]

As we demonstrate in Section 3, the IDLS-TRPP algorithm integrating these features as well as the fast neighborhood evaluation techniques from (Pei et al., 2020) bypasses existing methods on the popular benchmark instances. In Section 4, we further verify experimentally the effectiveness of the new features of the proposed algorithm.

3. Experimental results

This section aims to assess the performance of the proposed algorithm. For this purpose, we perform computational experiments over the benchmark instances in the literature and present comparisons with the best TRPP algorithm.

3.1. Experimental setup

Seven sets of 140 benchmark instances available in the literature are used, which include different numbers of customers (n=10, 20, 50, 100, 200, 500 and 1000). Each set contains 20 instances.[7] The first six sets were firstly introduced by Dewilde et al. (2013) based on graphs from TSPLIB, and the last set (with 1000 customers) was proposed by Pei et al. (2020).

IDLS-TRPP was coded in the C++ programming language and compiled with the g++ 7.5.0 compiler and the optimization flag -O3.[8] The experiments were performed on a computer with a 2.8 GHz AMD-opteron-4184 CPU running Linux OS. Considering the stochastic nature of the algorithm, IDLS-TRPP was independently executed 20 times on each instance with different random seeds. The cutoff-time T_{max} (in seconds) per run is set to be the number of customers in accordance with the setting in Pei et al. (2020). Given that our 2.8 GHz computer is slightly slower than the 3.2 GHz computer used in Pei et al. (2020). This stopping condition can be considered to be fair for the comparative study with respect to the main reference algorithm of Pei et al. (2020).

3.2. Tuning of parameters

We used the Irace automatic algorithm configuration package (nez et al., 2016) to determine a suitable setting for the parameters listed in Table 2, which also includes the range of values of each parameter. For this tuning experiment, the maximum number of runs (tuning budget) used by Irace is set to 2000. From the instances of large size (n=500 and 1000), we selected a subset of 10 training instances which are 500.1, 500.6, 500.12, 500.16, 500.17, 1000.1, 1000.2, 1000.5, 1000.6 and 1000.7. This experiment with Irace led to the following parameter setting: $l_{nf} = 25$, $l_{kes} = 5$, $K = 10$, $R = 2$, $p_1 = 2$, $p_2 = 1$ and $d_l = 5.9$, which was consistently used for all the experiments reported in this paper. This parameter setting can also be considered to be the default setting of the IDLS-TRPP algorithm.

[7] These instances can be download from: https://github.com/thetopjiji/TRPP.

[8] The source code will be made available on https://github.com/thetopjiji/TRPP upon the publication of this work.

3.3. Comparisons with state-of-the-art algorithms

This section presents the computational results obtained by IDLS-TRPP with respect to the reference algorithm GVNS-TRPP (Pei et al., 2020) over the 140 benchmark instances. We used GVNS-TRPP as the main reference, because the computational results reported in the literature indicate that GVNS-TRPP clearly dominates all other TRPP algorithms and holds the state-of-the-art results for the 140 instances.

Table 3 summarizes the results of IDLS-TRPP compared to GVNS-TRPP over the seven sets of instances (better results are indicates in bold). Column 'Size' describes the size of the instances in each set. Columns 'Best', 'Average' and 'Tavg' (columns 2–4) indicate respectively the best found results, average found results and average time to attain the best objective value obtained by GVNS-TRPP (averaged over the 20 instances in each set), while columns 5–7 depict the same information for our IDLS-TRPP algorithm. The last column 'imp' presents the improvement in percentage of the best objective value found by IDLS-TRPP over the best objective value of GVNS-TRPP. Note that it is not meaningful to compare the computation time of two algorithms if they do not report the same results (this is the case for several sets of instances in our case). So timing information is provided for indicative purposes only.

From Table 3, one observes that IDLS-TRPP is able to attain the best results reported in the literature for the instances of small sizes ($n = 10, 20, 50$). For the remaining four sets of instances, IDLS-TRPP achieves better results in terms of the average value of the best solutions (column 'Best'). Concerning the average results (column 'Average'), IDLS-TRPP performs better than GVNS-TRPP for the instances of large sizes ($n = 500, 1000$), while the reverse is true for the instances with $n = 100, 200$. Overall, IDLS-TRPP performs very well by updating 36 best-known solutions (only missing 9 best-known results) and matching the best-known results for 95 other instances.

Table 4 gives the detailed results for the instances of small sizes (n=10, 20 and 50). The first column 'Instance' indicates the name of each instance. For each instance, we list the optimal value in column 'Opt', the best found results of GVNS-TRPP in column 'GVNS-TRPP', and the best found results of IDLS-TRPP in column 'IDLS-TRPP'. From these results, we find that both IDLS-TRPP and GVNS-TRPP are able to attain the best-known solution for each instance very easily. These instances are thus easy for both algorithms.

Tables 5 and 6 show the computational results of the compared algorithms over the 100-customer and 200-customer instances. The first two columns 'Instance' and 'BestEver' list the names of instances and the best found values in the literature respectively. The next four columns indicate respectively the best value (column 'Best'), average value of 20 runs (column 'Average'), worst value (column 'Worst') and average time to attain the best objective value of 20 runs (column 'Time') for the reference algorithm GVNS-TRPP. The following four columns show the same information for IDLS-TRPP. The last column 'δ' gives the improvement of our algorithm compared to GVNS-TRPP, in terms of the best found value. The row 'Avg.' lists the average value of each column. Dominating best values are highlighted in bold, which indicate improved best-known results (with the improvement indicated by 'δ'). From these tables, one observes that IDLS-TRPP and GVNS-TRPP

Table 3
Overall results of IDLS-TRPP and the main reference algorithm GVNS-TRPP (Pei et al., 2020) on the seven sets of benchmark instances obtained under the same execution time. The timing information of GVNS-TRPP for the three sets of small instances ($n = 10, 20, 50$) is unavailable.

Size	GVNS-TRPP			IDLS-TRPP			*imp*
	Best	Average	Tavg	Best	Average	Tavg	
10	1785.70	1785.70	*	1785.70	1785.70	0.01	0
20	7965.80	7965.80	*	7965.80	7965.80	0.02	0
50	50382.90	50382.90	*	50382.90	50382.90	1.23	0
100	211879.40	211871.53	8.82	**211879.70**	211858.82	13.87	0.0001%
200	851445.80	851282.73	57.36	**851452.20**	851265.30	71.45	0.0008%
500	6637633.35	6622638.41	404.72	**6639248.90**	**6627811.94**	413.91	0.0243%
1000+	13202607.26	13160262.98	931.58	**13217678.58**	**13180951.73**	957.84	0.1142%
Win/Match/Fail				36/95/9			

[+] The result instance 1000.13 reported by GVNS-TRPP (Pei et al., 2020) is abnormal. For fair comparison, the averaged values here are the results excluding this instance. More detailed information is presented in Table 8.

Table 4
Computional results of the instances with $n = 10, 20, 50$. The optimal values of 10-customer and 20-customer instances were reported in Dewilde et al. (2013). We use 'Unk' to indicate 'Unknown optimal results' for the 50-customer instances.

Instance	n=10			n=20			n=50		
	Opt	GVNS-TRPP	IDLS-TRPP	Opt	GVNS-TRPP	IDLS-TRPP	Opt	GVNS-TRPP	IDLS-TRPP
1	2520	2520	2520	8772	8772	8772	Unk	50921	50921
2	1770	1770	1770	10174	10174	10174	Unk	52594	52594
3	1737	1737	1737	7917	7917	7917	Unk	52144	52144
4	2247	2247	2247	7967	7967	7967	Unk	45465	45465
5	2396	2396	2396	7985	7985	7985	Unk	45489	45489
6	1872	1872	1872	7500	7500	7500	Unk	55630	55630
7	1360	1360	1360	9439	9439	9439	Unk	44302	44302
8	1696	1696	1696	7999	7999	7999	Unk	55801	55801
9	1465	1465	1465	6952	6952	6952	Unk	44964	44964
10	1014	1014	1014	8582	8582	8582	Unk	47071	47071
11	1355	1355	1355	7257	7257	7257	Unk	51912	51912
12	1817	1817	1817	6857	6857	6857	Unk	53567	53567
13	1585	1585	1585	7043	7043	7043	Unk	46830	46830
14	2122	2122	2122	6964	6964	6964	Unk	52665	52665
15	1747	1747	1747	6270	6270	6270	Unk	58856	58856
16	1635	1635	1635	8143	8143	8143	Unk	49754	49754
17	2025	2025	2025	10226	10226	10226	Unk	42525	42525
18	1783	1783	1783	7625	7625	7625	Unk	40536	40536
19	1797	1797	1797	7982	7982	7982	Unk	55346	55346
20	1771	1771	1771	7662	7662	7662	Unk	61286	61286

perform similarly in terms of each performance indicator (Best, Average, Worst). This is confirmed by the Wilcoxon signed rank test applied to each pair comparison, leading to p-values superior to 0.05. However, it is worth noting that our algorithm achieves five record-breaking results (new lower bounds) including one 100-customer instance and four 200-customer instances (indicated by a positive 'δ' value).

Tables 7 and 8 show the comparative results of IDLS-TRPP and GVNS-TRPP for the sets of 500-customer and 1000-customer instances, respectively. In addition to the same quality information as before (Best, Average, Worst), the last row 'p-value' reports the results of the Wilcoxon signed rank test applied to the pair of values of each quality indicator. The dominating values for each quality indicator are indicated in bold.

From Tables 7 and 8, one observes that our algorithm globally dominates GVNS-TRPP for these large instances. For the 20 instances with 500-customers, IDLS-TRPP finds 15 new best solutions, even if it performs worse than GVNS-TRPP for the five remaining instances. The Wilcoxon signed rank test (p-value < 0.05) confirms that IDLS-TRPP performs significantly better than GVNS-TRPP in terms of the best objective value for this set of instances. As to the average and worst results, the global Avg. values indicate a better performance of IDLS-TRPP compared to GVNS-TRPP with statistically significant differences (p-values < 0.05).

Very similar observations can be made for the set of 20 largest instances with 1000-customers for which 16 new record-breaking results are reached. From row 'Avg.', one observes that IDLS-TRPP dominates GVNS-TRPP in terms of the best, average and worst results, which are confirmed by the corresponding Wilcoxon signed rank test (p-value < 0.05).

The dominance of IDLS-TRPP over GVNS-TRPP for these two sets of large instances in terms of each quality indicator is confirmed by the small p-values (< 0.05) from the Wilcoxon signed rank tests. Finally, it is interesting to observe that these improved results can be obtained by IDLS-TRPP with only a small increase of the computation time compared to the time required by GVNS-TRPP.

This experiment demonstrates the particular usefulness of the proposed algorithm for solving large and challenging TRPP instances, even if it performs very well on instances of smaller sizes as well.

4. Analysis of the key components

This section experimentally investigates the influences of two key components of the proposed algorithm: the intensification strategy introduced (Section 2.1) and the KES heuristic (Section 2.3.3). For these experiments, we focus on the more challenging instances with 200 and more customers. All the algorithmic variants tested in this section were run with the setup in Section 3.1 and their results are compared with the results of IDLS-TRPP reported in Table 3.

4.1. Influence of the intensification-driven mechanism

As explained in Section 2.1, the IDLS-TRPP algorithm uses an intensification mechanism inspired by the DGLS method introduced in Porumbel and Hao (2020) to intensively explore surrounding areas of each elite solution. This section experimentally investigates the influence

Table 5
Experimental results of the proposed algorithm IDLS-TRPP and the main reference algorithm GVNS-TRPP over the set of 100-customer instances. The results of column 'BestEver' are collected from the literature (Avci & Avci, 2017, 2019; Dewilde et al., 2013; Lu et al., 2019; Pei et al., 2020).

Instance	BestEver	GVNS-TRPP				IDLS-TRPP				δ
		Best	Average	Worst	Time	Best	Average	Worst	Time	
100.1	209952	209952	209952.00	209952	1.89	209952	209952.00	209952	2.32	0
100.2	196318	196318	196313.00	196268	22.92	196318	196296.10	196268	24.62	0
100.3	211937	211937	211937.00	211937	5.15	211937	211937.00	211937	6.56	0
100.4	217685	217685	217685.00	217685	2.14	217685	217685.00	217685	6.85	0
100.5	215119	215119	215119.00	215119	5.52	215119	215035.00	214879	6.93	0
100.6	228687	228687	228687.00	228687	2.23	228687	228687.00	228687	6.51	0
100.7	200064	200064	200064.00	200064	6.36	200064	200063.20	200056	25.33	0
100.8	205760	205760	205760.00	205760	8.93	205760	205715.75	205583	14.62	0
100.9	226240	226240	226240.00	226240	0.79	226240	226240.00	226240	6.26	0
100.10	218202	218202	218202.00	218202	1.19	218202	218202.00	218202	7.24	0
100.11	212503	212503	212442.00	212381	5.17	212503	212480.80	212392	30.45	0
100.12	222249	222249	222249.00	222249	2.25	222249	222249.00	222249	8.27	0
100.13	206957	206957	206957.00	206957	0.99	206957	206957.00	206957	10.73	0
100.14	215690	215690	215690.00	215690	2.41	215690	215690.00	215690	5.32	0
100.15	214041	214041	214041.00	214041	16.40	214041	213990.10	213531	30.28	0
100.16	214036	214036	213976.80	213740	13.89	214036	213929.05	213673	24.16	0
100.17	223636	223636	223635.85	223633	25.70	**223642**	223641.50	223640	10.44	6
100.18	192849	192849	192849.00	192849	4.47	192849	192849.00	192849	6.98	0
100.19	206755	206755	206723.00	206627	20.62	206755	206741.20	206607	24.65	0
100.20	198908	198908	198908.00	198908	27.42	198908	198835.75	198693	18.86	0
Avg.	211879.40	211879.40	211871.53	211849.45	8.82	**211879.70**	211858.82	211788.50	13.87	
p-value						3.17×10^{-1}	9.26×10^{-2}	2.84×10^{-2}		

Table 6
Experimental results of the proposed algorithm IDLS-TRPP and the main reference algorithm GVNS-TRPP over the set of 200-customer instances. The results of column 'BestEver' are collected from the literature (Avci & Avci, 2017, 2019; Dewilde et al., 2013; Lu et al., 2019; Pei et al., 2020).

Instance	BestEver	GVNS-TRPP				IDLS-TRPP				δ
		Best	Average	Worst	Time	Best	Average	Worst	Time	
200.1	877610	877610	877410.10	876549	62.46	877610	877400.65	876550	66.78	0
200.2	901898	901898	901495.70	901184	45.68	**901927**	901472.25	900516	71.02	29
200.3	888393	888393	888393.00	888393	22.06	888393	888393.00	888393	42.37	0
200.4	873910	873910	873812.60	873467	64.70	873910	873706.50	873424	79.28	0
200.5	849358	849358	849186.65	848111	37.10	849358	849000.80	847939	54.74	0
200.6	816928	816928	816916.65	816910	67.05	816928	816914.50	816909	99.39	0
200.7	784120	784120	784109.20	784059	65.02	784120	784109.35	784012	91.62	0
200.8	838026	838026	837888.05	837075	71.34	**838100**	837919.35	837208	75.62	74
200.9	891203	891203	891030.65	890637	42.33	891203	891072.10	890637	55.51	0
200.10	847303	847303	847019.15	845931	56.12	**847308**	846958.80	845899	83.91	5
200.11	804851	804851	804543.60	804087	43.27	804851	804651.85	804372	74.06	0
200.12	808966	808966	808905.20	808293	34.86	808966	808820.30	808071	33.92	0
200.13	861749	861749	861674.70	861006	64.87	861749	861642.00	861006	86.19	0
200.14	850601	850601	850588.70	850509	77.25	**850621**	850551.45	850397	77.73	20
200.15	848006	848006	847832.10	846711	76.62	848006	847643.35	846662	77.14	0
200.16	854075	854075	853813.70	852452	49.50	854075	853880.10	853099	82.44	0
200.17	861747	861747	861491.05	860397	58.80	861747	861471.35	861117	65.03	0
200.18	842953	842953	842720.30	841618	86.46	842953	842653.20	841742	87.83	0
200.19	822881	822881	822708.75	821919	60.00	822881	822735.90	821664	71.40	0
200.20	904338	904338	904114.75	903132	61.66	904338	904309.20	904295	53.08	0
Avg.	851445.80	851445.80	851282.73	850622.00	57.36	**851452.20**	851265.30	850695.60	71.45	
p-value						6.79×10^{-2}	3.34×10^{-1}	9.25×10^{-1}		

of this mechanism over the performance of IDLS-TRPP. For this purpose, we create an algorithmic variant ILS-TRPP by setting the search depth limit R to a very high value and keeping the other IDLS-TRPP components unchanged (i.e., lines 16–18 of Algorithm 1 will not be executed). Doing this disables the intensification mechanism because only one (long) iterated local search run instead of multiple bounded local search runs will be launched from the elite solution.

Table 9 summarizes the comparative results between ILS-TRPP and IDLS-TRPP with the same information as in Table 3 along with the last column 'p-values' from the Wilcoxon signed rank test applied to the best results of the compared algorithms for each set of instances. One observes that IDLS-TRPP outperforms ILS-TRPP in terms of the best and average results. The statistical significant difference in terms of the best results of the compared algorithms for the three sets of instances is confirmed by the small p-values < 0.05. This experiment demonstrates the relevance of the intensification mechanism used by the IDLS-TRPP algorithm.

Furthermore, to study the behaviors of the two compared algorithms throughout the execution, we performed an additional experiment to obtain the convergence charts (running profiles) of the algorithms on four representative and difficult instances: two 500-customer instances (*500.1* and *500.2*) and two 1000-customer instances (*1000.1* and *1000.2*). For this experiment, we ran each algorithm 20 times to solve each instance with the time limit of 500 s (for 500-customer instances) and 1000 s (for 1000-customer instances). The best objective values are recorded during the executions.

Fig. 3 shows the convergence charts that indicate how the average best objective value found of 20 runs by each algorithm (y-axis) evolves as a function of the running time of the algorithm (x-axis). We observe that both algorithms are able to attain good-quality solutions quickly (within 50 s) but IDLS-TRPP has a better long-term performance.

Table 7
Experimental results of the proposed algorithm IDLS-TRPP and the main reference algorithm GVNS-TRPP over the set of 500-customer instances. The results of column 'BestEver' are collected from the literature (Avci & Avci, 2017, 2019; Dewilde et al., 2013; Lu et al., 2019; Pei et al., 2020).

Instance	BestEver	GVNS-TRPP				IDLS-TRPP				δ
		Best	Average	Worst	Time	Best	Average	Worst	Time	
500.1	6678468	6678468	6664032.05	6651139	409.43	**6679339**	6666525.45	6655224	400.79	871
500.2	6463424	6463424	6449712.40	6433960	393.78	**6464121**	6452465.40	6441483	443.56	697
500.3	6425689	6425689	6410715.05	6399450	386.31	**6432179**	6421381.95	6400056	405.62	6490
500.4	6678804	6678804	6668202.20	6648148	368.07	**6679323**	6673726.55	6667802	407.93	519
500.5	6841247	6841247	6829442.80	6812340	423.51	**6846611**	6832527.20	6820874	408.30	5364
500.6	6166077	**6166077**	6155034.80	6140340	410.65	6165804	6155572.35	6148208	417.51	−273
500.7	6823818	6823818	6800739.45	6779367	413.40	**6825544**	6811925.75	6799550	445.83	1726
500.8	6588450	6588450	6575473.35	6552773	408.05	**6591982**	6577867.45	6564889	379.76	3532
500.9	6627147	6627147	6613714.25	6599039	390.71	**6630592**	6622398.25	6611337	399.06	3445
500.10	6762936	6762936	6749197.65	6731296	432.28	**6766107**	6754708.45	6740732	439.04	3171
500.11	6793871	6793871	6775750.75	6756956	403.40	**6794863**	6783465.20	6770846	424.63	992
500.12	6543323	**6543323**	6527608.85	6511491	406.48	6539219	6532311.45	6520929	427.70	−4104
500.13	6426788	6426788	6411856.40	6391554	415.56	**6429881**	6416052.75	6404018	433.20	3093
500.14	6799091	6799091	6787843.90	6778724	380.59	**6804111**	6792957.15	6776421	411.06	5020
500.15	6820990	6820990	6804416.95	6786935	445.51	**6823083**	6808188.85	6792175	403.90	2093
500.16	6643189	**6643189**	6627300.95	6608062	424.06	6641958	6633700.50	6622930	376.53	−1231
500.17	6656804	**6656804**	6638537.50	6628560	404.32	6653016	6643332.25	6635151	405.76	−3788
500.18	6635000	**6635000**	6613361.55	6586965	368.04	6632732	6617651.95	6599206	402.90	−2268
500.19	6680645	6680645	6671727.90	6658717	392.94	**6685770**	6673627.35	6659688	411.35	5125
500.20	6696906	6696906	6678099.35	6661584	417.34	**6698743**	6685852.55	6666850	433.79	1837
Avg.	6637633.35	6637633.35	6622638.41	6605870.00	404.72	**6639248.90**	6627811.94	6614918.45	413.91	
p-value						3.33×10^{-2}	8.86×10^{-5}	1.40×10^{-4}		

Table 8
Experimental results of the proposed algorithm IDLS-TRPP and the main reference algorithm GVNS-TRPP over the set of 1000-customer instances.

Instance	BestEver (Pei et al., 2020)	GVNS-TRPP				IDLS-TRPP				δ
		Best	Average	Worst	Time	Best	Average	Worst	Time	
1000.1	16037164	**16037164**	15974489.00	15909332	904.96	16036393	15995572.10	15970099	968.22	−771
1000.2	14442450	14442450	14405988.20	14382659	929.04	**14458979**	14429777.90	14392702	970.38	16529
1000.3	10924880	10924880	10885626.70	10831905	936.74	**10938518**	10906962.60	10879692	954.27	13638
1000.4	13418699	13418699	13370499.40	13345724	926.36	**13436056**	13387838.40	13331062	961.10	17357
1000.5	17050897	17050897	16999503.55	16962045	940.33	**17053027**	17022904.80	16992212	951.53	2130
1000.6	11803559	**11803559**	11750602.40	11707115	938.86	11792759	11765238.85	11731592	966.89	−10800
1000.7	12964458	**12964458**	12912944.25	12871634	920.78	12962632	12922063.95	12885054	962.51	−1826
1000.8	12572790	12572790	12534832.55	12484217	948.00	**12585070**	12544422.05	12500769	966.05	12280
1000.9	13938794	13938794	13871530.70	13810239	924.42	**13954128**	13911055.50	13832805	964.15	15334
1000.10	9962230	9962230	9917128.65	9871772	895.18	**9967005**	9934067.80	9875296	969.09	4775
1000.11	9242217	9242217	9211349.00	9166426	906.42	**9261025**	9229624.40	9206101	943.67	18808
1000.12	15124136	15124136	15086881.85	15038413	939.08	**15136887**	15099695.15	15057482	967.06	12751
1000.13[a]	15092052	**15092052**	15092052.00	15092052	993.81	10422643	10392569.50	10358825	966.57	−4669409
1000.14	11868317	11868317	11868317.00	11868317	1000.43	**11908334**	11867924.40	11835574	930.45	40017
1000.15	15767145	15767145	15728148.60	15682402	933.92	**15784134**	15739122.40	15698031	961.52	16989
1000.16	15140757	15140757	15100909.50	15060903	942.69	**15161694**	15112711.15	15042722	950.01	20937
1000.17	10175909	10175909	10145812.15	10135285	946.96	**10215460**	10186308.35	10153272	955.39	39551
1000.18	15059293	15059293	15014848.50	14989081	946.70	**15094532**	15068819.55	15033978	950.83	35239
1000.19	12385606	12385606	12340836.40	12314951	915.36	**12410568**	12375251.65	12325767	942.70	24962
1000.20	12970237	12970237	12924748.30	12887476	903.75	**12978692**	12938721.90	12889056	963.19	8455
Avg.[b]	13202607.26	13202607.26	13160262.98	13122099.79	931.58	**13217678.58**	13180951.73	13138592.95	957.84	
p-value						5.39×10^{-4}	1.55×10^{-4}	1.00×10^{-2}		

[a]The result of 15092052 for instance 1000.13 reported in Pei et al. (2020) is abnormal and wrong because it is larger than the upper bound of 14598152, that is obtained by using Equation (3): $f(\varphi) = \sum_{i=0}^{m} p_{x_i} - \sum_{i=0}^{m-1} (m-i) \cdot d_{x_i,x_{i+1}} \le \sum_{i=1}^{m} p_{x_i} \le \sum_{i=1}^{n} p_i$.

[b]The average value of the best found results is the result by excluding the instance of 1000.13.

Table 9
Overall results obtained by ILS-TRPP and IDLS-TRPP.

Size	ILS-TRPP			IDLS-TRPP			*imp*	p-value
	Best	Average	Tavg	Best	Average	Tavg		
200	851273.95	850882.72	100.29	**851452.20**	**851265.30**	71.45	0.0209%	8.84×10^{-5}
500	6615207.85	6602084.57	231.79	**6639248.90**	**6627811.94**	413.91	0.3634%	8.86×10^{-5}
1000[+]	12987142.35	12944792.46	298.98	**13077926.80**	**13041532.62**	958.28	0.6990%	8.86×10^{-5}

[+] The results here consider the experimental results obtained by the instance 1000.13, while Table 3 excludes the results of 1000.13 because of the fair comparison with GVNS-TRPP.

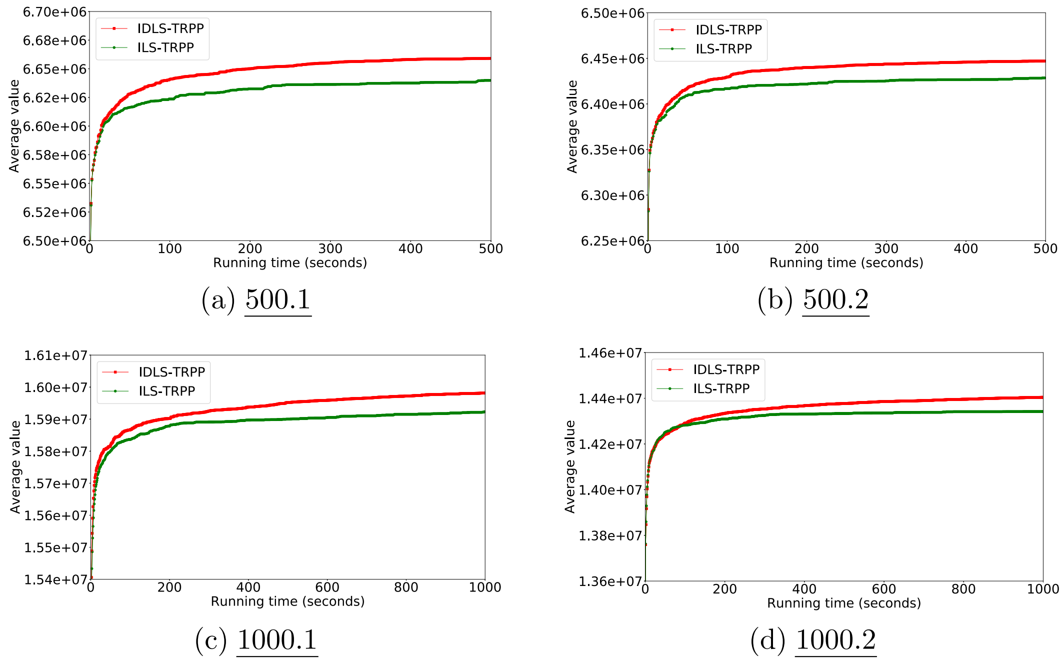


Fig. 3. Convergence charts (running profiles) of ILS-TRPP and IDLS-TRPP for solving four representative difficult instances (*500.1*, *500.2*, *1000.1* and *1000.2*). The results were obtained from 20 independent executions of each compared algorithm.

Table 10
Overall results obtained by IDLS-TRPP-noKES and IDLS-TRPP over 60 benchmark instances within same execution time.

Size	IDLS-TRPP-noKES			IDLS-TRPP			*imp*	p-value
	Best	Average	Tavg	Best	Average	Tavg		
200	851452.15	851201.23	78.88	**851452.20**	**851265.30**	71.45	0.0001%	6.55×10^{-1}
500	6636032.05	6622711.91	373.34	**6639248.90**	**6627811.94**	413.91	0.0485%	4.85×10^{-3}
1000[+]	13030820.05	12985507.28	491.17	**13077926.80**	**13041532.62**	958.28	0.3615%	8.86×10^{-5}

[+] The results here consider the experimental results obtained by the instance 1000.13, while Table 3 excludes the results of 1000.13 because of the fair comparison with GVNS-TRPP.

Indeed, ILS-TRPP generally begins to stagnate at its local optimum solution after some 200 s, while IDLS-TRPP continues to improve its solutions till the end of the time limit, showing a very favorable search behavior. This experiment shows that the intensification mechanism contributes favorably to the performance of the IDLS-TRPP algorithm.

4.2. Influence of the KES heuristic

To study the impacts of the *KES* heuristic on the performance of the algorithm, we created a variant IDLS-TRPP-noKES by disabling the *KES* heuristic (i.e., removing line 15 in Algorithm 3). We ran IDLS-TRPP-noKES with the same experimental setting as in Section 3.1 to make sure that both algorithms were performed using the same cutoff-time for each tested instance.

Using the same column headings as Table 9, Table 10 shows that IDLS-TRPP significantly dominates IDLS-TRPP-noKES, especially on the large size instances (n=500 and 1000), according to the Wilcoxon signed rank tests. One can conclude that the KES heuristic contributes positively to the proposed algorithm and is especially useful for solving instances of large size ($n > 200$).

To further study the influence of the N_{kes} neighborhood on the local optimization procedure, we extract EVNS from IDLS-TRPP by deleting the perturbation phase as well as the intensification mechanism, and create a variant: EVNS-noKES (disabling N_{kes}).

A supplementary experiment was conducted using these variants on 4 difficult and representative instances (*500.1*, *500.2*, *1000.1* and *1000.2*). For this experiment, each instance was solved 100 times by each algorithm until no improving solution exists in the neighborhoods. The best found solutions and the running time are recorded.

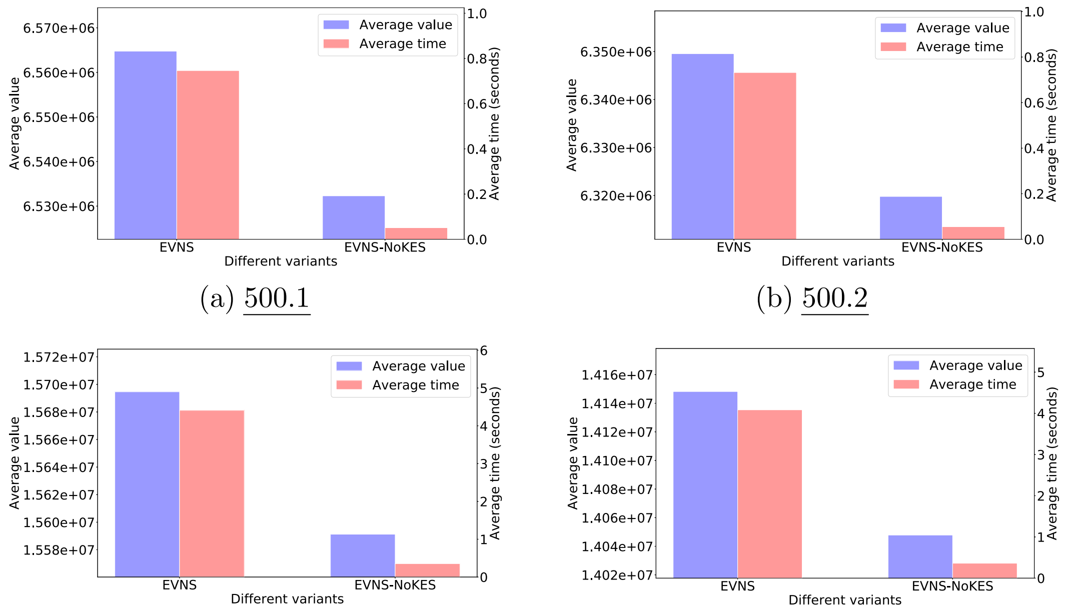
Fig. 4 summarizes the corresponding bar charts that describe how the average objective values (y-axis in the left, blue bars) and average running time (y-axis in the right, red bars) differ between the two variants. One can observe that EVNS which combines the *KES* heuristic with other neighborhoods outperforms EVNS-NoKES in terms of the best found solutions (blue bars) for all the cases. Although EVNS spends more time than EVNS-NoKES (e.g., 0.746 s vs 0.051 s for the instance *500.1*), EVNS is able to obtain good-quality solutions which are never achieved by EVNS-NoKES.

To summarize, EVNS combining the *KES* heuristic (which is powerful but time-consuming) and other neighborhoods makes a good trade-off between the computation time and solution quality. The experiments presented in this section confirm the positive role of the *KES* heuristic on the algorithm performance.

5. Conclusions

In this work, we presented an intensification-driven local search for solving the traveling repairman problem with profits. This algorithm integrates several innovative ingredients including the tree-like intensification mechanism inspired by the general DGLS framework (Porumbel & Hao, 2020), the K-exchange sampling neighborhood together with the associated *KES* heuristic inspired by the Lin–Kernighan heuristic, neighborhood reduction based on the candidate set strategy and fast evaluation techniques.

The experimental results over 140 benchmark instances showed that the proposed algorithm performs remarkably well and in particularly updates the best-known results for 36 difficult instances. These new



(a) 500.1 (b) 500.2 (c) 1000.1 (d) 1000.2

Fig. 4. Bar charts of EVNS and EVNS-noKES for solving four representative difficult instances (*500.1*, *500.2*, *1000.1* and *1000.2*). The results were averaged over 100 independent executions of each compared algorithm.

results will be useful to assess other TRPP algorithms. Additional experiments demonstrated the positive roles of the intensification mechanism and the K-exchange based heuristic to the algorithm performance.

Even if important progresses have been made in recent year for solving the TRPP, this work shows that improvements are still possible with simple and effective ideas. This work also demonstrates the potential interest of the DGLS framework (Porumbel & Hao, 2020), which can boost an underlying local search algorithm with the help of a tree-like intensification mechanism.

Given that the TRPP has a number of practical applications, the code of our algorithm that we will make publicly available can be used to solve some of these applications. The proposed algorithm or its components can also be integrated into more sophisticated methods such as hybrid evolutionary algorithms to build more powerful solution methods for this challenging problem.

CRediT authorship contribution statement

Jintong Ren: Conceptualization, Investigation, Data curation, Writing – original draft, Software. **Jin-Kao Hao:** Conceptualization, Validation, Writing – review & editing. **Feng Wu:** Supervision, Writing – review & editing. **Zhang-Hua Fu:** Supervision, Methodology, Validation, Writing – review & editing, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research was partially supported by the Shenzhen Science and Technology Innovation Commission, China under grant JCYJ20180508 162601910, the National Key R&D Program of China under grant 2020YFB1313300, and the Funding from the Shenzhen Institute of Artificial Intelligence and Robotics for Society, China under grant AC01202101110. The authors would like to thank the anonymous reviewers for their insightful comments that helped us to improve the paper. Thanks also go to Jia-Ming Xin for his help.

References

Afrati, F., Cosmadakis, S., Papadimitriou, C. H., Papageorgiou, G., & Papakostantinou, N. (1986). The complexity of the travelling repairman problem. *RAIRO-Theoretical Informatics and Applications-Informatique Théorique et Applications*, *20*(1), 79–87.

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.

Avci, M. G., & Avci, M. (2019). An adaptive large neighborhood search approach for multiple traveling repairman problem with profits. *Computers & Operations Research*, *111*, 367–385.

Bentley, J. J. (1992). Fast algorithms for geometric traveling salesman problems. *ORSA Journal on Computing*, *4*(4), 387–411.

Blum, A., Chalasani, P., Coppersmith, D., Pulleyblank, B., Raghavan, P., & Sudan, M. (1994). The minimum latency problem. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing* (pp. 163–171).

Dantzig, G. B., & Ramser, J. H. (1959). The truck dispatching problem. *Management Science*, *6*(1), 80–91.

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

Flood, M. M. (1956). The traveling-salesman problem. *Operations Research*, *4*(1), 61–75.

Frifita, S., Masmoudi, M., & Euchi, J. (2017). General variable neighborhood search for home healthcare routing and scheduling problem with time windows and synchronized visits. *Electronic Notes in Discrete Mathematics*, *58*, 63–70.

Fu, V. F., Redi, A. A. N. P., Halim, C., & Jewpanya, P. (2020). The path cover problem: Formulation and a hybrid metaheuristic. *Expert Systems with Applications*, *146*, Article 113107.

Hansen, P., & Mladenović, N. (2005). Variable neighborhood search. In *Search Methodologies* (pp. 211–238). Springer.

Helsgaun, K. (2000). An effective implementation of the Lin–Kernighan traveling salesman heuristic. *European Journal of Operational Research*, *126*(1), 106–130.

Karakostas, P., Sifaleras, A., & Georgiadis, M. C. (2019). A general variable neighborhood search-based solution approach for the location-inventory-routing problem with distribution outsourcing. *Computers & Chemical Engineering*, *126*, 263–279.

Karakostas, P., Sifaleras, A., & Georgiadis, M. C. (2020). Adaptive variable neighborhood search solution methods for the fleet size and mix pollution location-inventory-routing problem. *Expert Systems with Applications*, *153*, Article 113444.

Lin, S. (1965). Computer solutions of the traveling salesman problem. *The Bell System Technical Journal*, *44*(10), 2245–2269.

Lu, Y., Hao, J.-K., & Wu, Q. (2019). Hybrid evolutionary search for the traveling repairman problem with profits. *Information Sciences*, *502*, 91–108.

Lust, T., & Jaszkiewicz, A. (2010). Speed-up techniques for solving large-scale biobjective TSP. *Computers & Operations Research*, *37*(3), 521–533.

Mladenović, N., Urošević, D., Ilić, A., et al. (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.
nez, M. L.-I., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., & Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, *3*, 43–58.
Pei, J., Mladenović, N., Urošević, D., Brimberg, J., & Liu, X. (2020). Solving the traveling repairman problem with profits: A novel variable neighborhood search approach. *Information Sciences*, *507*, 108–123.
Porumbel, D., & Hao, J.-K. (2020). Distance-guided local search. *Journal of Heuristics*, *26*(5), 711–741.
Soylu, B. (2015). A general variable neighborhood search heuristic for multiple traveling salesmen problem. *Computers & Industrial Engineering*, *90*, 390–401.
Toth, P., & Vigo, D. (2003). The granular tabu search and its application to the vehicle-routing problem. *INFORMS Journal on Computing*, *15*(4), 333–346.
Xu, Z., & Cai, Y. (2018). Variable neighborhood search for consistent vehicle routing problem. *Expert Systems with Applications*, *113*, 66–76.