

REVISITING SEARCH METHODS FOR THE BOUNDED-DIAMETER MINIMUM SPANNING TREE PROBLEM

ROGÉRIO M. NEPOMUCENO¹, RODRIGO L. MAFORT² , FÁBIO PROTTI³ ,
ISABEL ROSSETI^{3,*} , LUIDI SIMONETTI⁴  AND EDOARDA VALLIM³

Abstract. The Bounded-Diameter Minimum Spanning Tree Problem (BDMSTP) is defined as follows: Given a graph G where each edge xy has a positive cost w_{xy} , the goal is to find an optimal spanning tree of G whose diameter does not exceed a prescribed positive integer $D \geq 2$. Applications of the BDMSTP appear in telecommunication network and fiber optic projects, data compression problems, and distributed systems design. This work revisits search methods for the BDMSTP and proposes alternative combinations of local search moves, intending to select efficient sets of moves and improve the quality of the solutions found in the literature. For small 50- and 100-node instances from the widely-used OR-Library, we obtain exact solutions whose optimal values were so far unknown. Having solved these easier cases, we can then concentrate our efforts on solving more challenging OR-Library graph instances with 250, 500, and 1000 nodes. The process of selecting efficient sets of search moves is done by encapsulating different local search versions (based on the Variable Neighborhood Descent method) in an ILS approach. Some new neighborhoods are also proposed. Computational tests show that these neighborhoods allowed the proposed ILS approach to obtain better results than those presented in the literature.

Mathematics Subject Classification. 68-XX, 90-XX.

Received April 19, 2024. Accepted January 30, 2025.

1. INTRODUCTION

Let $G = (V, E)$ be a graph with $|V| = n$. For a given positive integer $D \geq 2$, a *bounded-diameter spanning tree* of $G = (V, E)$ is a spanning tree $T = (V, E_T)$ whose diameter (the maximum distance between two nodes) does not exceed D . The cost $c(T)$ of T is the sum of the costs of its edges. The *Bounded-Diameter Minimum Spanning Tree Problem* (BDMSTP) consists of finding a bounded-diameter spanning tree of G such that $c(T)$ is minimized. Figure 1 shows an example of an optimal solution of the BDMSTP for $D = 2$. Note that a minimum spanning tree of the graph G shown in Figure 1a has cost 3 and diameter 3 (Fig. 1b). For $D = 2$, however, a bounded-diameter minimum spanning tree has cost $2 + \sqrt{2}$ (Fig. 1c).

Keywords. Spanning tree, Bounded-Diameter Minimum Spanning Tree, Iterated Local Search (ILS), heuristics.

¹ Instituto Federal do Triângulo Mineiro, Uberaba, MG, Brazil.

² Universidade do Estado do Rio de Janeiro, Rio de Janeiro, RJ, Brazil.

³ Universidade Federal Fluminense, Niterói, RJ, Brazil.

⁴ Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, Brazil.

*Corresponding author: rosseti@ic.uff.br

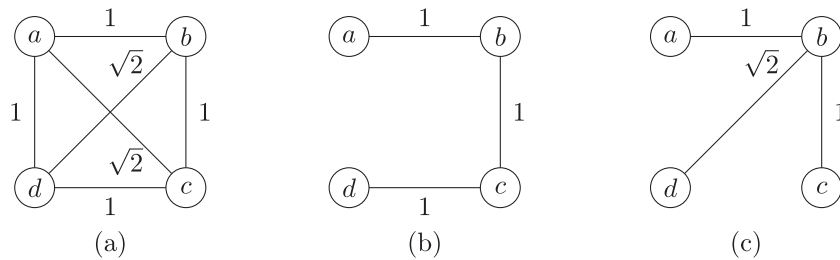


FIGURE 1. BDMSTP example.

Motivations to study the BDMSTP. Applications of the BDMSTP emerge in engineering problems, such as telecommunication network and fiber optic projects [3], data compression problems [6], and distributed systems design, when multiple exclusion is considered [39]. When $D \in \{2, 3, n - 1\}$, or when all edges have the same cost, the BDMSTP can be solved in polynomial time [1]. For $4 \leq D \leq n - 2$, however, the BDMSTP is proved to be NP-hard [11]. This means that, in practice, only relatively small instances of the BDMSTP can be solved by exact approaches in an acceptable time. Because of this fact, constructive heuristics and metaheuristics are interesting options to provide good solutions for the BDMSTP.

Contributions of this work. We apply the exact method described in [15] to optimally solve widely-used OR-Library graph instances of the BDMSTP with 50 and 100 nodes; such optimal solution values were so far unknown. Having solved these easier cases, we can then concentrate our efforts on solving more challenging OR-Library graph instances with 250, 500, and 1000 nodes. To deal with such instances, we revisit search methods for the BDMSTP and propose alternative combinations of local search moves, intending to select efficient sets of moves and improve the quality of the solutions found in the literature. The process of determining such sets of moves is done by encapsulating different local search versions into an Iterated Local Search (ILS) approach for the BDMSTP. Five new neighborhoods are proposed for the local search engine, which is a variant of the Variable Neighborhood Descent method called “Random VND”, or simply RVND [43, 45]. We show in Section 5 that this variety of different neighborhood structures contributes to the good performance of the proposed ILS-based method. In addition, we performed a comparative analysis of the neighborhoods, proposing four versions of the local search according to different subsets of neighborhoods. The IRACE package was used as an auxiliary tool to determine some of these subsets. As far as the authors know, this is the first attempt to use IRACE as a neighborhood selection tool. We also compared the results for the BDMSTP found in the literature with those obtained by the proposed ILS/RVND approach; the tests performed show that it was able to obtain better results than the best previous results. To the best of the authors’ knowledge, this is the first time an ILS/RVND approach has been proposed for the solution of the BDMSTP.

Some definitions and notation. If H is a graph, $V(H)$ and $E(H)$ denote, respectively, the node set and the edge set of H . A bounded-diameter spanning tree T of G is simply referred to as a BDST. For a BDST T of G and a node $v \in V$, $d_T(v)$ denotes the degree of v in T ; we drop the subscript when T is clear from the context.

Organization of the paper. Section 2 is a literature review on the BDMSTP. Section 3 contains a brief review of the exact method presented in [15] for the solution of the BDMSTP. Section 4 presents the proposed ILS/RVND-based heuristic for the BDMSTP, describing in detail the neighborhoods to be used for local search and the perturbation moves. Computational results are reported in Section 5. Section 6 contains our conclusions.

2. LITERATURE REVIEW

In this section, previous exact and heuristic approaches for the BDMSTP are presented.

Exact approaches for the BDMSTP. Several formulations for the BDMSTP are described in the literature. Achuthan *et al.* [2] presented a Mixed Integer Linear Programming formulation, as well as some branch-and-bound procedures, enabling the solution of complete graph instances with up to 50 nodes and diameter 4. Gouveia and Magnanti [13] described a model based on network flows, which solves random and Euclidean instances¹ with up to 100 nodes and 1000 edges, and diameter less than or equal to 8. Santos *et al.* [40] proposed an improvement of the method presented in [2], and solved Euclidean instances with up to 60 nodes and 150 edges with diameter 5, and 40 nodes and 100 edges with diameter 10. Gruber and Raidl [16] presented a 0–1 Integer Linear Programming model that solves random and Euclidean instances with up to 30 nodes, 200 edges, and diameter 8. Gruber and Raidl [18] also proposed another ILP formulation based on jump cuts, solving it *via* a branch-and-cut algorithm using various heuristics to deal with the separation problem (constructive heuristics, local search, and Tabu search); this method is applied to random-weight and Euclidean instances with 80 nodes, 800 edges, and diameter 9. Gouveia *et al.* [15] introduced a branch-and-cut algorithm capable of solving complete graph instances with up to 161 nodes and diameter 8.

Constructive heuristics for the BDMSTP. Many constructive heuristics are described in the literature for the BDMSTP. One of the best known is the One-Time Tree Construction (OTTC) heuristic, by Abdalla and Deo [1], which is based on Prim’s algorithm [36]. Raidl and Julstrom [37] proposed the Random Greedy Heuristic (RGH), which produces lower cost solutions when compared with OTTC. Julstrom [24] presented the Center-Based Tree Construction (CBTC) heuristic, which builds solutions with lower costs if compared with OTTC, especially when large diameters are considered. Still according to Julstrom [24], OTTC and CBTC behave in an “extremely greedy” way when applied to Euclidean instances of small diameters; as an alternative, the author presents the Randomized Center-Based Tree Construction (RTC) heuristic, which yields lower cost trees, when considering small diameters, if compared with OTTC and CBTC. In the case of larger diameters, CBTC presents lower cost trees than OTTC and RTC. The Center-Based Recursive Clustering (CBRC) heuristic, presented by Binh *et al.* [5], produces lower cost trees when compared with OTTC, RGH, and CBTC. More recently, new constructive heuristics developed for the BDMSTP show competitive results in comparison with the previous constructive heuristics [34, 42, 44]; however, such heuristics are not able to approach the results obtained by metaheuristic-based methods. Steitz [44] investigates how the structure of good solutions change depending on the diameter of the tree: “for loose bounds [*i.e.*, *high diameter values*], optimal solutions are similar to the much easier minimum spanning tree problem, and greedy heuristics perform best; in contrast, these approaches fail for tight diameter bounds” [44].

Metaheuristics applied to the BDMSTP. Several metaheuristic-based methods to solve the BDMSTP are also described in the literature. Raidl and Julstrom [37] proposed an evolutionary algorithm called Edge-Set-Code Evolutionary Algorithm (ESCEA), which presents better results than those presented by OTTC and RGH. Julstrom and Raidl [25] also presented another evolutionary algorithm called Permutation-Coded Evolutionary Algorithm (PCEA), which obtains better results than the ESCEA. Julstrom [23] proposed the Random Key Evolutionary Algorithm (RKEA), which presents solutions similar to those presented by the PCEA. Gruber and Raidl [17] presented a heuristic based on Variable Neighborhood Search (VNS), using a Variable Neighborhood Descent (VND) method as local search [12, 21, 22, 32], with better results than those obtained by the ESCEA, PCEA, and RKEA. Gruber *et al.* [19] presented the Level Encoded Evolutionary Algorithm (LEEA) and an Ant Colony Optimization (ACO) approach. In tests performed with non-prefixed time, the ACO approach shows the best results, when compared with the VNS and the LEEA. On the other hand, in tests with prefixed time, the LEEA shows the best results, overcoming the ACO approach and the VNS. Singh and Gupta [41] described a PCEA improvement, called PEA-I, that obtains better results than the ESCEA and PCEA. A hybrid algorithm (called here GILS), combining the principles of Greedy Randomized Adaptive Search (GRASP) and ILS, is proposed by Lucena *et al.* [29], obtaining the best results in 50% of the cases when compared with the best results presented by the VNS, ESCEA, and PCEA. It is worth remarking that Raidl and Julstrom [38] proposed

¹ An *Euclidean* instance consists of a complete graph G where each node x is associated with a point P_x in \mathbb{R}^2 and each edge xy has a cost w_{xy} equal to the Euclidean distance between P_x and P_y .

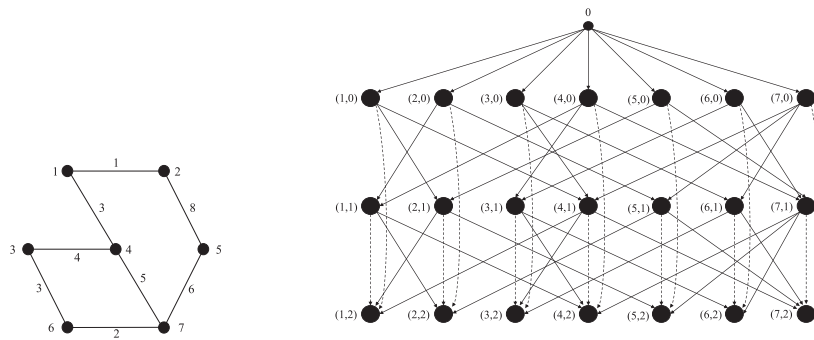


FIGURE 2. Transformation from the BDMSTP to the STP. On the left: instance G for the BDMSTP with $D = 4$. On the right: layered digraph H , the associated instance of the STP. (Example extracted from [15].)

a general representation of spanning trees in Evolutionary Algorithms for network design problems, describing initialization, recombination, and mutation operators for this representation. To conclude this section, Torkestani [46] presented an adaptive Learning Automata-based heuristic (LABD), and, more recently, Patvardhan and Prakash [33] presented a two-phase memetic algorithm (2PMA) for the BDMSTP.

3. AN EXACT METHOD FOR THE BDMSTP

In this section we briefly review the exact method proposed in [15] for the solution of the BDMSTP. The method first transforms the BDMSTP into a directed Steiner Tree Problem (STP) in a layered digraph, as follows. Suppose the diameter D is even. In this case, the following property (see [15]) is valid: *a tree T has diameter at most D if and only if there is a node c of T such that, for any node x of T , the path from c to x in T contains at most $D/2$ edges*. The idea is that c can be set as the root of T so that the height of T does not exceed $D/2$, that is, every path in T has at most D edges. This property is essential to transform the BDMSTP into the STP for an even value of D , as shown in Figure 2.

The graph G on the left-hand side is an instance of the BDMSTP with $D = 4$, and the layered digraph H on the right-hand side is the associated instance of the STP, containing a root node 0 and nodes $(j, k)^2$ for $j \in V$ and $k \in \{0, \dots, D/2\}$. Layer L_k consists of the nodes (j, k) , $j \in V$. Layer $L_{D/2}$ is the set R of required nodes. Arcs leaving node 0 have cost zero. Arcs from a layer L_{k-1} to L_k have costs equal to the costs of the corresponding edges in G . The dotted arcs $\langle (j, k), (j, D/2) \rangle$, for $j \in V$ and $0 \leq k \leq D/2 - 1$, have cost zero. A solution of the BDMSTP with input (G, D) amounts to determining a Steiner tree T_0 in H , rooted at node 0, such that the degree of node 0 is 1 and the leaves of T_0 are precisely those in layer $L_{D/2}$. Figure 3 shows a pair of solutions. Every bounded-diameter spanning tree T_c in G , centered at node c (node 4 in Fig. 3), corresponds to a Steiner tree T_0 in H with the same cost as T_c and depth $D/2 + 1$. Note that the edge leaving node 0 in T_0 determines the node c of G that will serve as the center of T_c . Conversely, any Steiner tree in H with leaves in $L_{D/2}$ and exactly one arc leaving node 0 corresponds to a spanning tree in G with the same cost, whose diameter does not exceed D .

For the case where D is odd, the following property is used in the transformation from the BDMSTP to the STP (see [15]): *for an odd integer D , a tree T has diameter bounded by D if and only if there are adjacent nodes c and c' of T (the centers) such that, for any node x of T , either the path from x to c or the path from x to c' in T contains at most $\lfloor D/2 \rfloor$ edges*. The idea is similar to the case where D is even, but now the “root”

² Although this notation is the same as the one used to represent undirected edges, we chose it in order to keep the same description of the transformation from the BDMSTP to the STP in [14]. Throughout the text, the meaning is clear from the context.

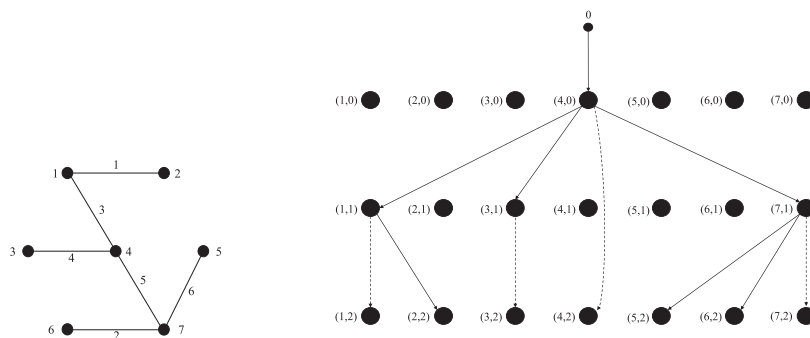


FIGURE 3. A solution of the BDMSTP on the left ($D = 4$), and the associated solution of the STP on the right. (Example extracted from [15].)

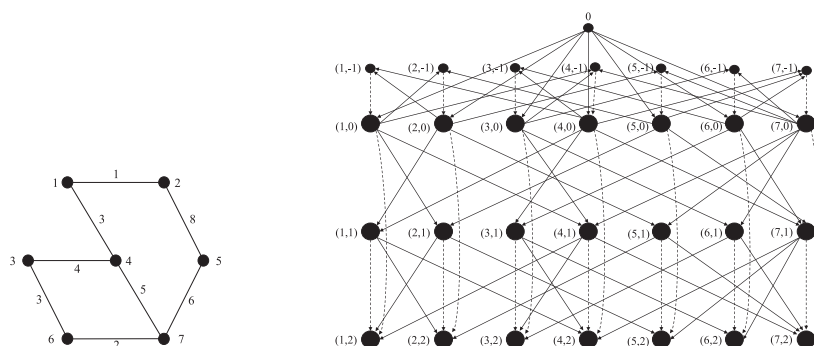


FIGURE 4. Transformation from the BDMSTP to the STP for odd diameter ($D = 5$). (Example extracted from [15].)

of T can be seen as a pair of adjacent nodes from which all the paths to the leaves depart. For this situation, the transformation uses two dummy layers $L_0 = \{(j, 0) \mid j \in V\}$ and $L_{-1} = \{(j, -1) \mid j \in V\}$, corresponding to the possible choices of the central edge (c, c') . See Figure 4, where G is an instance of the BDMSTP with $D = 5$. The root node 0 in the layered digraph H is connected to all nodes in L_0 by arcs with cost zero. A degree constraint must be added so that exactly one of these arcs is chosen. If (i, j) is an edge of G with cost w_{ij} then arcs $\langle(i, 0), (j, -1)\rangle$ and $\langle(j, 0), (i, -1)\rangle$ are created in H , both with cost w_{ij} . A constraint guaranteeing that exactly one of these arcs is chosen is also added. The two described constraints ensure that a central edge is chosen. Zero cost arcs $\langle(j, -1), (j, 0)\rangle, j \in V$, are created. This ensures that nodes $(c, 0)$ and $(c', 0)$ belong to the solution of the STP whenever edge (c, c') is chosen as the central edge. The remaining arcs are defined as in the case for even D . Figure 5 shows a pair of solutions for $D = 5$.

Having transformed the BDMSTP to the STP as explained, any of the many tools available for the solution of the STP can be used to provide solutions for the BDMSTP. The exact method in [15] uses the *Hop-Cut* model for the STP, described in [30]. In order to shorten the description, we assume that D is even (the case for odd D is only slightly different and easily inferable). The model uses the following non-negative variables: (1) X_{0j}^1 for each arc $\langle 0, (j, 0)\rangle, j \in V$; (2) X_{ij}^k for each arc $\langle(i, k - 1), (j, k)\rangle$ in H , where $k \in \{1, \dots, D/2\}$; (3) X_{jj}^k for each arc $\langle(j, k), (j, D/2)\rangle$ in $H, j \in V$. The model also uses binary variables x_{0j} for each arc $\langle 0, (j, 0)\rangle$, and x_{ij} for each $\langle i, j\rangle \in A$, where $A = \cup_{(i,j) \in E} \{\langle i, j\rangle, \langle j, i\rangle\}$. For $S \subseteq V(H)$ with $0 \notin S$ and $S \cap R \neq \emptyset$, $[V(H) \setminus S, S]$ denotes a cut in H , and $X[V(H) \setminus S, S]$ denotes the sum of the X_{ij}^k variables associated with the arcs belonging to the cut.

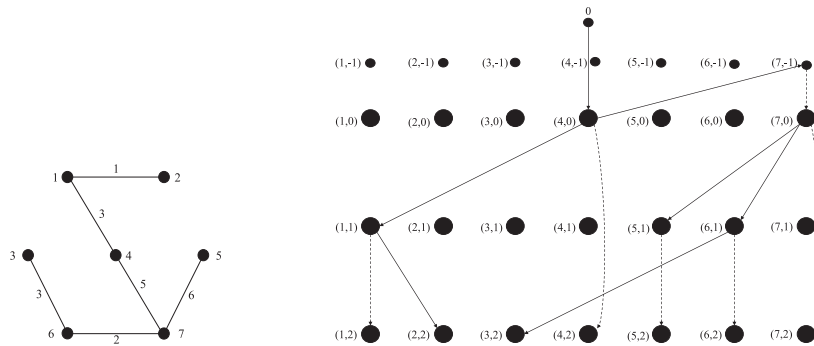


FIGURE 5. A pair of solutions of the BDMSTP and the STP, for $D = 5$. The central edge is $(4, 7)$. (Example extracted from [15].)

Hop-Cut model

(adapted for even diameter)

$$\min \sum_{\langle i,j \rangle \in A} w_{ij} x_{ij} \tag{1}$$

such that

$$X[V(H) \setminus S, S] \geq 1, \quad \forall S \subseteq V(H), 0 \notin S, S \cap R \neq \emptyset \tag{2}$$

$$x_{0j} = X_{0j}^1, \quad \forall j \in V \tag{3}$$

$$x_{ij} = \sum_{k=1}^{D/2} X_{ij}^k, \quad \forall \langle i, j \rangle \in A \tag{4}$$

$$\sum_{j \in V} x_{0j} = 1 \tag{5}$$

$$X_{ij}^k \geq 0, \quad \forall \langle i, j \rangle \in A, k \in \{1, \dots, D/2\} \tag{6}$$

$$x_{ij} \in \{0, 1\}, \quad \forall \langle i, j \rangle \in A. \tag{7}$$

The directed cut constraints (2) ensure that if $S \cap R \neq \emptyset$ then a feasible solution contains at least one arc in $[V(H) \setminus S, S]$. Constraints (3) and (4) relate X and x variables. Constraint (5) states that exactly one arc leaving node 0 is chosen.

To solve the Hop-Cut model, a branch-and-cut algorithm based on the one proposed in [9] is employed, which makes effective use of dual and primal heuristics to fix variables and speed up the convergence of the cutting plane generation. The algorithm incorporates a dual ascent heuristic to produce a good initial set of cuts to be used in the cutting plane generation. Fractional solutions are used to guide primal heuristics. All the details of the algorithm can be found in Section 4 of [15] and references therein.

4. ILS/RVND-BASED HEURISTIC FOR THE BDMSTP

ILS is a metaheuristic widely employed to solve combinatorial optimization problems, and is based on the idea that a locally optimal solution s^* obtained in the local search procedure can be improved through the application of a perturbation procedure, aiming at exploring the space of solutions near to s^* in order to refine the search and seek for a global optimum [28]. ILS has been successfully applied to several optimization problems, such as heterogeneous vehicle routing [35], multiple-travel vehicle routing [8], manufacturing cell formation [31], graph

coloring [7], and scheduling problems [10], to name just a few. The general structure of ILS is described in Algorithm 1.

Algorithm 1. General ILS structure.

```

1:  $s_0 \leftarrow \text{GENERATEINITIALSOLUTION}()$ 
2:  $s^* \leftarrow \text{LOCALSEARCH}(s_0)$ 
3: repeat
4:    $s' \leftarrow \text{PERTURBATION}(s^*, \text{history})$ 
5:    $s^{**} \leftarrow \text{LOCALSEARCH}(s')$ 
6:    $s^* \leftarrow \text{ACCEPTANCECRITERION}(s^*, s^{**}, \text{history})$ 
7: until stop condition is true

```

Procedure `GENERATEINITIALSOLUTION` determines a feasible solution s_0 to be used as a starting point for the local search. Procedure `LOCALSEARCH` explores the space of solutions seeking a local optimum. Basically, it consists of improving an initial solution s_0 through the exploration of neighboring solutions. Thus, if a neighboring solution s^* is better than s_0 then s_0 is replaced by s^* and the search is restarted from s^* . This process is repeated until no improvement is possible, *i.e.*, when s^* is a local optimum. In order to prevent the local search from remaining stagnant at a local optimum, procedure `PERTURBATION` enables different regions of the solution space to be explored. The perturbation must be strong enough to escape from a local optimum, but not too heavily, so as not to behave like a random restart [28]. In order to better explore the solution space, a history of previously visited solutions can be maintained. In the `PERTURBATION` procedure presented in Algorithm 1, s^* represents the current solution and s' is the new solution generated. Procedure `ACCEPTANCECRITERION` verifies whether the current solution s^* (line 4) will be replaced or not by a new solution s^{**} (line 5), based or not on a history of past computations. If the new solution s^{**} brings an improvement then it becomes the new current solution. The iterations of the **repeat** command cease when a prescribed stop condition is met.

4.1. Detailed specification of ILS subroutines

In this section we describe in more detail how each of the parts of the general ILS is tailored to meet our purposes.

Initial solution. In order to generate an initial solution s_0 for the local search, the constructive heuristic RGH [37] has been employed. This choice lies in the fact that RGH is easily implementable and provides good solutions.

Local search. The local search is based on a variant of the Variable Neighborhood Descent (VND) method [22], called *Random Variable Neighborhood Descent* (RVND) [43, 45]. RVND randomly explores neighborhood structures applied to the current solution, as described in Algorithm 2.

In Algorithm 2, s is the current solution of the BDMSTP, $value(s)$ stands for the cost of solution s , and $\mathcal{V} = (V_1, \dots, V_{\max})$ is a random ordering of the neighborhood structures, reset each time `LOCALSEARCH` is invoked. The algorithm goes through the neighborhood structure V_k of the current solution s , searching for a cheapest solution s' for the BDMSTP (function $BestNeighbor(s, V_k)$ in line 4). If s' is better than s then it becomes the current solution and the process is restarted ($k \leftarrow 1$). Otherwise a new neighborhood is visited ($k \leftarrow k + 1$). The process is repeated until no improvement is possible, that is, when $k = \max$.

The choice of RVND over VND is justified by the fact that the main objective of this work is the analysis of neighborhoods, and thus it is not desired to determine an ordering of neighborhood selection preferences, which is what VND does. The RVND algorithm, with its random choices, gives every neighborhood the same probability of being selected in a certain position in the ordering, eliminating the bias of a fixed ordering and preventing some neighborhoods from being executed more often than others.

Algorithm 2. Procedure LOCALSEARCH (s).

```

1: randomly set an ordering  $\mathcal{V} = (V_1, \dots, V_{\max})$  of the neighborhood structures
2:  $k \leftarrow 1$ 
3: repeat
4:    $s' \leftarrow \text{BestNeighbor}(s, V_k)$ 
5:   if  $\text{value}(s') < \text{value}(s)$  then
6:      $s \leftarrow s'$ 
7:      $k \leftarrow 1$ 
8:   else
9:      $k \leftarrow k + 1$ 
10:  end if
11: until  $k = \max$ 
12: return  $s$ 

```

Neighborhood structures. The neighborhood structures used in the implementation of the local search are *Edge Exchange* [17], *Node Swap* [17], *Subtree Optimize* [37], and five new neighborhoods introduced in this work: *Hierarchy Exchange*, *Hierarchy Rotation*, *Leaf Reallocation*, *Parent Swap*, and *Level Change*.

In all cases, an n -node tree rooted at a single center represents a feasible solution of the BDMSTP, if the diameter is even, or at two centers, if the diameter is odd. The remaining nodes cannot be at a depth greater than $\lfloor D/2 \rfloor$ [20]. Besides, if the diameter is odd, there exists an edge that connects the two central nodes. The tree that represents a feasible solution of the BDMSTP is oriented from the center (or centers) towards the leaves, defining a relation of predecessors and successors. The center (or centers) have depth zero, while the leaves have depth at most $\lfloor D/2 \rfloor$. This form of representing a solution based on node levels is also used in [14] to solve the Hop-Constrained Minimum Spanning Tree Problem (HMSTP).

As explained in [17], most local moves for the BDMSTP can be broadly classified into two groups: *tree-structure-based neighborhoods*, defined by the relationship between predecessors and successors in the tree; and *level-based neighborhoods*, where not only the predecessor/successor information is relevant, but also the level a node is assigned to is of interest. The former group includes *Edge Exchange* [17], *Node Swap* [17], *Hierarchy Exchange*, *Hierarchy Rotation*, and *Parent Swap*, while the latter includes *Subtree Optimize* [37], *Leaf Reallocation*, and *Level Change*. Some neighborhoods described in [14] (*Shift*, *Swap*, and *Shift-Swap* neighborhoods), although conceived to solve the HMSTP, can be directly applied to the BDMSTP and can also be classified into the latter group.

Edge Exchange. This neighborhood is described in [17], and works as follows. Let T be a BDST. The Edge Exchange neighborhood consists of exploring all possible trees obtained from disconnecting a subtree of T (by removing an edge) and connecting it into another position of the tree, provided that the cost is reduced and the diameter is not violated. All possible subtrees are examined, aiming at determining the connection of least cost. Consider the tree of Figure 6a. The disconnection of edge (b, c) produces two subtrees T_p (“primary tree”) and T_s (“secondary tree”), as seen in Figure 6b. Subsequently, the algorithm verifies whether creating an edge linking the root node c of T_s with a node of T_p reduces the cost of T . Suppose that the edge that produces the least cost among all the possibilities is (g, c) ; this leads to the new tree presented in Figure 6c. The time to examine all the edges of T is $O(n)$. The time to examine all possible replacements in order to identify the least cost is also $O(n)$, which results in an $O(n^2)$ total time for this neighborhood.

Node Swap. This neighborhood is also presented in [17], and explores the relation between a tree node, including its center (or one of the centers, if the diameter is odd), and its direct successors. A solution for this neighborhood is a tree resulting from replacing a node v by one of its successors, say, node u . In such case, the resulting tree of this operation is obtained by making v and its successors (other than u) successors of u , and the predecessor of v is made the predecessor of u . If v is a center, then u does not have a predecessor. Let T be the tree illustrated in Figure 7a. Assume $v = b$ and $u = e$. According to Figure 7b, the operation makes node a the direct predecessor

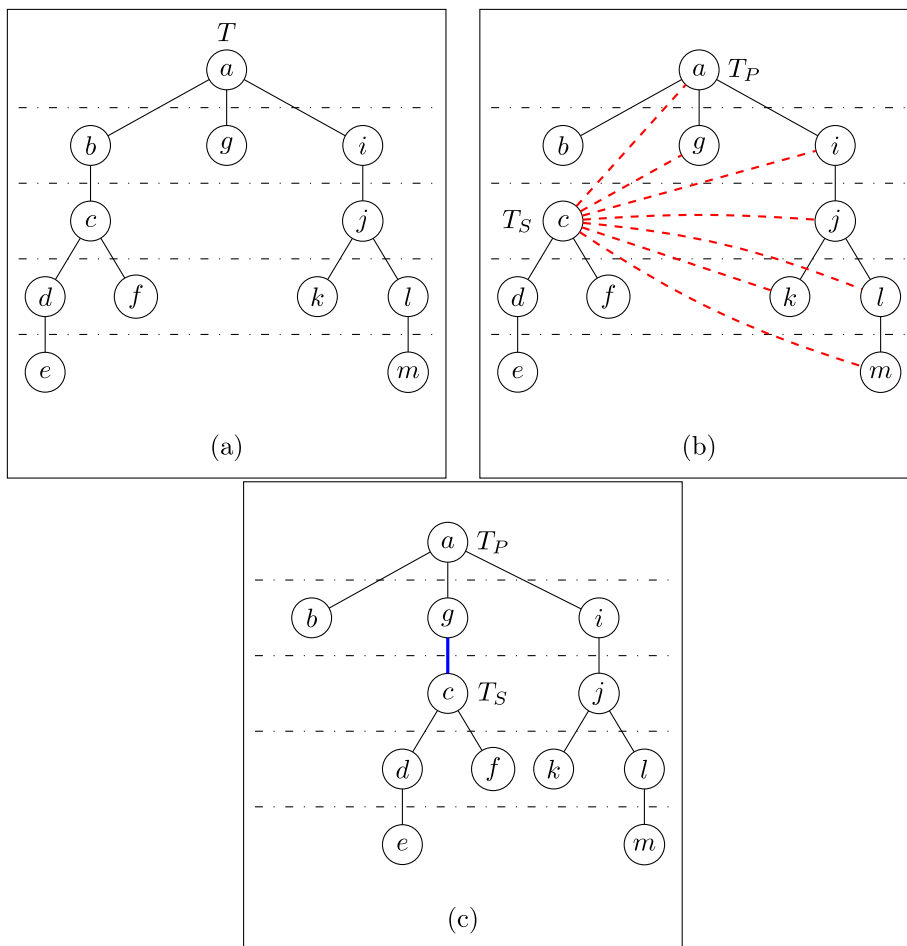


FIGURE 6. Edge Exchange example.

of e , while b and all of its successors (other than e) are made successors of e , as illustrated by Figure 7c. The time to verify all pairs (v, u) of T is $O(n)$. The time to verify all successors of a node v is the degree $d(v)$ of v . Thus, the total time spent by this neighborhood is $O(n\Delta)$, where Δ is the maximum degree of a node.

Subtree Optimize. The Subtree Optimize neighborhood was described in [37], and explores the relation between sibling nodes with depth $\lfloor D/2 \rfloor$ and their direct predecessor v , whose depth is $\lfloor D/2 \rfloor - 1$. Basically, this neighborhood rearranges the subtree rooted at v in a suitable way. Let S be the set of nodes in this subtree, and p the direct predecessor of v . The operation tries, for each $u \in S \setminus \{v\}$, to set u as the subtree root, connecting it to p . Also, v takes the place occupied by u before the change. The cost of the new subtree T_u with root u is easily given by equation (8).

$$w(T_u) = w_{pu} + \sum_{x \in S \setminus \{u\}} w_{ux}. \tag{8}$$

The subtree that minimizes this cost is chosen. If no cost improvement is achieved then no local rearrangement is done. Observe that this operation does not violate the diameter. Let T be the tree in Figure 8a. The operation selects node f , whose depth is $\lfloor D/2 \rfloor - 1$. Next, each successor of f (in this case, g or h) is placed at the root of the new subtree and the cost is verified, as shown in Figures 8b and 8c. Figure 8d shows an example, where the

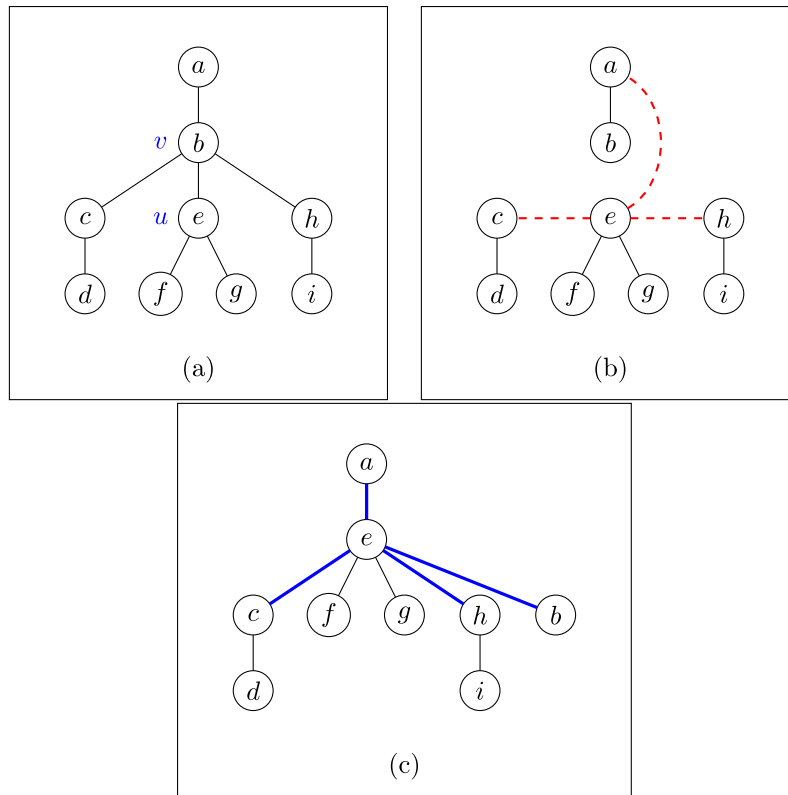


FIGURE 7. Node Swap example.

commutation of nodes g and f produces a resulting tree of lower cost. For a fixed node v , determining the cost of a subtree configuration requires $O(|S|)$ time, where $|S| = d(v) + 1$. In addition, there are $O(d(v))$ possible configurations to be examined. Thus, the complexity of this operation is $O(d(v)^2)$.

Hierarchy Exchange. This neighborhood explores the gradual inversion in the hierarchical order of a node v and its successors. The operation is applied to all nodes of the BDST until finding a resulting tree that minimizes the cost of the whole operation. It is described as follows. Let T be a BDST, and let $p, v, u_1, u_2, \dots, u_{k-1}, u_k$ be nodes of T such that v is the root of a subtree, p the direct predecessor of v , u_1 a direct successor of v , u_2 a direct successor of u_1 , and so on. Assume that the cost of edge (p, u_k) is lower than the cost of edge (p, v) . Thus, the operation makes u_k a direct successor of p , u_{k-1} a direct successor of u_k , u_{k-2} a direct successor of u_{k-1} , and so on until v (which is made a direct successor of u_1). If after this process some node violates the diameter then it will be disconnected from the tree and reconnected to another position using an edge of minimum cost. Figure 9a shows a BDST T . Nodes j and n are chosen to start the operation. Next, j becomes a direct successor of n , and n becomes a direct successor of i , as shown in Figure 9b. Suppose that this operation does not reduce the cost of the tree; then node o , which is a direct successor of n , is chosen and made a direct successor of i , while n becomes a direct successor of o , as illustrated in Figure 9c. At this point, suppose that the cost of the tree is lower; thus nodes that violate the diameter, in the example l and m , must be reconnected to other positions using edges of minimum cost, which are (g, l) and (f, m) , respectively, as illustrated in Figure 9d. Each operation of hierarchy inversion between predecessors and successors clearly takes $O(n)$ time. In the end, if the diameter of a node is violated, the time to reconnect it to another place of the tree is $O(n)$; in the worst case, $O(n)$ nodes need to be reconnected, yielding an $O(n^2)$ time complexity for inversion plus redistribution

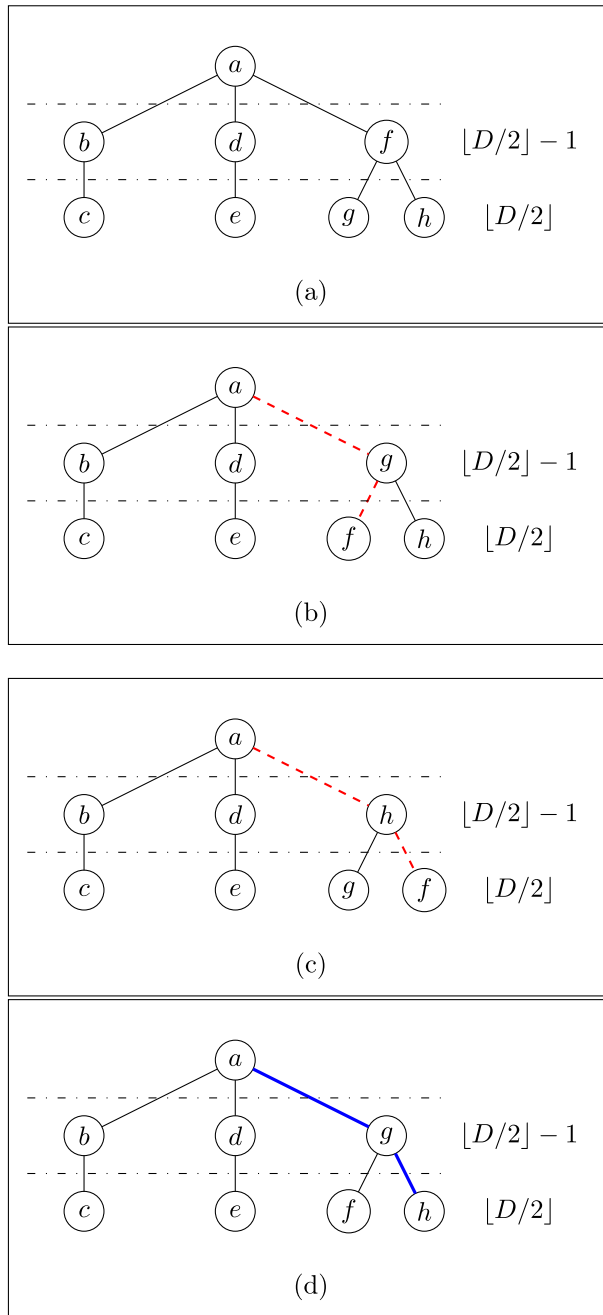


FIGURE 8. Subtree Optimize example.

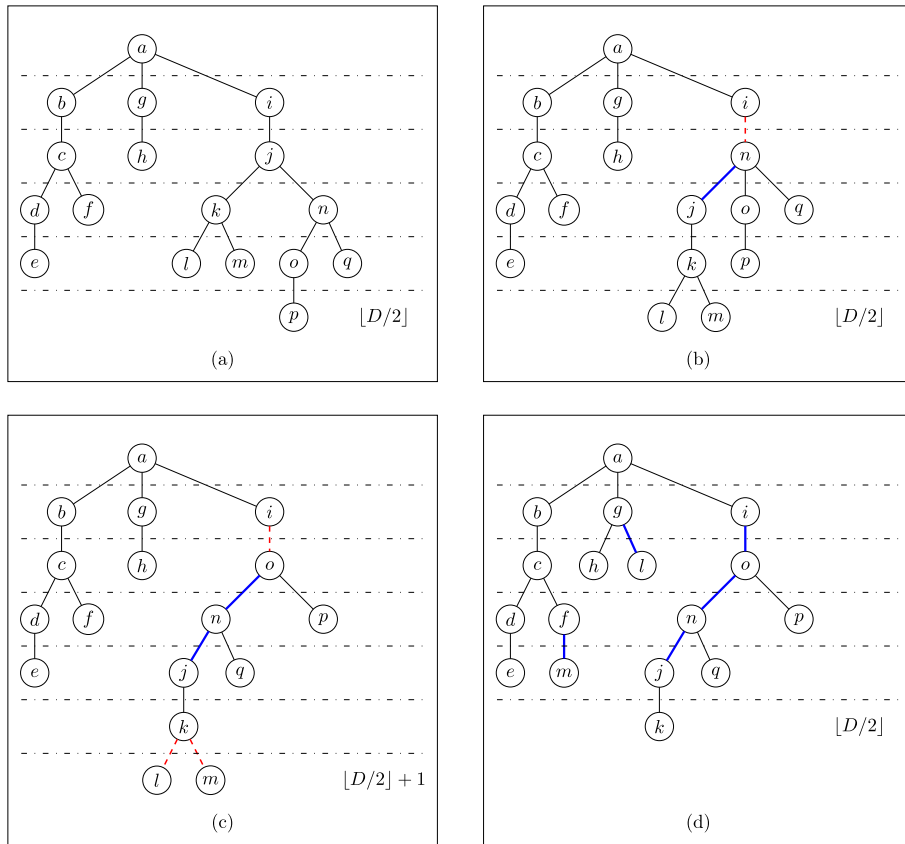


FIGURE 9. Hierarchy Exchange example.

of nodes. Since there are $O(n)$ nodes that can be the starting point of an operation of hierarchy inversion, the overall complexity of this neighborhood is $O(n^3)$.

Hierarchy Rotation. This neighborhood is similar to the previous one, but explores the gradual inversion in the hierarchy order of a node v node and its predecessors, and the distribution of the resulting subtree. Again, the operation is applied to all nodes of the BDST until it finds a resulting tree that minimizes the cost of the operation. It is described as follows. Let T be a BDST, with center c (or one of the centers, if the diameter is odd). Let v be the root of a subtree of T and $u_1, u_2, \dots, u_{k-1}, u_k$ be a path from v to c in T (u_1 is the direct predecessor of v and u_k is the direct successor of c in this path). The operation begins by making u_1 a direct successor of v and removing edge (u_1, u_2) . The resulting subtree, whose root is v , is reconnected to another position of the tree, provided that the cost is decreased and the diameter is not violated. If the cost of the resulting BDST does not decrease, then edge (u_2, u_3) is removed and u_2 becomes a direct successor of u_1 , using the edge (u_1, u_2) . Again, the resulting subtree, whose root is v , is reconnected to another position of the tree, provided that the cost decreases and the diameter is not violated. This process is repeated until node u_k is analyzed; then edge (c, u_k) is removed and u_k is made a direct successor of u_{k-1} , using the edge (u_{k-1}, u_k) .

Figure 10a shows a BDST T . First, node o is chosen to start the operation; node n becomes a direct successor of o and edge (m, n) is removed. Next, the operation checks whether connecting the subtree rooted at o to another position of the BDST decreases the cost, as Figure 10b shows. If the cost of the BDST does not decrease, then the operation is continued and node m , former predecessor of n , becomes a direct successor of

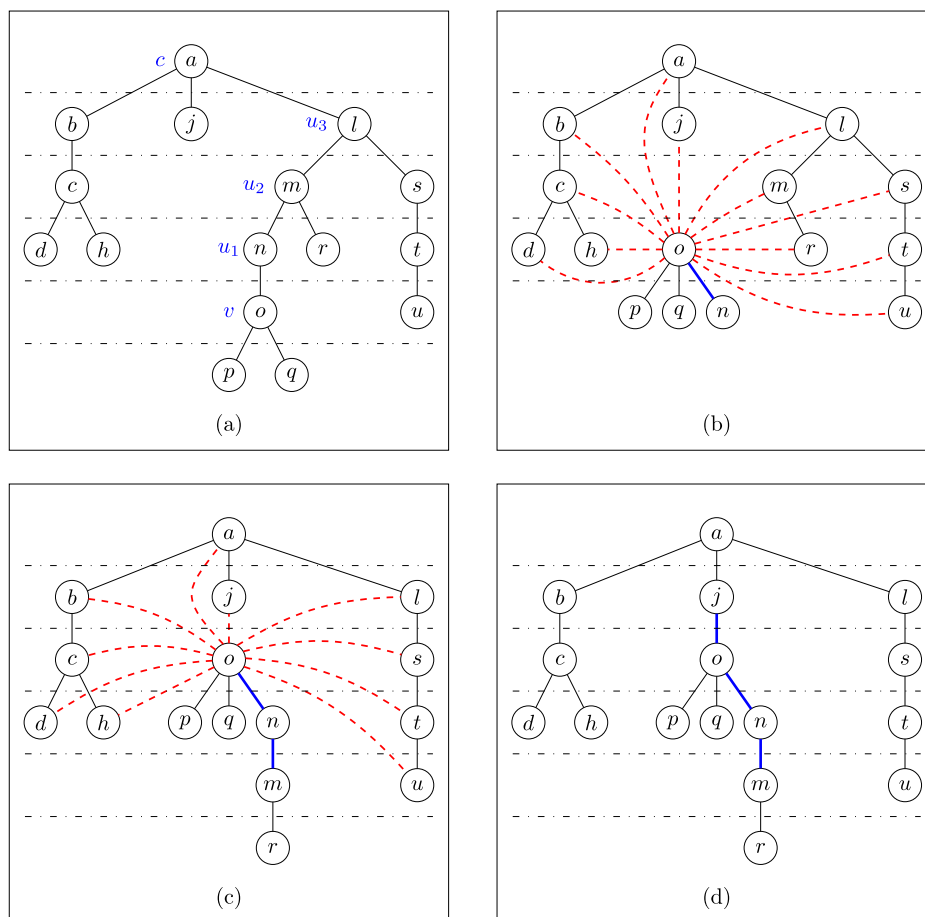


FIGURE 10. Hierarchy Rotation example.

n , as shown in Figure 10c. Again, the resulting subtree with root o is tested in all possible positions (Fig. 10c), aiming at decreasing the cost (provided that the diameter is not violated). According to Figure 10d, by making j the direct predecessor of o , suppose that the cost of the BDST is decreased and the diameter is not violated. Then o is made a direct successor of j , using the edge (j, o) , and the process finishes at this point. There are $O(n)$ possible nodes to start a single operation of hierarchy rotation. Each operation may involve $O(n)$ inversions of predecessor/successor pairs. In turn, each inversion needs $O(n)$ subtree reconnection tests in order to decrease the cost. Thus, the total cost of this neighborhood is $O(n^3)$.

Leaf Reallocation. This neighborhood explores the reallocation of a leaf throughout the BDST, aiming at decreasing the cost. Let T be the BDST of Figure 11a. The operation selects a leaf of T , in this case, node e . Next, the operation verifies whether putting node e as a direct successor of other nodes in the tree will reduce its cost, as illustrated in Figure 11b. Suppose that making node e a direct successor of node f reduces the cost of the tree. The resulting BDST is depicted in Figure 11c. We remark that this move is carried out provided that the diameter of the BDST is not violated. Exploring all the leaves takes $O(n)$ time, while testing a single leaf at other positions of the tree takes $O(n)$ time overall as well. This results in an $O(n^2)$ time for this neighborhood.

Parent Swap. Parent Swap tests whether moving a subtree with root v containing a single leaf u as its direct successor to another position of the tree reduces its cost. If the cost of the resulting tree does not decrease, the

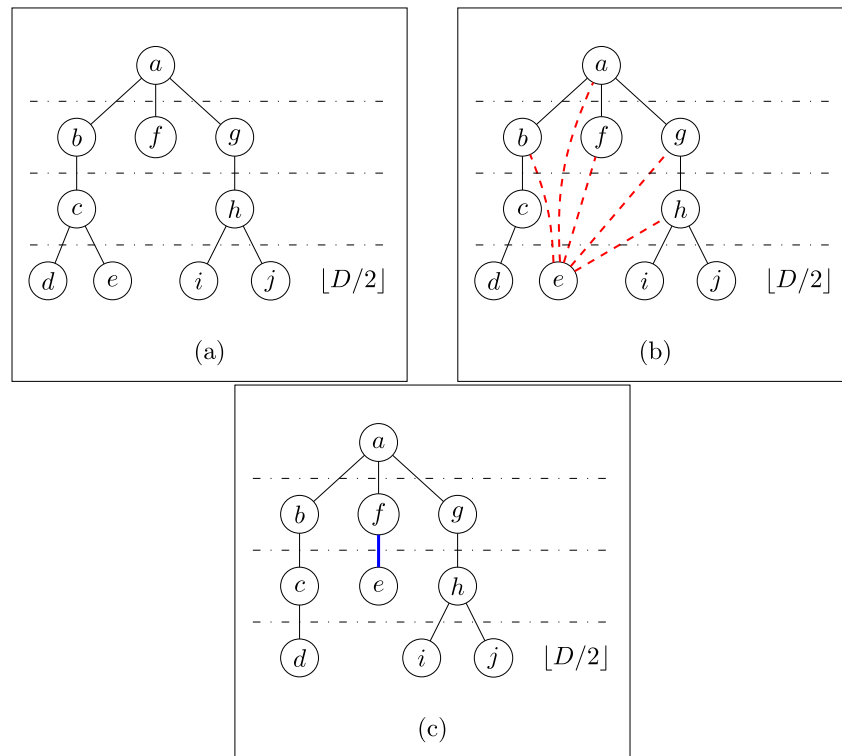


FIGURE 11. Leaf Reallocation example.

operation inverts the relation between nodes v and u (i.e., u becomes the new root of the subtree and v its direct successor), and repeats the same test, trying to move the new subtree to another position of the tree in order to reduce its cost. In each operation the diameter should not be violated. Let T be the BDST of Figure 12a. First, the operation selects node d , since it has a single leaf node as a direct successor. Next, the operation tests whether the subtree rooted at d can be moved to another position of the tree in order to reduce its cost, as shown in Figure 12b. If this does not occur, node e becomes the direct predecessor of d , and d becomes the direct successor of e (see Fig. 12c). Then, the same test illustrated in Figure 12b is performed, but now for the subtree having e as its root (Fig. 12c). Suppose that making node e a direct successor of f reduces the cost of the tree; then the process finishes, and Figure 12d shows the resulting BDST. Exploring all the nodes of T having only a single leaf as a direct successor takes $O(n)$ time. Verifying whether the connection of a single selected node to another position of the tree reduces its cost takes $O(n)$ time. The inversion operation between a node and its parent needs constant time, while verifying whether moving the resulting subtree from this inversion to another position of the tree in order to reduce the cost also takes $O(n)$ time. Hence, this neighborhood takes $O(n^2)$ time.

Level Change. Level Change verifies whether moving a subtree rooted at a node v lying in level k to another position in the tree such that v is made a direct successor of a node u at level $l \geq k$ reduces the cost of the tree and does not violate the diameter. Let T be the BDST of Figure 13a. The operation selects node i (at level three) as the subtree root. Next, the operation verifies whether connecting i as a direct successor of a node at level three (d , e , m , or o) or a node at level four (node p) reduces the cost of the tree (see Fig. 13b). In the example, suppose that connecting the subtree to node e reduces the cost of the tree. Then, node e is selected to be the direct predecessor of i , and the resulting tree is indicated in Figure 13c. There are $O(n)$ possible subtrees

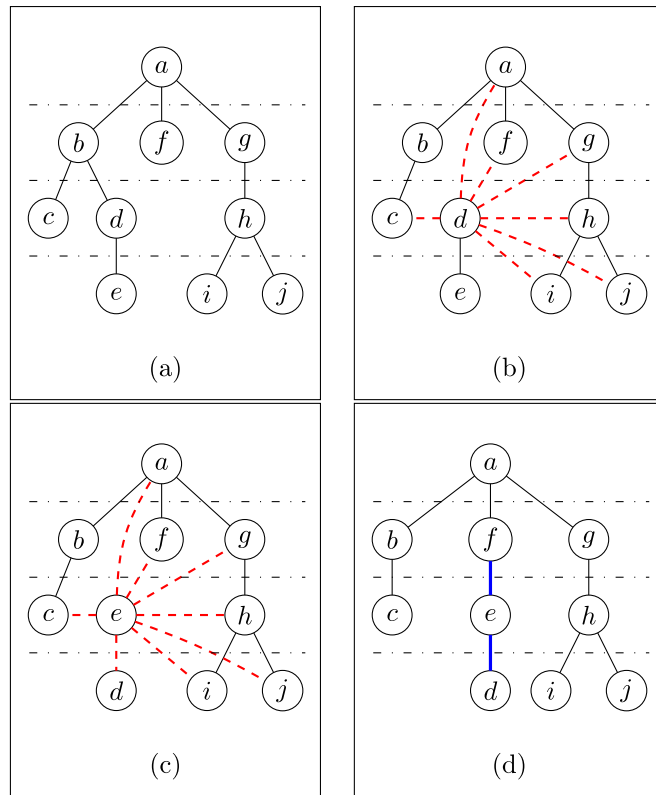


FIGURE 12. Parent Swap example.

(roots) to be selected. For a single selected subtree, the time to verify whether connecting it to nodes at the same level or higher levels reduces the cost is $O(n)$, while the connection of the subtree takes constant time. Thus, this neighborhood takes $O(n^2)$ time.

Perturbation. The proposed perturbation procedure employs four perturbation moves from the literature: (a) *Edge-Exchange Move*, a variant of *Edge Exchange* where the choice of the subtree root to be used in the operation is random; (b) *Node-Swap Move*, a variant of *Node Swap* that uses the same strategy above (a random choice of the subtree used); (c) *Edge-Delete Move*, a variant of *Edge Delete* [37]; (d) *Center-Change Move*, a variant of *Center Exchange* [37].

The choice of which perturbation move (EE, NS, ED, or CC) will be applied in each perturbation call is made randomly. Preliminary tests indicated that this approach is able to produce perturbed solutions with a good level of diversity; moreover, it contributed to improving computational times, as these neighborhoods are fast to execute. The history of previous solutions is not considered.

Below, we describe in more detail *Edge-Delete Move* and *Center-Change Move*.

Edge-Delete Move. Let T be the BDST of Figure 14a. This move randomly removes an edge of T , forming two subtrees T_p and T_s , as shown in Figure 14b, where edge (i, j) was removed. Next, all nodes of T_s are disconnected and individually reconnected to other places of T_p using edges of minimum cost; see Figure 14c. The time to randomly select one node of T is constant. The subtree T_s may contain $O(n)$ nodes, each requiring an $O(n)$ time to be reconnected at some position of T_p . This results in an $O(n^2)$ total time for this move.

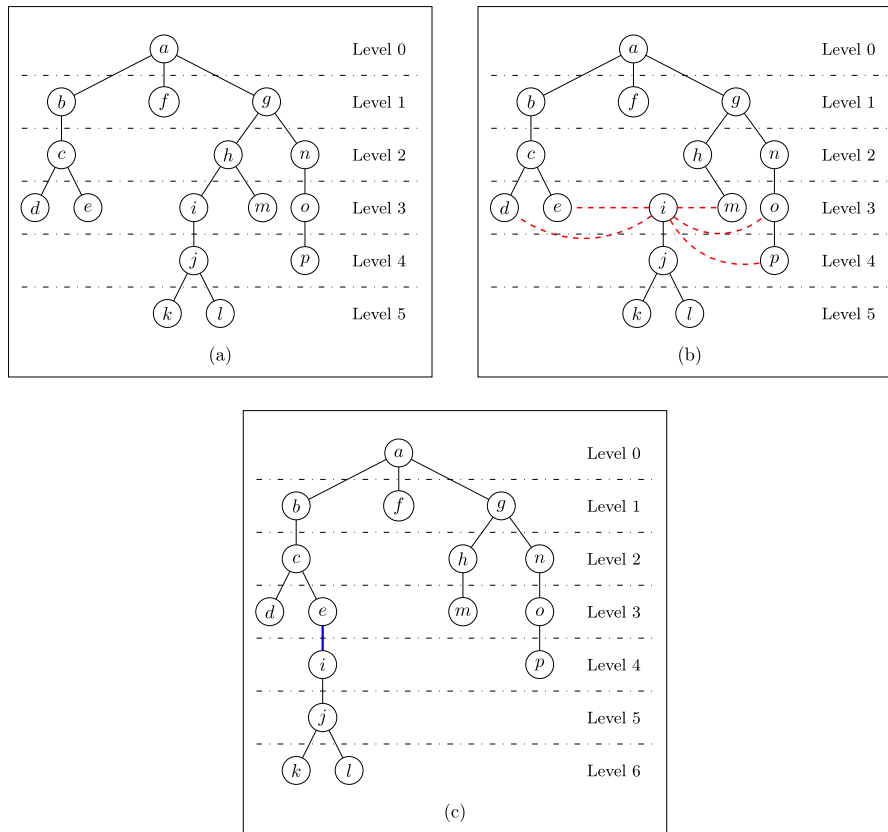


FIGURE 13. Level Change example.

Center-Change Move. Let T be a BDST. Center-Change Move randomly selects a node of T (in Fig. 15a, suppose that c is the selected node) and makes it the new center, if the diameter is even, or one of the centers, if the diameter is odd. The successors of the old center (in the example, node a) become the direct successors of c , while node a becomes a direct successor of a randomly chosen node (in the example, according to Fig. 15b, node a becomes the successor of node b). The time to randomly select a node v of T is constant. The time to make v the new center of the tree is constant, while making the successors of the old center u successors of v is $O(k)$, where k is the number of direct successors of u . The time to connect the old center u as a direct successor of a randomly chosen node is constant. Therefore, the total time for this move is $O(\Delta)$, where Δ is the maximum degree of a node in the tree.

Acceptance criterion. The acceptance criterion adopted is simply the comparison of the costs of solutions s^* and s^{**} (line 6 of Algorithm 1), returning the one with lower cost. The history of previous solutions is not considered, as in many ILS implementations.

The stop condition in line 7 of Algorithm 1 will be discussed in the next section.

5. COMPUTATIONAL RESULTS

5.1. Experiments with 50- and 100-node instances

We applied the exact method presented in [15], described in Section 3, to 50- and 100-node instances of the OR-Library [4]. The method has been applied to two datasets widely explored in the BDMSTP literature: the

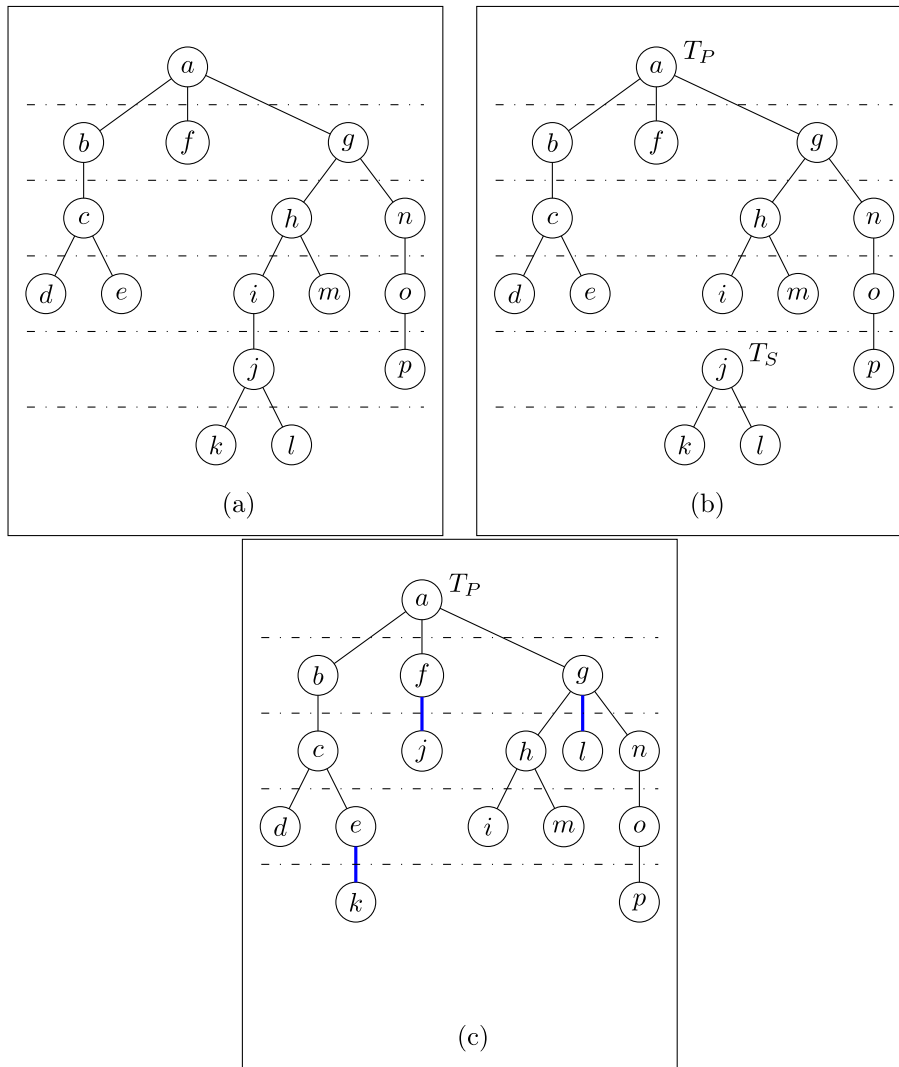


FIGURE 14. Edge-Delete Move example.

first consists of the first five OR-Library instances with 50 nodes (diameter set to 5), and the second of the first five OR-Library instances with 100 nodes (diameter set to 10). The experiments of the exact method were conducted on an Intel®Core™i9-8950hk 2.9 GHz CPU with 32 GB of RAM, running under the Linux operating system Ubuntu version 20.04 LTS. We implemented this method using the callable routines in Xpress Solver 8.6. All Xpress parameters have been kept at their default values. The choice to use this machine is justified by the fact that it has been configured to be entirely dedicated to running exact methods using Xpress.

We compared the optimal solutions obtained by the exact method with the best heuristic results found in the literature. See Table 1. In the table, n is the number of nodes, id the identification of the instance, d the diameter, opt the value of the optimal solution, t the time spent by the exact method (in seconds), $heur$ the best heuristic value found in the literature (with the corresponding reference), and gap is the relative gain obtained by the exact method, calculated by Equation 9.

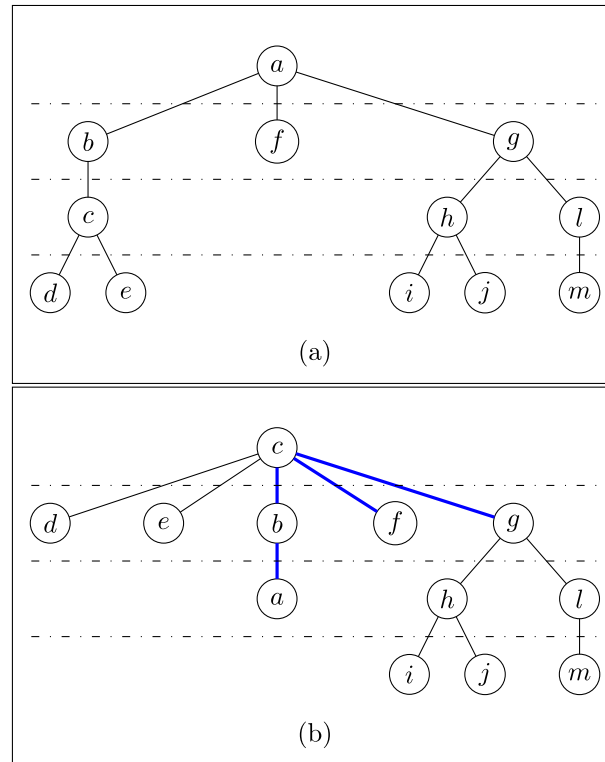


FIGURE 15. Center-Change Move example.

$$gap = \frac{heur - opt}{opt} \times 100 \quad (\%). \quad (9)$$

Results in Table 1 show that no heuristic method was able to obtain optimal solutions of the BDMSTP for the instances listed in the table, despite all the research efforts in the 2000s. This means that, for such instances, it is very difficult for heuristic methods to escape from local optima.

The exact method was unable to reach optimal solutions or lower bounds for instances of the OR-Library with 250 nodes onwards, even in 21 600s of execution.

For the sake of presenting complete results on small instances, in Table 2 we compare the best heuristic results of the literature (column *heur* in Tab. 1) with the ILS/RVND approach presented in this work, hereafter simply called ILS/RVND. The experiments were run on Microsoft[®] Azure[®], using a performance-oriented architecture (“Standard.F4s.v2”), with 4 processor cores and 8 GB of RAM. According to the platform documentation³, this architecture is based on Intel[®] Xeon[®] Platinum 8168 @ 2.70 GHz. All the routines of ILS/RVND were implemented in Java 7.

The meaning of the columns in Table 2 is the following: *n* is the number of nodes, *id* is the identification of the instance, *d* is its diameter, *literature* is the method that obtains the best result known in the literature, *best* is the best solution value obtained by the method indicated in the previous column, *mean* is the average solution value obtained over several runs, *t* is the average time spent over the runs (in seconds), *best** is the best solution value obtained by ILS/RVND, *mean** is the average solution value obtained over 50 runs of ILS/RVND, and *t** is the running time of each run of ILS/RVND (the stopping condition for each run is set to 1 second for 50-node

³ <https://docs.microsoft.com/en-us/azure/virtual-machines/fsv2-series> – Last accessed on September 25, 2021.

TABLE 1. Optimal values obtained by the exact method, applied to 50- and 100-node OR-Library Euclidean instances. In the table, n is the number of nodes, id the identification of the instance, d the diameter, opt the value of the optimal solution, t the time spent by the exact method (in seconds), $heur$ the best heuristic value found in the literature (with the corresponding reference), and gap is the relative gain obtained by the exact method.

n	id	d	opt	t (s)	$heur$	gap
50	1	5	7.31	0.27	7.60 [29]	3.97%
50	2	5	7.37	0.81	7.61 [29]	3.26%
50	3	5	6.99	1.23	7.24 [29]	3.58%
50	4	5	6.32	0.75	6.59 [29]	4.27%
50	5	5	7.04	0.86	7.25 [29]	2.98%
100	1	10	7.25	9629.38	7.76 [19]	7.03%
100	2	10	7.33	1801.26	7.85 [19]	7.09%
100	3	10	7.43	2055.01	7.90 [17]	6.33%
100	4	10	7.46	1920.35	7.98 [19]	6.97%
100	5	10	7.63	18405.90	8.16 [19]	6.95%

TABLE 2. Comparison between ILS/RVND and the best results found in the literature for 50- and 100-node instances. In the table, n is the number of nodes, id is the identification of the instance, d is its diameter, $literature$ is the method that obtains the best result known in the literature, $best$ is the best solution value obtained by the method indicated in the previous column, $mean$ is the average solution value obtained over several runs, t is the average time spent over the runs (in seconds), $best^*$ is the best solution value obtained by ILS/RVND, $mean^*$ is the average solution value obtained over 50 runs of ILS/RVND, and t^* is the running time of each run of ILS/RVND.

n	id	d	$literature$	$best$	$mean$	$t(s)$	$best^*$	$mean^*$	$t^*(s)$
50	1	5	GILS [29]	7.60	7.60	10.00	7.60	7.61	1.00
50	2	5	GILS [29]	7.61	7.62	10.00	7.61	7.61	1.00
50	3	5	GILS [29]	7.24	7.24	10.00	7.24	7.26	1.00
50	4	5	GILS [29]	6.59	6.59	10.00	6.59	6.59	1.00
50	5	5	GILS [29]	7.25	7.25	10.00	7.25	7.25	1.00
100	1	10	ACO [19]	7.76	7.77	27.78	7.76	7.82	10.00
100	2	10	LEEA [19]	7.85	7.86	734.65	7.85	7.88	10.00
100	3	10	VNS [17]	7.90	7.96	38.66	7.90	7.94	10.00
100	4	10	LEEA [19]	7.98	8.09	732.83	7.98	8.00	10.00
100	5	10	ACO [19]	8.16	8.17	24.45	8.16	8.20	10.00

instances and 10 seconds for 100-node instances). Table 2 shows that ILS/RVND performs competitively with the best methods in the literature for small instances.

5.2. Experiments with 250-, 500-, and 1000-node instances

After solving the easier 50- and 100-node cases in the preceding section, we now apply ILS/RVND approach to Euclidean instances of the OR-Library with 250, 500, and 1000 nodes. For the tests, we used three datasets: (1) the first five instances with $n = 250$ (diameter 15); (2) the first five instances with $n = 500$ (diameter 20); (3) the first five instances with $n = 1000$ (diameter 25).

TABLE 3. Five versions of ILS/RVND.

ILS/RVND version	Neighborhoods used							
	EE	NS	SO	HE	HR	LR	PS	LC
ILS-8	x	x	x	x	x	x	x	x
ILS-6a	x	x		x	x	x		x
ILS-6b		x	x	x	x	x		x
ILS-4		x		x	x	x		
ILS-3	x	x	x					

As in the previous subsection, the experiments were performed on Microsoft[®] Azure[®], using the performance-oriented architecture “Standard_F4s_v2”, based on Intel[®] Xeon[®] Platinum 8168 @ 2.70 GHz with 4 processor cores and 8 GB of RAM. Likewise, the routines of ILS/RVND were implemented in Java 7.

We divide the computational experiments into two parts. In the first part, we conduct an experiment to compare the eight neighborhoods described in Section 4.1. In the second part, we compare the results obtained by ILS/RVND with the results from the literature.

5.2.1. Comparing the neighborhoods

Regarding the neighborhoods described in Section 4.1, we use the following abbreviations: EE – Edge Exchange; NS – Node Swap; SO – Subtree Optimize; HE – Hierarchy Exchange; HR – Hierarchy Rotation; LR – Leaf Reallocation; PS – Parent Swap; LC – Level Change.

Figure 16 shows the results of the application of the ILS/RVND to the five 250-node instances. We performed 50 runs for each instance (hence 250 runs overall), where each run uses the following stop condition: 1000 iterations of the ILS/RVND main loop with no improvement in solution value. For each neighborhood in Figure 16, the k th bar from left to right shows the number of times, over all the 250 runs, the neighborhood was able to improve the current solution (*i.e.*, the number of times the test in line 5 of Algorithm 2 is true) when it is selected as the k th ($1 \leq k \leq 8$) neighborhood in the random ordering of the local search.

Let $ni(V, k)$ be the number of improvements achieved by neighborhood V when it is selected as the k th neighborhood in the random ordering of the local search, according to Figure 16. For every $V \in \{\text{EE, NS, SO, HE, HR, LR, PS, LC}\}$ and $k < 8$, we observe that $ni(V, k) > ni(V, k + 1)$, with a similar decay for all the neighborhoods (this behavior is virtually identical for the 500- and 1000-node instances, and thus we omit here the graphical results for them). As expected, for a single execution of the local search, the position occupied by a neighborhood in the random ordering directly influences its *effectiveness* (number of times it contributes to improving the current solution). Since the RVND strategy does not prioritize any neighborhood over the others along several executions of the local search (unlike the usual VND method), we can easily calculate the *absolute effectiveness* of each neighborhood, determined regardless of the positions it occupied along the executions of Algorithm 2. For each neighborhood V , its absolute effectiveness is given by $\sum_{k=1}^8 ni(V, k)$. See Figure 17.

Figure 17 indicates that SO and PS are the least effective neighborhoods. This is also valid for 500- and 1000-node instances. An interesting question is therefore to create other versions of the local search, by choosing some suitable subsets of neighborhoods. In Table 3, we show five versions of ILS/RVND, chosen for a comparative study: ILS-8 (the original proposal, using the eight neighborhoods described in Sect. 4.1); ILS-6a (that discards the neighborhoods PS and SO); ILS-6b (that uses a subset of six neighborhoods selected by the IRACE package [27], as described below); ILS-4 (that uses a subset of four neighborhoods selected by IRACE); and ILS-3 (which uses only the neighborhoods previously known in the literature [17, 37]).

Use of the IRACE package. The IRACE package was used as an auxiliary tool to conduct a comparative study to evaluate the performance, in terms of solution quality, of some subsets of neighborhoods. The computational cost of the neighborhoods was not considered in the study, in order to focus IRACE’s work on selecting

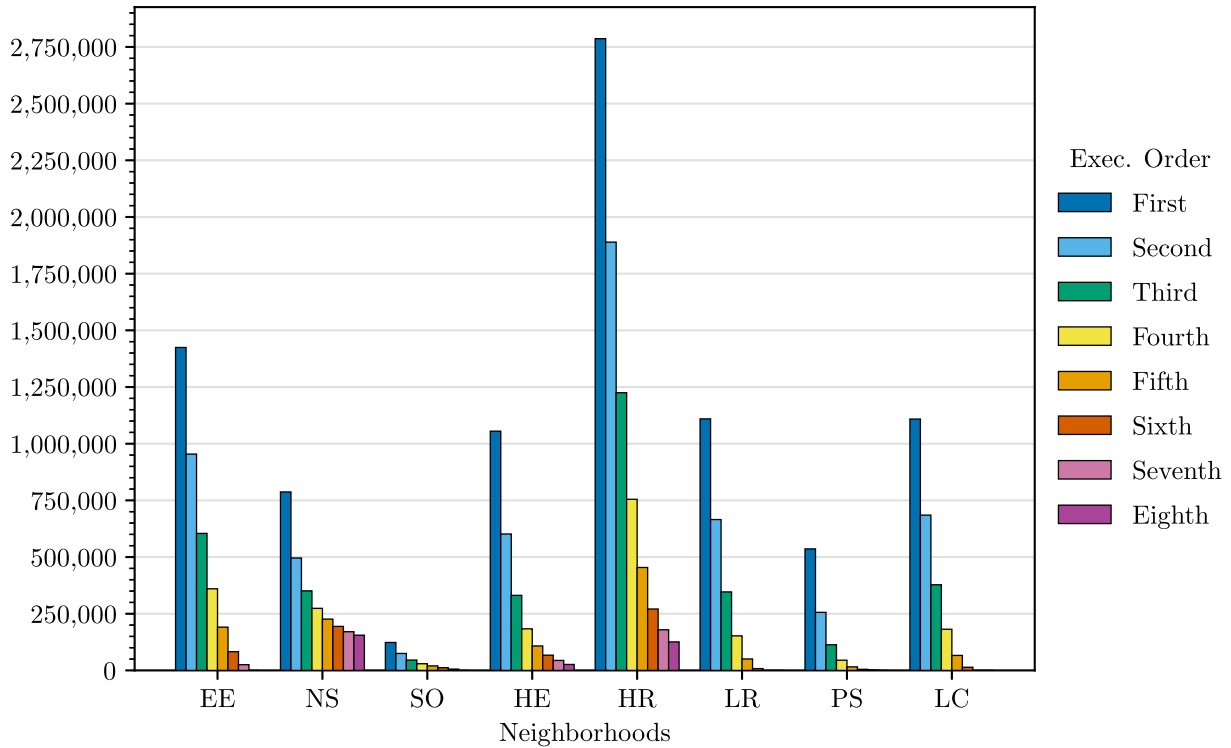


FIGURE 16. Number of solution improvements for each neighborhood according to its position in the random ordering of the local search. The number of improvements is counted over all the 250 runs on the 250-node instances.

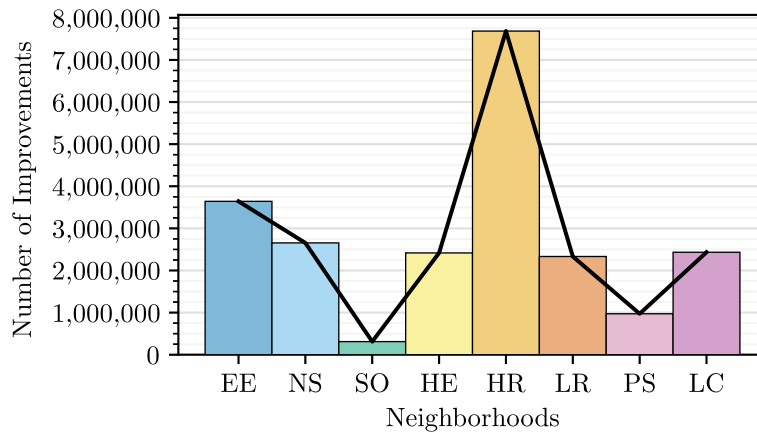


FIGURE 17. Absolute effectiveness of each neighborhood over all the 250 runs on the 250-node instances. From the most to the least effective, we have the following order of neighborhoods: HR – EE – NS – LC – HE – LR – PS – SO. Note that LC, HE, and LR have a similar effectiveness.

those neighborhoods that contribute most to improve solution quality. In IRACE configuration settings, each neighborhood is simply modeled as a binary variable with values *enabled/disabled*.

TABLE 4. Results obtained by the five ILS versions listed in Table 3, for the 250-, 500-, and 1000-node instances.

<i>n</i>	<i>id</i>	ILS-8			ILS-6a			ILS-6b			ILS-4			ILS-3		
		<i>best</i>	<i>mean</i>	<i>time</i>	<i>best</i>	<i>mean</i>	<i>time</i>	<i>best</i>	<i>mean</i>	<i>time</i>	<i>best</i>	<i>mean</i>	<i>time</i>	<i>best</i>	<i>mean</i>	<i>time</i>
250	1	12.17	12.27	85.69	12.17	12.28	83.73	12.19	12.28	98.18	12.17	12.27	109.34	12.34	12.45	33.51
250	2	12.01	12.14	62.75	12.02	12.15	64.80	12.00	12.11	79.06	12.03	12.11	92.27	12.11	12.30	31.94
250	3	11.96	12.05	58.57	11.97	12.04	56.27	11.96	12.03	72.18	11.96	12.02	79.18	12.04	12.17	29.28
250	4	12.41	12.51	72.56	12.44	12.52	67.41	12.41	12.51	88.34	12.40	12.48	100.15	12.54	12.70	33.15
250	5	12.20	12.29	80.84	12.20	12.31	67.96	12.21	12.31	82.07	12.21	12.30	94.73	12.24	12.48	30.82
500	1	16.74	16.91	698.42	16.74	16.91	790.89	16.74	16.91	866.47	16.79	16.90	917.52	16.92	17.15	321.13
500	2	16.61	16.78	681.90	16.65	16.76	767.10	16.65	16.79	791.33	16.65	16.75	832.79	16.78	17.04	323.15
500	3	16.77	16.87	755.77	16.76	16.89	773.58	16.76	16.86	983.38	16.73	16.86	1014.10	16.89	17.12	335.20
500	4	16.76	16.92	752.34	16.77	16.93	633.19	16.78	16.91	792.84	16.78	16.89	890.19	16.99	17.18	315.51
500	5	16.40	16.54	748.18	16.39	16.52	665.08	16.37	16.54	826.75	16.38	16.49	1044.47	16.54	16.79	349.50
1000	1	23.53	23.71	6727.05	23.57	23.70	6376.81	23.57	23.70	6844.58	23.56	23.68	7531.09	23.84	24.07	3433.29
1000	2	23.31	23.45	7961.44	23.30	23.43	7609.14	23.29	23.42	9403.90	23.30	23.42	8952.48	23.64	23.87	3671.62
1000	3	23.15	23.26	6775.58	23.10	23.23	8244.05	23.11	23.23	6937.81	23.09	23.21	8775.89	23.37	23.61	3780.75
1000	4	23.55	23.68	7099.89	23.53	23.68	8406.42	23.50	23.67	8057.31	23.42	23.65	7517.29	23.86	24.03	3672.12
1000	5	23.23	23.36	8589.94	23.22	23.36	8923.28	23.18	23.34	8869.59	23.19	23.32	10075.39	23.54	23.77	3711.82

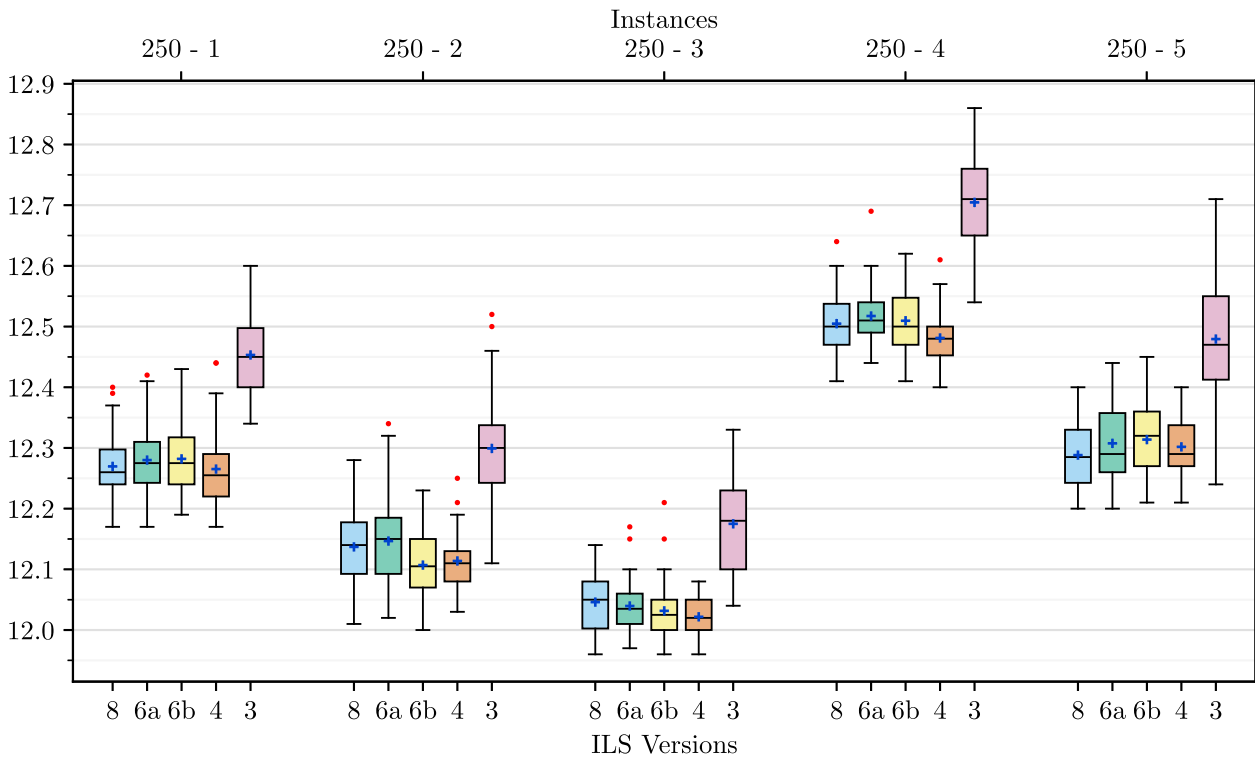


FIGURE 18. Boxplots for the 250-node instances, obtained from 50 runs of ILS/RVND for each pair (*instance, ILS version*).

IRACE was used first to select a subset of six neighborhoods through the execution of 300 experiments, where the value of each experiment consists of the average of the solution values obtained by 10 executions of ILS/RVND over the first instance of 250 nodes. The stop condition of each single ILS/RVND run was set to 250

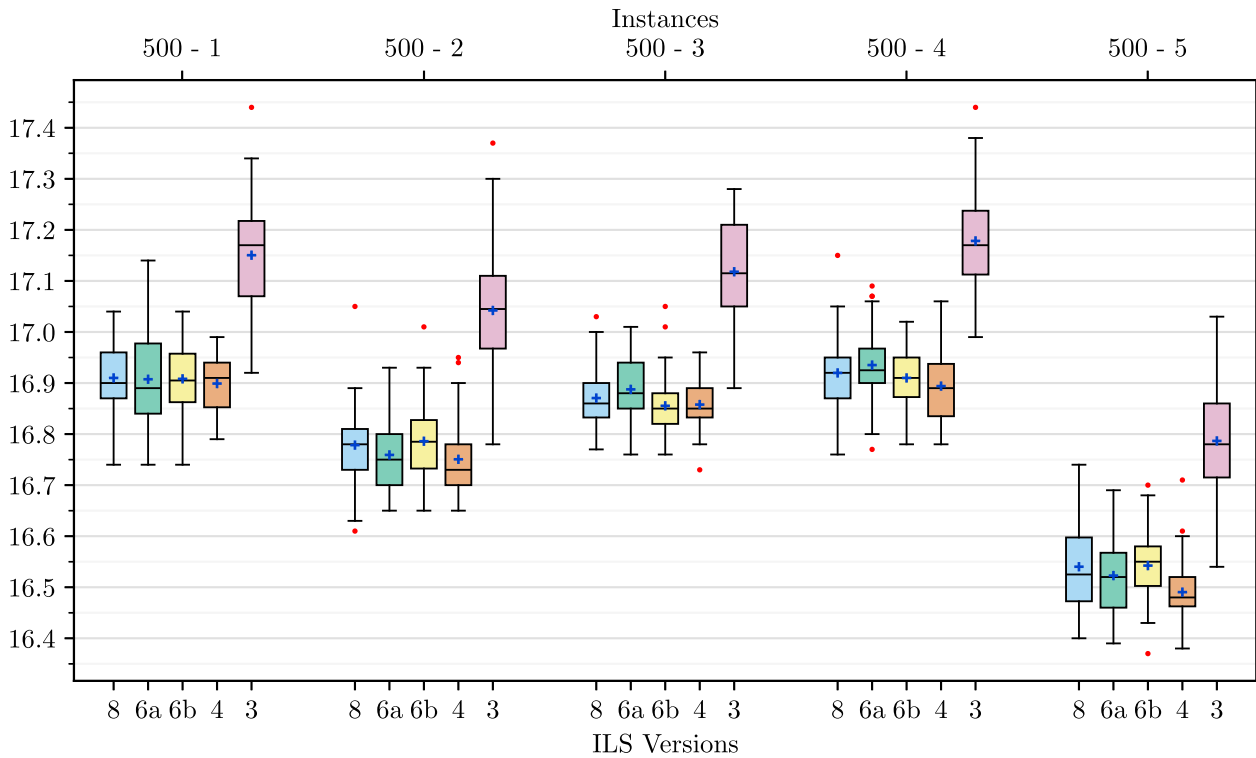


FIGURE 19. Boxplots for the 500-node instances, obtained from 50 runs of ILS/RVND for each pair (*instance*, *ILS version*).

iterations with no improvement. In this setting, IRACE’s response was counterintuitive in relation to what can be observed in Figure 17. While the figure suggests disabling PS and SO, IRACE recommended disabling PS and EE. Therefore, it turns out interesting that the comparative study includes the versions ILS-6a and ILS-6b (see Tab. 3).

Next, IRACE was used to select a subset of four neighborhoods, through the same experimental setting. IRACE’s response recommends enabling HR, NS, HE, and LR. Therefore, the comparative study also includes the version ILS-4.

Table 4 shows the results obtained by the five ILS versions listed in Table 3, for the 250-, 500-, and 1000-node instances. In the table, the columns have the following meaning: *n* is the number of nodes of the instance, *id* is the identification of the instance, *best* is the best solution value obtained by the ILS version indicated one row above, *mean* is the average solution value obtained over several runs, and *t* is the average time spent over the runs (in seconds). For each pair (*instance*, *ILS version*), 50 runs are performed, where a single run uses the following stop condition: 1000 iterations with no solution improvement. Figures 18–20 show boxplots constructed with the values obtained from the 50 runs for each pair (*instance*, *ILS version*). The average value corresponding to each pair (*instance*, *ILS version*), which is shown in the *mean* columns of Table 3, is marked with “+” in the boxplots.

From the results obtained from comparing different combinations of the ILS/RVND algorithm (see Tab. 4), we can observe that each combination offers gains in some aspects. For example, ILS-3 (which only uses the neighborhoods previously known in the literature) was faster than the other heuristics in all test cases. On the other hand, this behaviour is not repeated when considering solution quality. This corroborates the initial

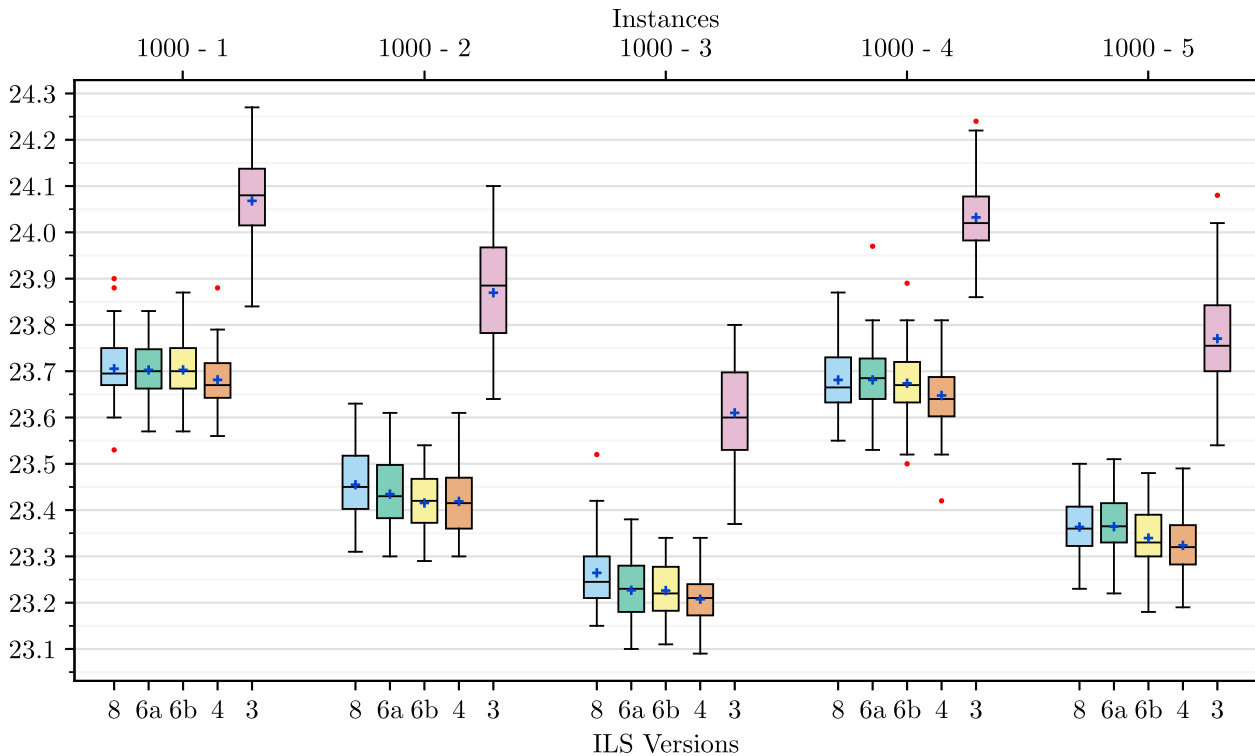


FIGURE 20. Boxplots for the 1000-node instances, obtained from 50 runs of ILS/RVND for each pair (*instance, ILS version*).

hypothesis that the use of the new neighborhoods structures allows the ILS/RVND to obtain better results than the previously known in the literature.

Despite ILS-4 demonstrating superior average results compared to the other versions, two critical factors deserve attention. First, its execution time was 25%, on average, longer than that of ILS-8. With the stopping criterion set at 1,000 iterations without improvement, the increased execution time suggests that ILS-4 took more time to converge. This delay indicates that adding the four additional neighborhoods applied in ILS-8 allowed the algorithm to explore more diverse solutions and effectively escape local optima, ultimately enhancing convergence speed. Figure 17 further supports this by showing how the neighborhoods deactivated in ILS-4 contribute to improvements in the best solution when applied. The second key consideration is the quality of the best solutions: ILS-8 matches or surpasses ILS-4 in 53% of the instances. Since all neighborhoods proved effective, mainly when used together, we chose to prioritize ILS-8 further tests and for comparisons with other algorithms in the literature, underlining its overall efficacy.

5.2.2. Experiments with ILS-8

For each instance, we perform 50 ILS/RVND runs using the following stop condition: 1000 iterations of the ILS/RVND main loop with no solution improvement, similarly to the approach used in [17, 19, 29].

From these experiments, we extracted information about the solutions returned in each of the fifty executions performed for each instance. Each instance corresponds to a column in the graphic, and each column is formed by three parts: the left and the center parts present, respectively, a violin plot and a box plot showing the results' distribution; the right part exhibits the values of each solution found.

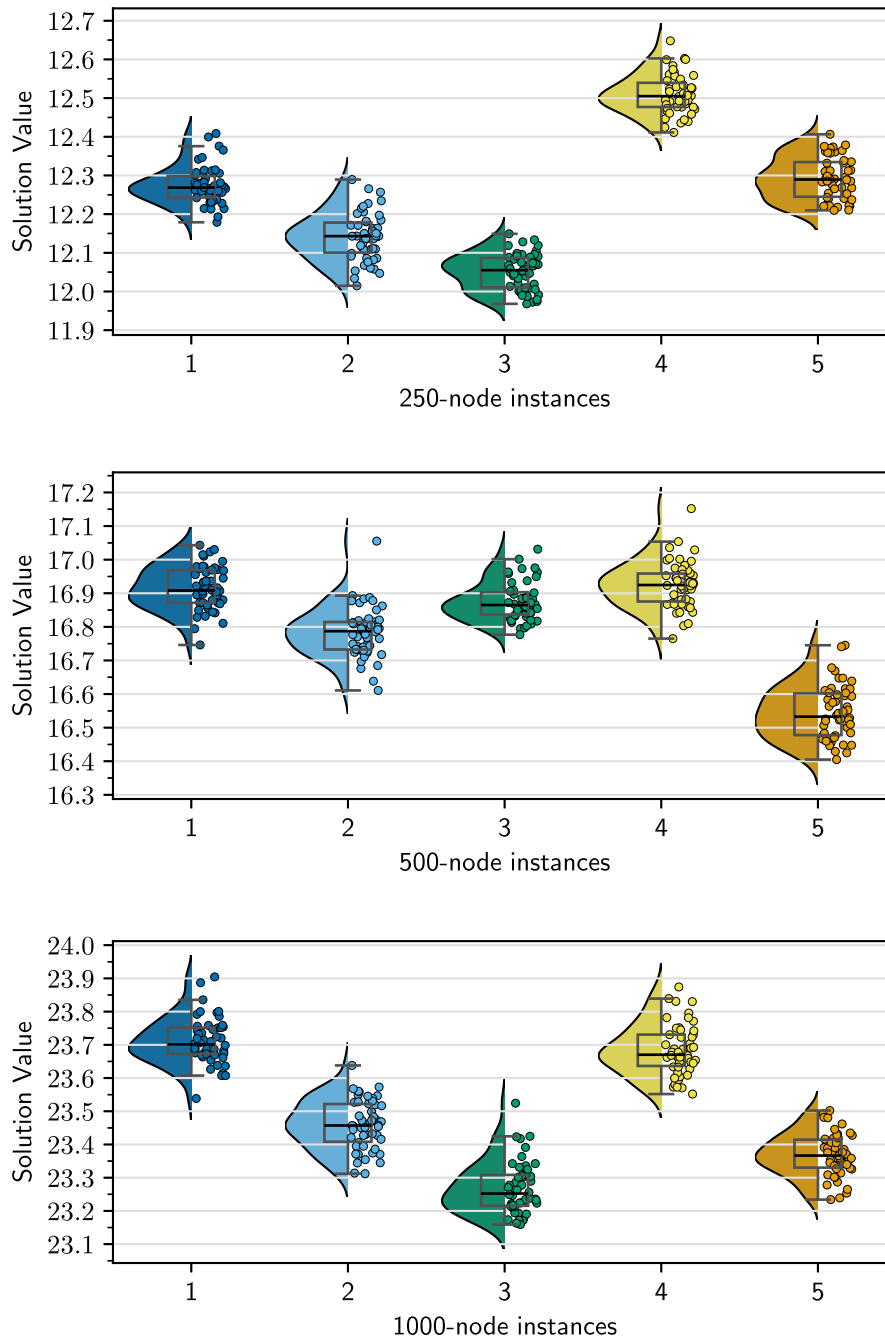


FIGURE 21. Distribution of solution values considering fifty executions of ILS-8 for each instance.

Observing the graphics, we can conclude that the results of the proposed ILS are concentrated around the mean solution value (marked as a red “+” in each boxplot), with a slight tendency towards values below it. Figure 21 presents three graphics referring to 200-, 500-, and 1000-node instances.

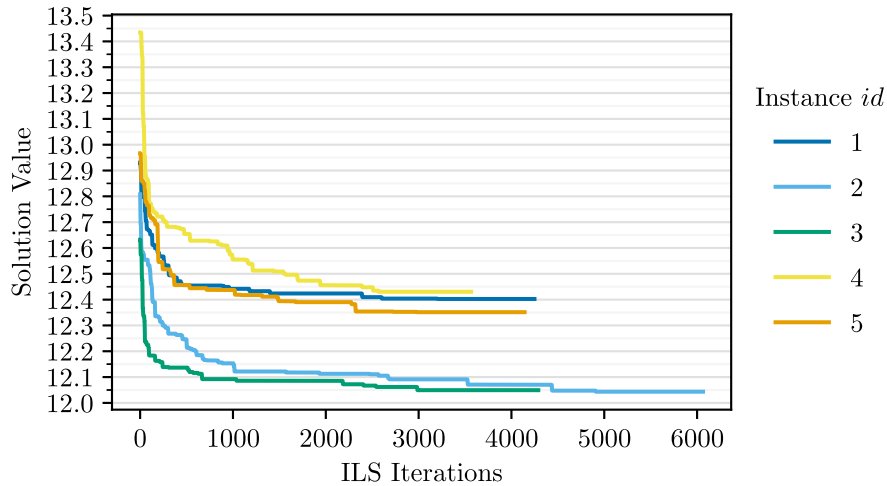


FIGURE 22. Convergence curves for single executions of ILS-8 for each of the 250-node instances.

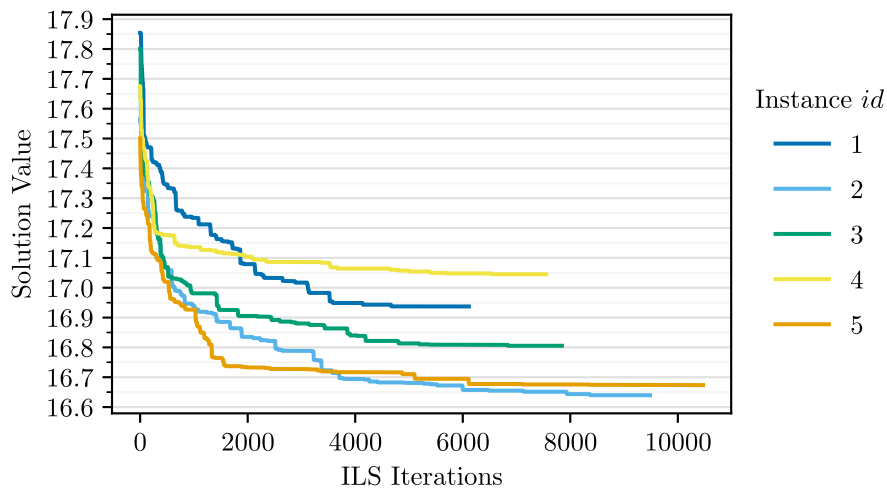


FIGURE 23. Convergence curves for single executions of ILS-8 for each of the 500-node instances.

We also performed an analysis based on a single randomized execution of the ILS algorithm. From this execution, we extract information about the convergence of the algorithm. As expected for ILS algorithms, the first iterations find many improvements from the initial solution, and then the algorithm starts a stabilization phase, where it finds improvements less frequently, which precedes the algorithm stop condition: 1000 iterations. Figures 22–24 show the convergence curve of a random execution of the algorithm for each instance.

5.2.3. Comparison with the literature

Continuing the experiments with ILS-8, we compared the original proposal of ILS/RVND (the ILS-8 version of the previous section) with the best results found in the literature, for the 250-, 500-, and 1000-node instances. We choose to show the values obtained by ILS-8 in the table because it was able to achieve the best solution values when looking at the columns *best* of Table 4. Table 5 presents the results of the comparison.

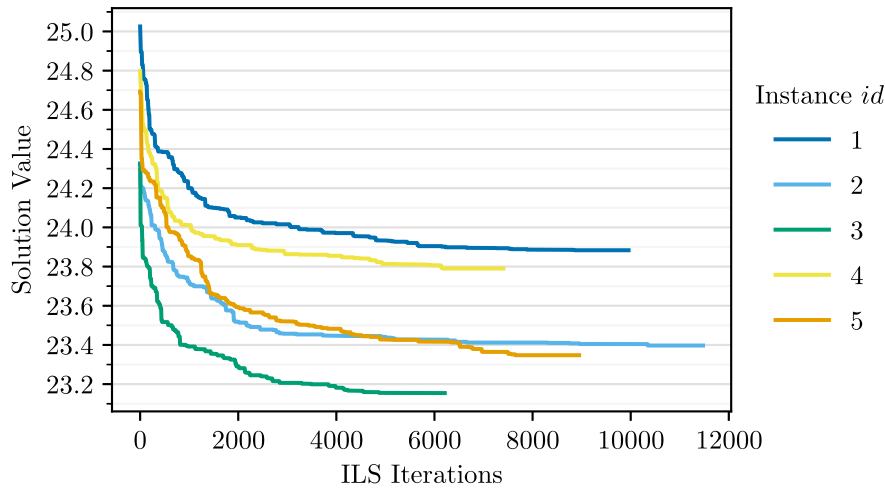


FIGURE 24. Convergence curves for single executions of ILS-8 for each of the 1000-node instances.

TABLE 5. Comparison between the results of ILS/RVND and the best results found in the literature.

n	id	d	lit	$best$	$mean$	sd	$t(s)$	$best^*$	$mean^*$	sd^*	$t^*(s)$
250	1	15	ACO ^a	12.23	12.28	0.02	174.17	12.17	12.27	0.04	85.69
250	2	15	LABD ^b	12.00	<i>n.a.</i>	<i>n.a.</i>	102.43	12.01	12.14	0.06	62.75
250	3	15	ACO ^a	12.00	12.02	0.01	145.29	11.96	12.05	0.04	58.57
250	4	15	PEA-I ^c	12.42	12.57	0.05	<i>n.a.</i>	12.41	12.51	0.04	72.56
250	5	15	ACO ^a	12.23	12.29	0.04	211.11	12.20	12.29	0.05	80.84
500	1	20	LABD ^b	16.37	<i>n.a.</i>	<i>n.a.</i>	210.24	16.74	16.91	0.06	698.42
500	2	20	ACO ^a	16.63	16.70	0.03	1012.91	16.61	16.78	0.07	681.90
500	3	20	ACO ^a	16.79	16.84	0.03	1069.84	16.77	16.87	0.05	755.77
500	4	20	ACO ^a	16.80	16.92	0.04	1010.91	16.76	16.92	0.06	752.34
500	5	20	ACO ^a	16.42	16.46	0.02	947.26	16.40	16.54	0.07	748.18
1000	1	25	LEEA ^a	23.43	23.57	0.08	565.38	23.53	23.71	0.06	6727.05
1000	2	25	LEEA ^a	23.46	23.67	0.08	561.49	23.31	23.45	0.07	7961.44
1000	3	25	2PMA ^d	23.53	23.76	0.12	<i>n.a.</i>	23.15	23.26	0.07	6775.58
1000	4	25	LEEA ^a	23.79	23.96	0.09	602.30	23.55	23.68	0.07	7099.89
1000	5	25	2PMA ^d	23.56	23.87	0.12	<i>n.a.</i>	23.23	23.36	0.06	8589.94

Notes. ^(a)Pentium 4 2.8 GHz running under the Linux operating system. ^(b)The author did not provide the computer configuration. ^(c)Pentium 4 2.4 GHz and 512 MB RAM running under the Linux operating system. ^(d)Intel Xeon X5650 2.67 GHz, processor-based system having 24 processor cores with 12 MB cache per processor and 32 GB RAM running the CentOS 5.5 Linux operating system.

As said above, both the perturbation procedure (line 4 of Algorithm 1) and the acceptance criterion (line 6 of Algorithm 1) do not consider the history of past computations. Such an approach is similar to that adopted in [29, 31, 35], and, as we shall see, leads ILS/RVND to good results when compared with the other existing heuristic methods.

The results are shown in Table 5, whose columns have the following meaning: n is the number of nodes, id is the identification of the instance, d is the diameter, lit is the method that obtains the best result known in the

literature, *best* is the best solution value obtained by the method indicated in the previous column, *mean* is the average solution value obtained over several runs, *st* is the standard deviation obtained over the runs, *t* is the average time spent over the runs (in seconds), *best** is the best solution value obtained by ILS/RVND, *mean** and *st** are the average solution value and the standard deviation (resp.) obtained over 50 runs of ILS/RVND, and *t** is the average running time spent over the 50 runs of ILS/RVND. In each row of Table 5, the best results are shown in bold text.

By comparing columns *best* and *best** in Table 5, ILS/RVND was able to improve 80% of the results found in the literature (12 of 15 instances). Regarding the *mean** column, ILS/RVND was able to overcome the results of the literature in about half of the cases. Furthermore, ILS/RVND obtained acceptable average execution times (column *t**), with standard deviation values (column *st**) within a reasonable limit.

6. CONCLUSIONS

In this work, we employed the exact method described in [15] to optimally solve widely-used OR-Library Euclidean instances of the BDMSTP with 50 and 100 nodes, whose optimal values were so far unknown.

To deal with larger OR-Library graph instances with 250, 500, and 1000 nodes, we described an ILS approach for the BDMSTP. Five new neighborhoods have been proposed for the local search, which is based on the Random Variable Neighborhood Descent method (RVND). In addition, four perturbation moves have been used.

We performed a comparative analysis of the neighborhoods, proposing five versions of the local search according to different subsets of neighborhoods. The IRACE package was used as an auxiliary tool to determine some of these subsets. As far as the authors know, this is the first attempt to use IRACE as a neighborhood selection tool. We also compared the results for the BDMSTP found in the literature with those obtained by the proposed ILS/RVND approach. The tests performed show that it was able to obtain good solution values when compared with previous existing results.

ACKNOWLEDGMENTS

This work was partially supported by CNPq and FAPERJ.

DATA AVAILABILITY STATEMENT

No new data/codes were created or analyzed in this study.

REFERENCES

- [1] A. Abdalla and N. Deo, Random-tree diameter and the diameter-constrained MST. *Int. J. Comput. Math.* **79** (2002) 651–663.
- [2] N. Achuthan, L. Caccetta, P. Caccetta and J. Geelen, Computational methods for the diameter restricted minimum weight spanning tree problem. *Australas. J. Comb.* **10** (1994) 51–71.
- [3] K. Bala, K. Petropoulos and T.E. Stern, Multicasting in a linear lightwave network, in IEEE INFOCOM '93 The Conference on Computer Communications, Proceedings. Vol. 3. IEEE Computer Society Press (1993) 1350–1358.
- [4] J.E. Beasley, OR-Library: Euclidean Steiner problem. <http://people.brunel.ac.uk/~mastjjb/jeb/info.html> (2005).
- [5] T.T.H. Binh, R.I. McKay, N.X. Hoai and N.D. Nghia, New heuristic and hybrid genetic algorithm for solving the bounded diameter minimum spanning tree problem, in Proceedings of the 11th Annual Conference On Genetic and Evolutionary Computation – GECCO '09. ACM Press, New York, NY, USA (2009) 373.
- [6] A. Bookstein and S.T. Klein, Compression of correlated bit-vectors. *Inf. Syst.* **16** (1991) 387–400.
- [7] M. Chiarandini and T. Stützle, An application of iterated local search to graph coloring, in Proceedings of the Computational Symposium on Graph Coloring and its Generalizations. Ithaca, New York, NY, USA (2002).
- [8] D.M. Da Silva, Y.A.M. Frota and A. Subramanian, Uma heurística para o problema de roteamento de veículos com múltiplas viagens, in Congresso Latino-Iberoamericano Investigación Operativa (2012) 1880–1891.
- [9] M.P. de Aragão, E. Uchoa and R.F. Werneck, Dual heuristics on the exact solution of large steiner problems. *Electron. Notes Discrete Math.* **7** (2001) 150–153.

- [10] X. Dong, H. Huang and P. Chen, An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion. *Comput. Oper. Res.* **36** (2009) 1664–1669.
- [11] M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-completeness. *Mathematical Sciences Series*. W.H. Freeman (1979).
- [12] F. Glover and G.A. Kochenberger, editors. Handbook of Metaheuristics. Vol. 57 of *International Series in Operations Research & Management Science*. Springer US, Boston, MA (2003).
- [13] L. Gouveia and T. Magnanti, Network flow models for designing diameter-constrained minimum spanning and steiner trees. *Networks* **41** (2003) 159–173.
- [14] L. Gouveia, A. Paias and D. Sharma, Restricted dynamic programming based neighborhoods for the hop-constrained minimum spanning tree problem. *J. Heuristics* **17** (2011) 23–37.
- [15] L. Gouveia, L. Simonetti and E. Uchoa, Modeling hop-constrained and diameter-constrained minimum spanning tree problems as Steiner tree problems over layered graphs. *Math. Program.* **128** (2011) 123–148.
- [16] M. Gruber and G.R. Raidl, A new 0–1 ILP approach for the bounded diameter minimum spanning tree problem, in Proceedings of 2nd International Network Optimization Conference, edited by L. Gouveia and C. Mourão (2005) 178–185.
- [17] M. Gruber and G.R. Raidl, Variable neighborhood search for the bounded diameter minimum spanning tree problem, in Proceedings of the 18th Mini Euro Conference on Variable Neighborhood Search, edited by P. Hansen, N. Mladenović, J.A.M. Pérez, B.M. Batista and J.M. Moreno-Vega. (2005).
- [18] M. Gruber and G.R. Raidl, (Meta-)Heuristic Separation of Jump Cuts in a Branch&Cut Approach for the Bounded Diameter Minimum Spanning Tree Problem. Vol. 10. Springer US (2009) 209–229.
- [19] M. Gruber, J. van Hemert and G.R. Raidl, Neighbourhood searches for the bounded diameter minimum spanning tree problem embedded in a VNS, EA, and ACO, in Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation – GECCO '06. ACM Press, New York, NY, USA (2006) 1187.
- [20] G.Y. Handler, Minimax location of a facility in an undirected tree graph. *Transp. Sci.* **7** (1973) 287–293.
- [21] P. Hansen and N. Mladenović, Variable neighborhood search: principles and applications. *Eur. J. Oper. Res.* **130** (2001) 449–467.
- [22] P. Hansen and N. Mladenović, Variable Neighborhood Search, in Search Methodologies. Springer US, Boston, MA (2005) 211–238.
- [23] B.A. Julstrom, Encoding bounded-diameter spanning trees with permutations and with random keys, in Genetic and Evolutionary Computation – GECCO 2004 (2004) 1272–1281.
- [24] B.A. Julstrom, Greedy heuristics for the bounded diameter minimum spanning tree problem. *ACM J. Exp. Algorithmics* **14** (2009) 1–14.
- [25] B.A. Julstrom and G.R. Raidl, A permutation-coded evolutionary algorithm for the bounded-diameter minimum spanning tree problem, in 2003 Genetic and Evolutionary Computation Conference’s Workshops Proceedings, Workshop on Analysis and Design of Representations (2003) 2–7.
- [26] W. Liu, Y. Gong, W. Chen, Z. Liu, H. Wang and J. Zhang, Coordinated charging scheduling of electric vehicles: a mixed-variable differential evolution approach. *IEEE Trans. Intell. Transp. Syst.* **21** (2020) 5094–5109.
- [27] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, M. Birattari and T. Stützle, The irace package: iterated racing for automatic algorithm configuration. *Oper. Res. Perspect.* **3** (2016) 43–58.
- [28] H.R. Lourenço, O.C. Martin and T. Stützle, Iterated local search, in Handbook of Metaheuristics. Kluwer Academic Publishers, Boston (2003) 320–353.
- [29] A. Lucena, C.C. Ribeiro and A.C. Santos, A hybrid heuristic for the diameter constrained minimum spanning tree problem. *J. Global Optim.* **46** (2010) 363–381.
- [30] N. Maculan, The steiner problem in graphs, in Annals of Discrete Mathematics. Vol. 31 of *North-Holland Mathematics Studies*, edited by S. Martello, G. Laporte, M. Minoux and C.C. Ribeiro. North-Holland (1987) 185–212.
- [31] I.C. Martins, R.G.S. Pinheiro, F. Protti and L.S. Ochi, A hybrid iterated local search and variable neighborhood descent heuristic applied to the cell formation problem. *Expert Syst. App.* **42** (2015) 8947–8955.
- [32] N. Mladenović and P. Hansen, Variable neighborhood search. *Comput. Oper. Res.* **24** (1997) 1097–1100.
- [33] C. Patvardhan and V.P. Prakash, Serial and parallel memetic algorithms for the bounded diameter minimum spanning tree problem. *Expert Syst.* **38** (2021) e12610.
- [34] C. Patvardhan, V.P. Prakash and A. Srivastav, Fast heuristics for large instances of the euclidean bounded diameter minimum spanning tree problem. *Informatika* **39** (2015) 281–292.
- [35] P.H.V. Penna, A. Subramanian and L.S. Ochi, An iterated local search heuristic for the heterogeneous fleet vehicle routing problem. *J. Heuristics* **19** (2013) 201–232.

- [36] R.C. Prim, Shortest connection networks and some generalizations. *Bell Syst. Tech. J.* **36** (1957) 1389–1401.
- [37] G.R. Raidl and B.A. Julstrom, Greedy heuristics and an evolutionary algorithm for the bounded-diameter minimum spanning tree problem, in Proceedings of the 2003 ACM symposium on Applied computing – SAC '03. ACM Press, New York, NY, USA (2003) 747.
- [38] G.R. Raidl and B.A. Julstrom, Edge sets: an effective evolutionary coding of spanning trees. *IEEE Trans. Evol. Comput.* **7** (2003) 225–239.
- [39] K. Raymond, A tree-based algorithm for distributed mutual exclusion. *ACM Trans. Comput. Syst.* **7** (1989) 61–77.
- [40] A.C. Santos, A. Lucena and C.C. Ribeiro, Solving diameter constrained minimum spanning tree problems in dense graphs, in Experimental and Efficient Algorithms, edited by C.C. Ribeiro and S.L. Martins. Springer Berlin Heidelberg (2004) 458–467.
- [41] A. Singh and A.K. Gupta, Improved heuristics for the bounded-diameter minimum spanning tree problem. *Soft Comput.* **11** (2007) 911–921.
- [42] K. Singh and S. Sundar, A heuristic for the bounded diameter minimum spanning tree problem, in Proceedings of the 2nd International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence – ISMSI '18. ACM Press, New York, NY, USA (2018) 84–88.
- [43] M.J.F. Souza, I.M. Coelho, S. Ribas, H.G. Santos and L.H.C. Merschmann, A hybrid heuristic algorithm for the open-pit-mining operational planning problem. *Eur. J. Oper. Res.* **207** (2010) 1041–1051.
- [44] W. Steitz, New heuristic approaches for the bounded-diameter minimum spanning tree problem. *INFORMS J. Comput.* **27** (2015) 151–163.
- [45] A. Subramanian, L.M.A. Drummond, C. Bentes, L.S. Ochi and R. Farias, A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. *Comput. Oper. Res.* **37** (2010) 1899–1911.
- [46] J.A. Torkestani, An adaptive heuristic to the bounded-diameter minimum spanning tree problem. *Soft Computing* **16** (2012) 1977–1988.
- [47] F. Zhao, X. He and L. Wang, A two-stage cooperative evolutionary algorithm with problem-specific knowledge for energy-efficient scheduling of no-wait flow-shop problem. *IEEE Trans. Cybern.* **51** (2021) 5291–5303.
- [48] F. Zhao, R. Ma and L. Wang, A self-learning discrete jaya algorithm for multiobjective energy-efficient distributed no-idle flow-shop scheduling problem in heterogeneous factory system. *IEEE Trans. Cybern.* **52** (2021) 12675–12686.
- [49] F. Zhao, L. Zhang, J. Cao and J. Tang, A cooperative water wave optimization algorithm with reinforcement learning for the distributed assembly no-idle flowshop scheduling problem. *Comput. Ind. Eng.* **153** (2021) 107082.
- [50] S. Zhou, L. Xing, X. Zheng, N. Du, L. Wang and Q. Zhang, A self-adaptive differential evolution algorithm for scheduling a single batch-processing machine with arbitrary job sizes and release times. *IEEE Trans. Cybern.* **51** (2021) 1430–1442.



Please help to maintain this journal in open access!

This journal is currently published in open access under the Subscribe to Open model (S2O). We are thankful to our subscribers and supporters for making it possible to publish this journal in open access in the current year, free of charge for authors and readers.

Check with your library that it subscribes to the journal, or consider making a personal donation to the S2O programme by contacting subscribers@edpsciences.org.

More information, including a list of supporters and financial transparency reports, is available at <https://edpsciences.org/en/subscribe-to-open-s2o>.