# A Hybrid Heuristic for the Diameter Constrained Minimum Spanning Tree Problem

Abilio Lucena[1], Celso C. Ribeiro[2], and Andréa C. Santos[3]

[1] Federal University of Rio de Janeiro, Av. Pasteur 250,
Rio de Janeiro, RJ 22290-240, Brazil.
lucena@facc.ufrj.br
[2] Federal University Fluminense, Institute of Computing,
Rua Passo da Pátria 156, Niterói, RJ 24210-240, Brazil.
celso@ic.uff.br
[3] Blaise Pascal University, LIMOS,
Complexe Scientifique des Cézeaux, Aubière, 63173, France.
andrea@isima.fr

**Abstract.** We propose a hybrid GRASP and ILS based heuristic to the diameter constrained minimum spanning tree problem. The latter typically models network design applications where, under a given quality requirement, all vertices must be connected at minimum cost. An adaptation of the One Time Tree Heuristic is used to build feasible diameter constrained spanning trees. Solutions thus obtained are then attempted to be improved through local search. Four different neighborhoods are investigated, in a scheme similar to VND. Upper bounds within 2% of optimality were obtained for problems in two test sets from the literature. Additionally, upper bounds stronger than those previously obtained in the literature are reported for OR-Library instances.

## 1 Introduction

Let $G = (V, E)$ be a connected undirected graph with a set $V$ of vertices, a set $E$ of edges, and a cost $c_{ij}$ associated with every edge $[i, j] \in E$. Considering $\mathcal{T}$ a spanning tree of $G$, the *diameter* of $\mathcal{T}$ is equal to the number of edges in the longest path linking any two nodes $i, j \in V$, $i \neq j$. Given an integer parameter $2 \leq D \leq |V| - 1$, the *Diameter Constrained Minimum Spanning Tree Problem* (DCMST) seeks a least cost spanning tree of $G$ whose diameter does not exceed $D$. DCMST was proved to be $NP$-hard when $D \geq 4$ [10] and has been used as a model for applications in telecommunication [4, 28], data compression [6], and distributed mutual exclusion in parallel computing [7, 27].

Several DCMST formulations are found in the literature. The very first ones were introduced in [2, 3] and use single commodity network flows to simultaneously impose solution connectivity and diameter constraints. These formulations distinguish odd and even $D$ cases and variants of them are found in [3]. They are reinforced with valid inequalities and submitted to a lifting procedure in [31]. Formulations based on multi-commodity network flows were suggested in [12, 11]. One of these formulations does not distinguish the odd and even $D$ cases, while the others do. The formulations in [11, 12] produce tighter linear programming (LP) relaxations than those previously obtained in the literature. However, these improved bounds are attained at a considerable CPU time cost. More recently, new odd and even $D$ formulations were introduced and reinforced with valid inequalities in [14]. Proven optimal solutions for complete graph instances with up to 25 vertices were obtained in [14, 31] and for sparse graph instances with up to 60 vertices and 600 edges in [12, 13, 31].

A number of heuristics for solving DCMST are found in the literature. The One Time Tree (OTT) constructive heuristic, based on the Prim algorithm [25], and heuristics which iteratively refine a minimum spanning tree were suggested in [1]. A randomized greedy heuristic (RGH) and a genetic algorithm were proposed in [26]. From the available computational evidence, RGH performs better than OTT. Improved versions of the genetic algorithm are described in [21, 22]. A Variable Neighborhood Search (VNS) heuristic is proposed in [14] and this algorithm has been shown to perform better than the previous genetic algorithms.

A hybrid algorithm combining the principles of Greedy Randomized Adaptive Search Procedures (GRASP) [9, 29] and Iterated Local Search (ILS) [23] is proposed in this paper. Hybridizations of the iterated local search metaheuristic with multistart approaches have been successfully developed to a number of applications (see e.g. [8, 16, 30]) and they seem to be a promising search strategy. Our algorithm was tested on three test sets from the literature. The first set was suggested in [31], while the second originates from [12]. The third test set was used in [14, 15, 21, 26] and comes from the OR-Library [5]. Proven optimal solutions are known for instances in the first and the second test sets and were used to evaluate the quality of the solutions produced by the hybrid algorithm. For instances in these sets, the best solution values obtained by the hybrid algorithm were always within 2% of the corresponding optimal values. Additionally, for several instances in the third test set, new best upper bounds were established.

This paper is organized as follows. GRASP and ILS metaheuristics are briefly reviewed in Section 2. In Section 3, the hybrid algorithm is described in detail. Computational results are reported in Section 4. Finally, concluding remarks are drawn in Section 5.

## 2 GRASP and ILS

Metaheuristics are general high-level procedures that coordinate simple heuristics and rules to find good approximate (often optimal) solutions to computationally difficult combinatorial optimization problems. Among them, we find simulated annealing, tabu search, GRASP, ILS, genetic algorithms, ant colonies, and others. They are based on distinct paradigms and offer different mechanisms to escape from locally optimal solutions. Metaheuristics have been widely used in the literature to find good quality solutions to $NP$-hard problems. Additionally, they have recently been combined into hybrid heuristics that borrow ideas from two or more metaheuristics. In particular, the hybrid algorithm proposed in this paper is based on GRASP and ILS.

GRASP is a multi-start metaheuristic which combines a constructive phase with a local search procedure. The pseudo-code of a generic GRASP algorithm is presented in Algorithm 1. A feasible solution is generated and submitted to a local search at every iteration. The best feasible solution obtained after a given pre-specified number of iterations is returned as the output.

**Input**: stopping criterion, *seed*
**Output**: $S^*$

Initialize $S^*$;
**while** (stopping criterion not met) **do**
    $S \leftarrow$ Construction($seed$);
    $S \leftarrow$ Local search($S$);
    **if** $S$ is better than $S^*$ **then**
        $S^* \leftarrow S$;
    **end**
**end**
**return** $S^*$;

**Algorithm 1**: GRASP pseudo-code.

As for ILS, it essentially applies, at every iteration, perturbations to feasible solutions and the resulting perturbed solutions are then submitted to a local search. A generic ILS pseudo-code is presented in Algorithm 2. Initially, a feasible solution is generated and a local search is applied to it. After that, perturbation performing iterations, followed by local search, are successively implemented. In this process, an acceptance criterion is used to determine, after carrying out a perturbation and local search round, if the solution thus obtained is fit to become the new current solution. The best overall solution thus obtained is returned as the output.

## 3 A hybrid DCMST heuristic

The main components of our hybrid algorithm are a fast constructive heuristic to generate feasible solutions, local search procedures, and perturbation implementing procedures. The algorithm follows the same general scheme adopted

```
Input: stopping criterion
Output: S*
S ← initial solution;
S ← Local search(S);
while (stopping criterion not met) do
    S' ← Perturbation(S);
    S' ← Local search(S');
    if S' is better than S* then
        S* ← S';
    end
    S ← Acceptance criterion(S, S');
end
return S*;
```

**Algorithm 2**: ILS pseudo-code.

in [30]. Fast heuristics such as RGH, OTT or a modified randomized version of OTT (OTT-M) could be used to generate initial solutions. They are described in Section 3.1 and compared in Section 4.1. A semi-greedy (or greedy randomized) heuristic, designed with the same purpose, is described in Section 3.2. In Section 3.3, we present four different neighborhoods which are exploited in a scheme similar to Variable Neighborhood Descent (VND) [18, 19]. Perturbations and filters are discussed respectively in Sections 3.4 and 3.5. An introductory general view of the hybrid algorithm is presented in Figure 1.

The hybrid algorithm is initiated with the application of a constructive heuristic, such as OTT-M. If the underlying graph is complete, the solution thus obtained is guaranteed to be feasible and is accepted outright. However, for sparse graphs, no guarantee exists that a feasible solution is returned. For such graphs, artificial edges with high costs are inserted into the original graph to make them complete. If the number of artificial edges in the solution returned by the constructive heuristic is large, the solution is rejected and the constructive heuristic is called once again. This process is repeated until either a feasible solution is obtained or else the number of artificial edges in the solution is considered as acceptable.

The feasible solution produced by the procedure outlined above is submitted to local search. In turn, local search is followed by a pre-specified number of perturbation and local search rounds to conclude an iteration of the hybrid algorithm. Within these rounds, local search is only carried out for perturbed solutions that survive a given filter check, to be described in Section 3.5.

Provided a stopping criterion has not been met, a new iteration of the hybrid algorithm is initiated. In this process, the overall best feasible solution found by the algorithm is returned as the output. In summary, for the hybrid algorithm, the local search phase of every GRASP iteration is replaced by a ILS heuristic.

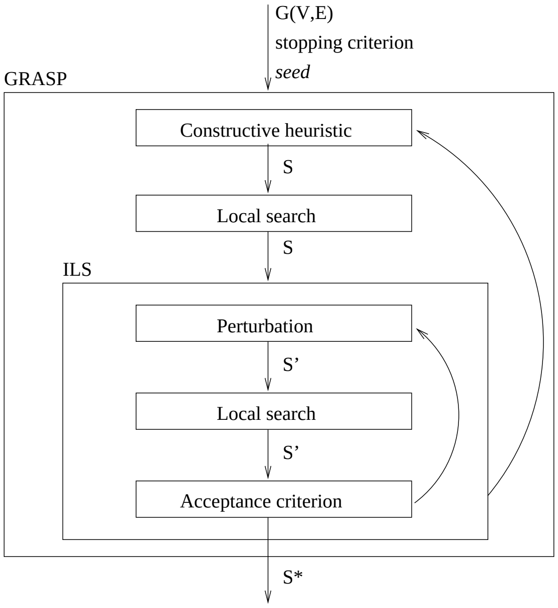


**Fig. 1.** A general view of the hybrid algorithm.

Associated filters help to select feasible solutions that are worthy spending time on.

### 3.1 Constructive heuristics

Greedy constructive heuristics expand feasible solutions, one element at a time, after evaluating the impact in the objective function of adding each possible element still missing in the current partial solution under construction. In a minimization problem, for example, the impact a variable has on entering the solution is evaluated by a function that estimates the increase in objective function value. In doing so, and provided no infeasibility results, a candidate element with the best immediate recorded impact is selected to expand the incomplete solution in hand.

The best constructive heuristics available to DCMST are OTT [1] and RGH [26]. Heuristic OTT is based on Prim's algorithm [25]. It is initialized with any single vertex in $V$. Being somewhat lax with the notation, let us call that single vertex a (non spanning) tree $\mathcal{T}$ of $G$. At each iteration of the algorithm, tree $\mathcal{T}$ is expanded with the introduction of a vertex in $V$ not yet included in $\mathcal{T}$. The chosen vertex is connected to $\mathcal{T}$ through a least cost edge that does not create a cycle in that structure. In doing so, a check is made to prevent violations of the diameter constraint: candidate edges that would lead to diameter violations are not considered for the expansion of a (non spanning) tree $\mathcal{T}$ of $G$. The algorithm terminates when all vertices in $V$ belong to the tree, which thus becomes a spanning tree for $G$. Since the minimum-cost edge whose insertion does not create a cycle can be determined in time $O(|V|^2)$, algorithm OTT runs in time $O(|V|^3)$ [1].

For the case of RGH [26], tree $\mathcal{T}$ is also initialized with a single vertex in $V$. At each iteration, $\mathcal{T}$ is expanded through a randomly chosen node which is not yet in $\mathcal{T}$. This node is connected to the tree via the smallest cost edge, satisfying the diameter constraint. Algorithm RGH takes $O(|V|^2)$ time in the worst case.

Heuristic RGH performed better than OTT in the experiments reported in [26]. However, it is important to highlight that there is no guarantee that either RGH or OTT will necessarily return a feasible solution for sparse graphs instances.

### 3.2 Semi-greedy constructive heuristic OTT-M

At each iteration of a semi-greedy heuristic, good variables, but not necessarily the local best, may eventually be selected to enter the expanding solution. A semi-greedy strategy to select an entering variable, based on a Restricted Candidate List (RCL), is proposed in [20]. First, a RCL is drawn and then the next entering element is randomly selected from it. Various criteria, including the immediate impact on the objective function and an upper limit on the number of elements allowed in the list, may be used in controlling the RCL size. Semi-greedy heuristics are also referred to as greedy randomized heuristics, since they randomly select one element from those in RCL.

We modified OTT by introducing a RCL to it. Such a list invokes the product of two different criteria to select member vertices. The first one is the cost of the least cost edge that connects a given vertex to a vertex in $\mathcal{T}$ (to attempt to induce low cost solutions). The second is the cardinality of the largest path that results in $\mathcal{T}$, after insertion of that vertex (to attempt to avoid diameter violations). A given candidate vertex is accepted into RCL if the product of its two corresponding values is less than or equal to

$$f_{lim} = f_{min} + \alpha(f_{max} - f_{min}). \tag{1}$$

For equation (1), $f_{min}$ and $f_{max}$ give, respectively, the smallest and the largest corresponding products specified above, for all vertices currently outside $\mathcal{T}$. Parameter $\alpha$ is to assume a value in [0,1]. If $\alpha$ is set to 0, a pure greedy minimization algorithm is implied, while $\alpha = 1$ leads to a completely random construction.

We have also implemented an on-line automatic adjustment of the value of $\alpha$ by following the Reactive GRASP strategy suggested in [24]. Accordingly, let

$\{\alpha_1, \alpha_2, ..., \alpha_m\}$, where $m \geq 1$, be a set of discrete values with corresponding associated probabilities $\{p_i : i = 1, \ldots, m\}$. In our experiments, we fixed $m$ to 10 and used $\alpha_i = 0.1(i-1)$, for $i = 1, \ldots, m$. The value of $\alpha$ is randomly chosen from that set to build a solution using the semi-greedy constructive heuristic.

In relation with the discussion above, assume that probabilities $\{p_i = 1/m : i = 1, \ldots, m\}$ are initially taken. Assume as well that $A_i$ is the average value of the feasible solutions obtained under the use of $\alpha = \alpha_i$ in equation (1). In order to penalize low probability values, a dumping factor $\beta > 0$ is used. As a result, corresponding probabilities are computed as

$$p_i = q_i / \sum_{j=1}^{m} q_j, \quad i = 1, \ldots, m \tag{2}$$

where

$$q_i = (1/A_i)^{\beta}, \quad i = 1, \ldots, m. \tag{3}$$

In doing so, values of $\alpha$ leading to better results will have a higher probability of being selected. Probabilities $p_i$ were updated at every 100 iterations and we set $\beta = 8$ in the computational experiments. Likewise OTT, the worst case complexity of OTT-M is $O(|V|^3)$.

### 3.3 Neighborhoods

Four different neighborhoods were investigated in our local search procedures. They are denoted respectively by *restricted* 1-*opt neighborhood*, *restricted* 2-*opt neighborhood*, *subtree adoption neighborhood*, and *path exchange neighborhood*. Accordingly, the local search procedures are respectively identified by the names of the neighborhoods they exploit.

While implementing local search procedures, we use a property of feasible diameter constrained spanning trees. For even $D$, Handler [17] pointed out that a *central vertex* $i \in V$ must exist in any diameter constrained tree $\mathcal{T}$ such that no other vertex in $\mathcal{T}$ is more than $D/2$ edges away from $i$. Conversely, if $D$ is odd, a *central edge* $e = [i, j] \in E$ must exist in $\mathcal{T}$ such that no vertex in $\mathcal{T}$ is more than $(D-1)/2$ edges away from its closest extremity of $[i, j]$. Thus, any diameter constrained tree we investigate either has an associated central vertex or else a central edge, depending on $D$ being even or odd. Furthermore, trees are oriented, from their roots toward their leaves, in a father/sons structure. In doing so, a root vertex of $\mathcal{T}$ is assumed to be hierarchically higher than any other vertex in the tree.

**Restricted 1-opt neighborhood.** Given a diameter constrained spanning tree $\mathcal{T}$ of $G$, the 1-opt neighborhood consists of all solutions that can be obtained by exchanging, under certain conditions, one edge in $\mathcal{T}$ for another not in $\mathcal{T}$. These conditions specify the way an edge is removed from $\mathcal{T}$ and how the two resulting non spanning trees of $G$ should be reconnected, to generate a new diameter constrained tree.

Removing a given edge in $\mathcal{T}$ results in two (non spanning) trees of $G$. They are respectively denoted *primary tree* ($\mathcal{T}_p$) and *secondary tree* ($\mathcal{T}_s$). Primary trees must necessarily contain the central vertex of $\mathcal{T}$ in the even $D$ case or the central edge of $\mathcal{T}$ in the odd $D$ case. Additionally, for odd $D$ instances, we ensure that no central edge of $\mathcal{T}$ is ever selected for removal. Edges that are candidates for reconnection of $\mathcal{T}_p$ and $\mathcal{T}_s$ must necessarily link a vertex in $\mathcal{T}_p$ to the root vertex in $\mathcal{T}_s$. Among all candidate edges that do not result in violations of the diameter constraint, one with the least cost is selected for reconnection. A restricted 1-opt neighborhood move is accepted whenever the returned tree has a cost smaller than $\mathcal{T}$. Such a move is depicted in Figure 2.

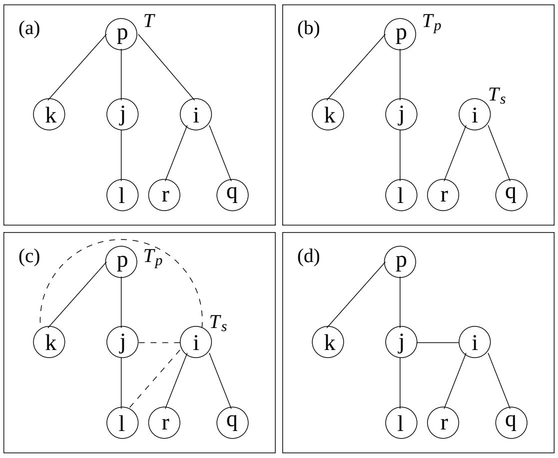


**Fig. 2.** Example of a restricted 1-opt neighborhood move.

Figure 2-(a) shows the diameter constrained tree $\mathcal{T}$ over which restricted 1-opt neighborhood moves are to be performed. Figure 2-(b) shows trees $\mathcal{T}_p$ and $\mathcal{T}_s$ that result from the removal of edge $[i, p]$ from $\mathcal{T}$. Candidate edges for reconnection of $\mathcal{T}_p$ and $\mathcal{T}_s$ are identified by dotted lines in Figure 2-(c). Assume that $[i, j]$ is the least cost reconnection edge that does not incur in diameter constraint violations. The feasible tree that results from the suggested restricted 1-opt neighborhood move is shown in Figure 2-(d).

The local search procedure we implemented to exploit restricted 1-opt neighborhoods enforces that every candidate edge is removed, one at a time, from a feasible tree $\mathcal{T}$. The resulting subtrees are reconnected through the least cost edge satisfying the diameter constraint. The first improving tree is then accepted as the new incumbent solution and the search stops when no improvement is attained after investigating all possibilities involved. Since $O(|V|)$ edges could be removed from $\mathcal{T}$, and since reconnecting the resulting subtrees as mentioned above involves $O(|V|)$ steps, examining all moves in each local search iteration takes time $O(|V|^2)$ in the worst case.

**Restricted 2-opt neighborhood.** Given a diameter constrained spanning tree $\mathcal{T}$ of $G$, its associated restricted 2-opt neighborhood is defined by the diameter constrained spanning trees of $G$ that, under certain conditions, are obtained after removing exactly two edges in $\mathcal{T}$. These conditions are associated with the way the two edges are selected and the way the three resulting non spanning trees of $G$ are reconnected.

Removing two edges in $\mathcal{T}$ results in one primary tree and two secondary trees being generated. Central edges must never be candidates for removal. Furthermore, the two secondary trees thus generated must necessarily be reconnected through an edge linking their respective roots. Once the two secondary trees are reconnected, one is left with a primary tree and a single secondary tree. Then, the restricted 1-opt neighborhood rules, defined above for reconnection of a primary and a secondary tree are applied. A restricted 2-opt neighborhood move for an even $D$ instance is depicted in Figure 3.

Figure 3-(a) shows the diameter constrained tree $\mathcal{T}$ to which restricted 2-opt neighborhood moves will be applied. Figure 3-(b) shows trees $\mathcal{T}_p$ and the two secondary trees $\mathcal{T}_s$ and $\mathcal{T}_s'$ which were obtained while removing edges $[p, i]$ and $[q, j]$ from $\mathcal{T}$. The two secondary trees $\mathcal{T}_s$ and $\mathcal{T}_s'$ are reconnected through edge $[i, j]$ (their respective roots) in Figure 3-(c). Candidate edges for reconnection of $\mathcal{T}_p$ and the new $\mathcal{T}_s$ are identified by dotted lines in Figure 3-(d) and Figure 3-(e). Assume that $[j, k]$ is the least cost reconnection edge that does not incur in diameter constraint violations. The feasible tree that results from the suggested restricted 2-opt neighborhood move is then shown in Figure 3-(f).

The local search using the 2-opt neighborhood updates the current solution when the first improvement move is found, following a first-improvement search strategy. There are $O(|V^2|)$ possibilities to remove two edges in $\mathcal{T}_p$ and reconnecting each subtree takes time $O(|V|)$. Thus, the evaluation of all moves in each local search iteration runs in time $O(|V^3|)$.

**Subtree adoption neighborhood.** Given a diameter constrained tree $\mathcal{T}$, a non leaf vertex $i$ of $\mathcal{T}$ must be selected to initiate a subtree adoption neighborhood move. This vertex is required to have at least two children and, once it is selected, the corresponding edge linking vertex $i$ to its parent $p$ in $\mathcal{T}$ must be removed from the tree. Such action gives rise to a primary tree $\mathcal{T}_p$ and a secondary tree $\mathcal{T}_s$. Denote by $i_1, \ldots, i_l$, where $l \geq 2$, the children of vertex $i$

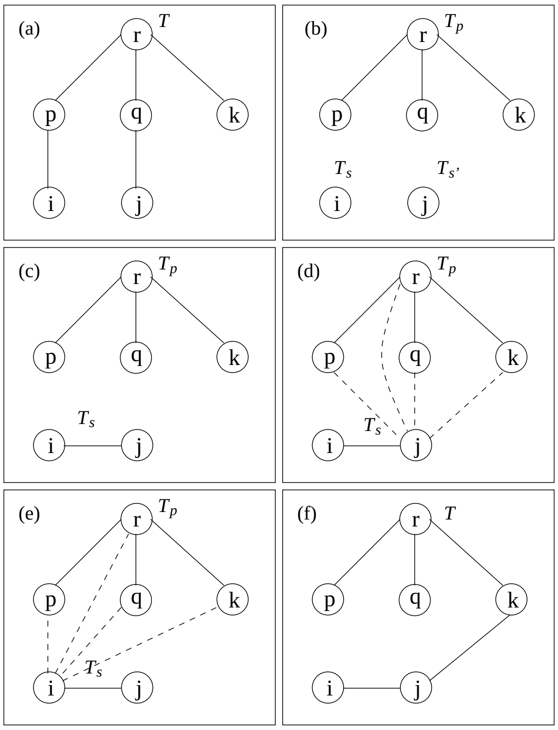


**Fig. 3.** Example of a restricted 2-opt neighborhood move.

in $\mathcal{T}_s$. The subtree adoption neighborhood consists of building a new secondary tree $\mathcal{T}_s$ having the same set of vertices as $T_s$ and such that one of the vertices in $\{i_1, \ldots, i_l\}$, say $i_k$, will act as its root. In this process, one is required to forcing $i$ to become a leaf of the new secondary tree and every (sub)tree having a vertex in $\{i_1, \ldots, i_l\} \setminus \{i_k\}$ as root should be directly connected to $i_k$. Furthermore, $\mathcal{T}_p$ is reconnected to $\mathcal{T}_s$ by edge $[p, i_k]$. Diameter constraint feasibility is maintained throughout this process.

A subtree adoption move is accepted whenever the returned tree has a cost smaller than $\mathcal{T}$. The best move in the neighborhood is applied to the current solution, characterizing a best-improvement search-strategy. There are $O(|V|)$ nodes capable to participate of a subtree adoption move. Computing the new secondary tree cost requires up to $O(|V|)$ steps. Therefore, each local search iteration takes time $O(|V^2|)$.

A subtree adoption move is shown in Figure 4. Tree $\mathcal{T}$ is shown in Figure 4-(a) where $i$ is the vertex selected to implement the move. All possible roots for secondary trees are depicted in Figures 4-(b) to 4-(d), together with their corresponding associated subtrees.

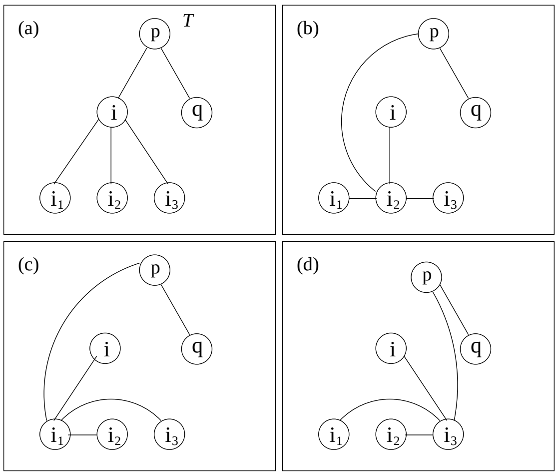


**Fig. 4.** Subtree adoption move.

**Path exchange neighborhood.** Given a diameter constrained tree $\mathcal{T}$, let $i$, $j$, and $k$ be the vertices that define a two edges path, where $j$ is a child of $i$ and $k$ is a child of $j$. A path exchange neighborhood is defined by the diameter constrained trees of $G$ obtained, under certain conditions, after removing edges $[i, j]$ and $[j, k]$ from $\mathcal{T}$. A description of these conditions follows.

Neither edge $[i, j]$ or $[j, k]$ may be a central edge of $\mathcal{T}$. Vertex $i$, however, is allowed to be a central vertex of that tree. In turn, vertex $j$ is requested to have at least two children.

After removing edges $[i, j]$ and $[j, k]$ from $\mathcal{T}$, three (sub)trees of $G$ result. One of them is a primary tree, while the others are secondary ones. Assume that $j_1, \ldots, j_l$, where $l \geq 2$, are the children of $j$, including $k$. The two secondary trees are then required to be reconnected through an edge that links $k$ to one of the vertices in $\{j_1, \ldots j_l\} \setminus \{k\}$, say vertex $j_s$. Either vertex $j_s$ or $k$ should then be selected as root of the resulting secondary tree and be directly linked to vertex $i$, in an attempt to obtain a diameter constrained tree of $G$. For the path exchange move described above, only one diameter constraint feasibility test must be carried out. This test is associated with the reconnection of the secondary tree rooted in $j$. That, in turn, may bring about diameter constraint violations.

A subtree adoption move is accepted whenever the returned tree has a cost smaller than $\mathcal{T}$ and it does not violate the diameter constraint, thus characterizing a first-improvement search strategy. Investigating all moves in a local search iteration takes $O(|V|^2)$ time.

Figure 5 illustrates a path exchange neighborhood move. Figure 5-(a) shows the initial diameter constrained tree $\mathcal{T}$. Figure 5-(b) indicates the removal path $[i, j]$ and $[j, j_2]$ from $\mathcal{T}$. Two possibilities to reconnect the secondary subtrees are specified in Figure 5-(c) and Figure 5-(d). In Figure 5-(c), the new path contains edges $[i, j_1]$ and $[j_1, j_2]$, while in Figure 5-(d), the new path contains edges $[i, j_3]$ and $[j_3, j_2]$.

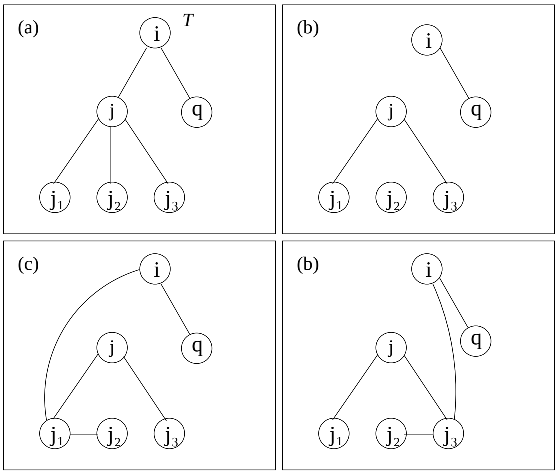


**Fig. 5.** A path exchange move.

## 3.4 Perturbations

Two types perturbations may be applied to the current solution in this study. They involve the exchange of a central vertex when $D$ is even or one extremity of the central edge when $D$ is odd.

Assume that trees are oriented, from their roots toward their leaves, in a parent/children structure as in the local search procedures presented in Section 3.3. Let $\mathcal{T}$ be a feasible oriented diameter constrained tree of $G$. For the first perturbation, a child of the central vertex becomes the new root of $\mathcal{T}$ when $D$ is even, or a new extremity of the central edge when $D$ is odd. After this exchange, tree vertex levels must be adjusted accordingly. For even $D$ instances, under the exchange just suggested, vertices that are more than $D/2$ edges away from the new root must be initially disconnected from $\mathcal{T}$. They should then be attempted to be feasibly reconnected to $\mathcal{T}$, through a least cost edge criterion. Similar arrangements must also be made for odd $D$ instances.

The second perturbation consists of selecting randomly a vertex to act as the new central vertex when $D$ is even, or as a new extremity of the central edge when $D$ is odd. In the odd case, one of the extremities of the central edge is randomly chosen to be replaced and is called here the old extremity. Afterwards, the former central vertex when $D$ is even (resp. the old extremity of the former central edge when $D$ is odd) is attempted to be feasibly reconnected to the resulting tree through a least cost criterion. These two perturbations are executed alternately at every iteration of the hybrid algorithm, with the first perturbation being applied first.

For each type of perturbation, a limit is imposed on the number of consecutive perturbation rounds during the ILS phase of the hybrid algorithm. For the first type of perturbation, this limit is equal to the degree of the central vertex when $D$ is even (resp. degree of the replaced extremity of the central edge when $D$ is odd). For the second type of perturbation, up to $|V|/3$ consecutive perturbation rounds are allowed. Local search is applied to the solution obtained after the perturbation. The resulting solution replaces the incumbent as the new current solution if its cost is better than that of the latter.

### 3.5 Filters

Two different filters are used in our hybrid algorithm. The first one is used in the constructive phase of the procedure. As mentioned before, sparse graph instances are treated as if they originate from complete graphs. This is accomplished by introducing missing edges into the corresponding sparse graph, at a sufficiently high cost. However, in this case, the output of the constructive heuristic may eventually contain artificial edges. If that is the case, we would hope that such edges would be eliminated through the use of the local search procedures. However, it was experimentally noticed that, after local search procedures are applied, solutions containing a large number of artificial edges would frequently lead to infeasible solutions or else to feasible solutions with a poor quality.

As a result, a filter is applied to impose a cutoff point for solutions to be submitted to local search. Accordingly, only those solutions containing less than $|V|/3$ artificial edges are submitted to local search. Rejected solutions are simply discarded and a new feasible solution is then attempted to be built through OTT-M. The best solution is retained if no solution with less than $|V|/3$ artificial edges is obtained after ten trials.

Another filter is applied immediately after the perturbations are carried out. As before, the filter decides if the solution in hand is fit to be submitted to local search. Accordingly, only solutions whose cost is at most $(1 + maxtarget)$ times that of the best solution found are submitted to local search, where $maxtarget$ is a given parameter. Conversely, rejected solutions are submitted, yet again, to perturbations. The initial *maxtarget* value is eventually adjusted throughout the execution of the algorithm, being increased by 2% after 100 consecutive feasible solutions are generated without improving the best solution value so far obtained. That toll should include solutions obtained at the constructive and the perturbation phases of the algorithm. A similar scheme was previously used in [30].

### 3.6 Complete algorithm

We denote by $cost(S)$ the cost of a solution $S$. The pseudo-code of the hybrid heuristic is presented in Algorithm 3.

The best solution $S^*$ is initialized in line 1. The loop in lines 2 to 38 is performed until the stopping criterion is attained. The GRASP component of the hybrid heuristic appears in lines 3 to 18. If the graph is dense, a unique solution is built in line 4 using a constructive heuristic and submitted to the local search phase in line 19. Otherwise, in case the graph is sparse, at most ten trials are made in lines 6 to 18 to build an initial solution. The best solution generated in the construction phase is initialized in line 7. The loop in lines 8 to 16 makes at most ten attempts to build a solution with at most $|V|/3$ artificial edges. Any of the constructive heuristics may be used in line 9, before the acceptance criterion is applied in lines 10 to 12. The best over the ten generated solutions ($S'$) is saved in lines 13 to 15.

The solution resulting from the construction phase ($S$) is saved in line 17. Local search is applied to that solution in line 19. The best solution found so far ($S^*$) is updated in lines 20 to 22.

The ILS phase of each iteration of the hybrid algorithm is performed in lines 23 to 37. The loop in lines 23 to 34 implements a perturbation in line 24 to the current solution until an iteration limit is attained. If the relative gap between the cost of the solution $S'$ thus obtained and the best solution found so far $S^*$ is less than the filter value *maxtarget*, then $S'$ is submitted in line 26 to local search. The resulting solution $S'$ is accepted as the new incumbent solution in lines 27 and 28 if it is better than the current incumbent $S$. The best solution so far obtained $S^*$ is replaced by $S'$ in lines 29 and 31 if the latter improves on the former. Finally, the test in lines 35 to 37 increases the filter value *maxtarget* by 2% after 100 consecutive iterations without an improvement on the best solution value obtained so far. As the output, the overall best solution found $S^*$ is returned by the hybrid algorithm in line 39.

```
    Input: stopping criterion, number of perturbations, seed, maxtarget
    Output: S*
 1  Initialize S* ← nil;
 2  while stopping criterion not met do
 3      if graph G is complete then
 4          S ← Construction(seed);
 5      end
 6      else
 7          S' ← nil;
 8          for trials = 1, ..., 10 do
 9              S ← Construction(seed);
10              if S has more than |V|/3 artificial edges then
11                  Go to line 16;
12              end
13              if S improves upon S' then
14                  S' ← S;
15              end
16          end
17          S ← S';
18      end
19      S ← Local search(S);
20      if (S improves upon S*) then
21          S* ← S;
22      end
23      while perturbation iterations limit not met do
24          S' ← Perturbation(S);
25          if (cost(S') − cost(S*))/cost(S*) ≤ maxtarget then
26              S' ← Local search(S');
27              if S' improves upon S then
28                  S ← S';
29                  if S' improves upon S* then
30                      S* ← S';
31                  end
32              end
33          end
34      end
35      if S* not improved after 100 iterations then
36          maxtarget ← maxtarget + 0.02;
37      end
38  end
39  return S*;
```

**Algorithm 3**: Pseudo-code of the hybrid heuristic.

## 4 Computational results

Computational experiments were performed on a Pentium IV machine with a 2.0 G Hz clock and 512 MB of RAM memory. Three test sets of instances were used in the experiments.

The first set was used in [31]. The second set was taken from [11] and was also used in [13]. Instances in the first and second sets were similarly generated. The first test set contains instances that originate from either complete or sparse graphs and have associated diameter values $D$ that could irrespectively be odd or even. For the second test set, all instances originate from sparse graphs and have odd $D$ diameter values associated with them. Instances in this test set have shown to be difficult to solve to optimality, because DCMST formulations tend to perform badly for the odd $D$ case [13, 14, 31]. Optimal solutions for the instances in the first and in the second sets were obtained, respectively, in [31] and [13]. They are used here to evaluate the quality of the solutions found by the hybrid heuristic.

The third test set is formed by Euclidean Steiner problem instances taken from the OR-Library [5]. Fifteen different instances are available for each value of $n$ in $\{50, 70, 100, 250, 500, 1000\}$. For each of these values, only the first five instances available were used in our experiments. Such instances were previously used in [14, 15, 21, 26], under different values of $D$. They are defined over complete graphs and optimality of the corresponding best solutions so far generated was not yet proved.

### 4.1 Constructive heuristics

We first compare the three constructive heuristics (OTT, OTT-M, and RGH). The following metrics were used:

- For each heuristic, we denote its *absolute rank* the number of instances for which it found the best overall result. If, for a given instance, two or more heuristics draw for best result, they all count as best.
- An *average rank* is calculated for every heuristic tested and every test set considered. For a given instance, the heuristic that returns the best result is assigned one point, the one that follows is assigned two points, and the one that returns the worst result is assigned three points. Whenever the three heuristics draw for best result, they are all assigned one point. Whenever two heuristics draw for second best, they are both assigned two points. For every heuristic and every test set, the corresponding *average rank* is equal to the average number of points assigned to the heuristic. The smaller the average rank, the better the corresponding heuristic is.
- For each instance in the first and in the second test sets, the gap between its optimal solution value and the upper bound provided by each heuristic tested are computed. An *average gap* for the results obtained is then computed for each heuristic and every test set. For the third test set, given that proven optimal solution values are unavailable, the corresponding best solution values found in the literature are used instead.

In this experiment, each heuristic tested was applied 1000 times to each instance. Computational results are shown in Tables 1 to 3. The best results in each table are highlighted in bold face. For all test sets, OTT-M performed consistently better than the other two heuristics. It was indicated in [26] that RGH performs well for instances in the third test set, while OTT performs poorly. We obtained similar results. However, in our experiments, OTT-M turned out to perform even better than RGH.

| Heuristics | Absolute rank | Average rank | Average gap |
|---|---|---|---|
| OTT | 6 | 2.6 | 18.26% |
| RGH | 5 | 2.2 | 6.60% |
| OTT-M | **39** | **1.2** | **1.47%** |

**Table 1.** Results for the first test set.

| Heuristics | Absolute rank | Average rank | Average gap |
|---|---|---|---|
| OTT | 3 | 2.33 | 42.75% |
| RGH | 0 | 2.46 | 38.92% |
| OTT-M | **10** | **1.21** | **17.38%** |

**Table 2.** Results for the second test set.

| Heuristics | Absolute rank | Average rank | Average gap |
|---|---|---|---|
| OTT | 0 | 3.00 | 267.22% |
| RGH | 13 | 1.57 | 17.42% |
| OTT-M | **17** | **1.43** | **17.28%** |

**Table 3.** Results for the third test set.

The robustness of OTT-M must be emphasized. Indeed, in spite of the fact that the characteristics of the three test sets are quite different from each other, OTT-M performed uniformly well on all of them. Therefore, OTT-M is taken as the hybrid algorithm constructive heuristic and is used in all experiments that follow.

### 4.2 Hybrid heuristic

The main objective of the experiments reported in this section was to evaluate the quality of the solutions returned by the hybrid algorithm. We also attempted, whenever possible, to compare results obtained by the hybrid heuristic with those reported in the literature.

The main parameter $\alpha$ of the GRASP phase of the algorithm is automatically set by the reactive strategy presented in Section 3.2. Different values for the filter value *maxtarget* were considered in the ILS phase: 5%, 10%, 15%, and 20%. We performed 500 iterations of the hybrid algorithm on instances of the first, second, and third test sets for fine tuning the best *maxtarget* value. Table 4 shows the results obtained for these instances. The last four columns display, for each instance, the cost of the best solution found under each different *maxtarget* value used. The best results obtained are highlighted in bold face. The algorithm performed better with $maxtarget = 0.15$ for all but two of the instances in Table 4. Therefore, this value was selected to be used on the remaining experiments.

| | | | | | *maxtarget* | | | |
|---|---|---|---|---|---|---|---|---|
| Test set | $\|V\|$ | $\|E\|$ | $D$ | id. | 5% | 10% | 15% | 20% |
| First | 25 | 300 | 4 | | **500** | **500** | **500** | **500** |
| | 20 | 50 | 4 | | **442** | **442** | **442** | **442** |
| | 40 | 100 | 4 | | **755** | **755** | **755** | **755** |
| | 60 | 150 | 5 | | 990 | **983** | 989 | 989 |
| Second | 60 | 600 | 4 | | 128 | 127 | **124** | **124** |
| | 60 | 600 | 5 | | **967** | **967** | **967** | **967** |
| | 60 | 600 | 7 | | 158 | 156 | **155** | **155** |
| | 60 | 600 | 7 | | 804 | 804 | **800** | 803 |
| Third | 70 | 2415 | 7 | 1 | **7.23** | 7.24 | **7.23** | **7.23** |
| | 70 | 2415 | 7 | 2 | 7.09 | 7.10 | **7.08** | **7.08** |
| | 70 | 2415 | 7 | 5 | 7.26 | 7.26 | **7.25** | 7.28 |
| | 100 | 4950 | 10 | 1 | **7.83** | 7.89 | 7.89 | 7.90 |
| | 100 | 4950 | 10 | 3 | 8.01 | 8.01 | **8.00** | **8.00** |
| | 100 | 4950 | 10 | 5 | **8.21** | 8.24 | **8.21** | 8.25 |
| | 250 | 31125 | 15 | 3 | 12.17 | 12.21 | **12.13** | 12.18 |
| | 250 | 31125 | 15 | 4 | 12.79 | 12.74 | **12.70** | 12.72 |
| | 250 | 31125 | 15 | 5 | 12.53 | 12.62 | **12.44** | **12.44** |

**Table 4.** Calibration of the filter parameter *maxtarget*.

Five runs of the hybrid algorithm, each one under a different seed, were performed using a run time limit as the stopping criterion. Time limits were set to allow at least 500 iterations for all instances with the same number of vertices in the same test set. For the complete graph instances in the first test set, a 0.1 second time limit was imposed on the 10 vertices instances. Corresponding figures for instances with 20, 40, and 60 vertices were set, respectively, to 0.40, 0.50, and 1 second. For the sparse graph instances in the first set, 5, 10, and 20 seconds were imposed as the time limits for instances with, respectively, 20, 40, and 60 vertices. Time limits of 10 and 50 seconds were imposed to instances of the second test set with, respectively, 40 and 60 vertices. Finally, time limits of 10, 50, 200, 1500, 3000, and 4500 seconds were imposed to instances of the third test set with, respectively, 50, 70, 100, 250, 500, and 1000 vertices.

Computational results for the experiment described above are presented in Tables 5 to 7. Each table line corresponds to a different instance. Number of vertices, number of edges, and diameter values $D$ are given for each instance. Corresponding optimal values $cost(S^*)$ are indicated in Tables 5 to 6 for the instances of the first and in the second test sets. We also indicate in these tables the least time in seconds quoted in the literature to solve each of these instances to proven optimality. We notice that the first and second test sets have been used in [12, 31] exclusively to test and compare different problem formulations. They have not been used before as a test bed for the evaluation of DCMST heuristics. For each instance in the third test set, the corresponding best solution value known $cost(S')$ is taken from Gruber and Raidl [15] (VNS heuristic) and Raidl and Julstrom [21, 26] (evolutionary algorithm) and is given on Table 7 (recall that proven optimal solution values for these instances are still unknown). For this table, a tag is used to match each instance with those in the OR-Library. The last columns in Tables 5 to 7 give, for each instance, the value of the best solution obtained over five runs, the relative gap between the latter and the optimal solution value (only for the instances in the first and second test sets), the average value of the solutions obtained over five runs, and the time limit in seconds used as the stopping criterion.

The results in Table 5 show that the hybrid algorithm managed to find optimal solutions for 41 out of the 42 instances in the first test set. The optimum was not found only for the instance with $|V| = 60$, $|E| = 150$, and $D = 5$. The algorithm found optimal solutions for 39 out of these 42 instances in all five runs (i.e., except for two instances: $|V| = 20$, $|E| = 190$, and $D = 10$; $|V| = 40$, $|E| = 100$, and $D = 7$; $|V| = 60$, $|E| = 150$, and $D = 5$;). The CPU times quoted in the literature to attain proven optimal solutions for these instances using the same computational resources are much higher than those quoted for the hybrid heuristic, see [31].

Table 6 shows that the best solutions found over five runs of the hybrid heuristic for the instances in the second test set are at most at 1.4% from their corresponding optimal solution values. Furthermore, optimal solutions were found for eight out of the 12 instances in the set.

Finally, bold face highlighted results in Table 7 indicate, for 15 out of the 30 instances in the third test set, that the hybrid algorithm either matched or else improved upon corresponding best results taken from the literature. However, the performance of the hybrid algorithm cannot be directly compared with those algorithms that produced the best solutions known to date, since different computational resources were used in the experiments. Nevertheless, we notice that the hybrid heuristic returned best solution values known for 50% of the instances in the third test set. This is particularly interesting if one considers that those are among the largest and hardest instances, for which optimality is not proven.

| $\|V\|$ | $\|E\|$ | $D$ | $cost(S^*)$ | seconds | best | gap (%) | average | seconds |
|---|---|---|---|---|---|---|---|---|
| | | 4 | 252 | 0.77 | 252 | 0.00 | 252.00 | |
| | | 5 | 230 | 0.13 | 230 | 0.00 | 230.00 | |
| | | 6 | 221 | 0.08 | 221 | 0.00 | 221.00 | |
| 10 | 45 | 7 | 203 | 0.14 | 203 | 0.00 | 203.00 | 0.1 |
| | | 8 | 232 | 0.05 | 232 | 0.00 | 232.00 | |
| | | 9 | 254 | 0.10 | 254 | 0.00 | 254.00 | |
| | | 10 | 254 | 0.06 | 254 | 0.00 | 254.00 | |
| | | 4 | 346 | 24.17 | 346 | 0,00 | 346.00 | |
| | | 5 | 331 | 22.27 | 331 | 0,00 | 331.00 | |
| | | 6 | 314 | 32.53 | 314 | 0,00 | 314.00 | |
| 15 | 105 | 7 | 303 | 19.23 | 303 | 0,00 | 303.00 | 0.4 |
| | | 8 | 240 | 45.64 | 240 | 0,00 | 240.00 | |
| | | 9 | 290 | 7.80 | 290 | 0,00 | 290.00 | |
| | | 10 | 286 | 8.95 | 286 | 0,00 | 286.00 | |
| | | 4 | 349 | 1,888.02 | 349 | 0.00 | 349.00 | |
| | | 5 | 414 | 216.36 | 414 | 0.00 | 414.00 | |
| | | 6 | 298 | 593.91 | 298 | 0.00 | 298.00 | |
| 20 | 190 | 7 | 333 | 5.05 | 333 | 0.00 | 333.00 | 0.5 |
| | | 8 | 331 | 1,338.41 | 331 | 0.00 | 331.00 | |
| | | 9 | 327 | 91.33 | 327 | 0.00 | 327.00 | |
| | | 10 | 324 | 172.81 | 324 | 0.00 | 324.40 | |
| | | 4 | 500 | 158,836.45 | 500 | 0.00 | 500.00 | |
| | | 5 | 429 | 51,551.8 | 429 | 0.00 | 429.00 | |
| | | 6 | 378 | 5,194.90 | 378 | 0.00 | 378.00 | |
| 25 | 300 | 7 | 408 | 16,617.61 | 408 | 0.00 | 408.00 | 1.0 |
| | | 8 | 369 | 30,080.28 | 369 | 0.00 | 369.00 | |
| | | 9 | 336 | 9,172.04 | 336 | 0.00 | 336.00 | |
| | | 10 | 379 | 459.88 | 379 | 0.00 | 379.00 | |
| | | 4 | 442 | 0.64 | 442 | 0.00 | 442.00 | |
| | | 5 | 381 | 5.58 | 381 | 0.00 | 381.00 | |
| | | 6 | 329 | 10.81 | 329 | 0.00 | 329.00 | |
| 20 | 50 | 7 | 362 | 2.23 | 362 | 0.00 | 362.00 | 5.0 |
| | | 8 | 366 | 29.93 | 366 | 0.00 | 366.00 | |
| | | 9 | 362 | 26.16 | 362 | 0.00 | 362.00 | |
| | | 10 | 359 | 74.15 | 359 | 0.00 | 359.00 | |
| | | 4 | 755 | 54.14 | 755 | 0.00 | 755.00 | |
| | | 5 | 729 | 20.67 | 729 | 0.00 | 729.00 | |
| | | 6 | 599 | 909.95 | 599 | 0.00 | 599.00 | |
| 40 | 100 | 7 | 667 | 207.64 | 667 | 0.00 | 667.40 | 10.0 |
| | | 8 | 580 | 161,592.78 | 580 | 0.00 | 580.00 | |
| | | 9 | 552 | 4,584.19 | 552 | 0.00 | 552.00 | |
| 60 | 150 | 5 | 968 | 11,644.75 | 983 | 1.55 | 990.60 | 20.0 |

**Table 5.** Results for the instances of the first test set.

| $\lvert V\rvert$ | $\lvert E\rvert$ | $D$ | $cost(S^*)$ | seconds | best | gap (%) | average | seconds |
|---|---|---|---|---|---|---|---|---|
| 40 | 400 | 5 | 612 | 224.00 | 612 | 0.00 | 613.60 | 10.0 |
| | | 7 | 527 | 962.00 | 527 | 0.00 | 530.20 | |
| | | 9 | 495 | 31,349.00 | 495 | 0.00 | 495.40 | |
| | | 5 | 253 | 1,719.00 | 253 | 0.00 | 253.00 | |
| | | 7 | 171 | 92.00 | 171 | 0.00 | 171.00 | |
| | | 9 | 154 | 2.93 | 154 | 0.00 | 154.00 | |
| 60 | 600 | 5 | 965 | 1,600.00 | 965 | 0.00 | 966.60 | 50.0 |
| | | 7 | 789 | 2,417.00 | 796 | 0.89 | 800.40 | |
| | | 9 | 738 | 51,841.00 | 741 | 0.41 | 744.40 | |
| | | 5 | 256 | 13,426.00 | 257 | 0.39 | 257.80 | |
| | | 7 | 150 | 3,062.00 | 152 | 1.33 | 155.00 | |
| | | 9 | 124 | 3,768.21 | 124 | 0.00 | 126.00 | |

**Table 6.** Results for the instances of the second test set.

## 5 Concluding remarks

We developed new heuristics for the diameter constrained minimum spanning tree problem. The OTT-M semi-greedy heuristic is an extension of the OTT constructive heuristic which makes use of a restricted candidate list to select the edge that will be placed into the current partial solution at each iteration. It also employs a reactive strategy for parameter tuning. OTT-M performed better than OTT and RGH in the computational experiments carried out, finding solutions systematically better for the instances in three test sets taken from the literature.

A hybrid heuristic combining the main principles of the GRASP and ILS metaheuristics was also proposed to solve DCMST. This algorithm obtained solutions within 2% of optimality for all instances in the first two test sets considered in this study, at a small fraction of the time required by exact algorithms to obtain proven optimal solutions. For instances originating from the OR-Library, proven optimal solutions have not yet been obtained. However, our hybrid algorithm improved upon or matched the best results in the literature for 15 out of 30 problems. The hybrid heuristic is very robust, since it performed consistently well and improved the best solutions in the literature to date for three fairly different test sets. We notice that the VNS algorithm proposed in [14] was tested on some OR-Library instances, but not on those of the first and second test sets (for which optimal solution values are known).

## References

1. A. M. Abdalla. *Computing a diameter-constrained minimum spanning tree.* PhD thesis, University of Central Florida, College of Engineering and Computing Science, Orlando, 2001.

| $\vert V\vert$ | $\vert E\vert$ | $D$ | tag | $cost(S')$ | best | average | seconds |
|---|---|---|---|---|---|---|---|
| 50 | 1225 | 5 | 1 | 7.60 | **7.60** | 7.60 | 10.0 |
| | | | 2 | 7.68 | **7.61** | 7.62 | |
| | | | 3 | 7.24 | **7.24** | 7.24 | |
| | | | 4 | 6.59 | **6.59** | 6.59 | |
| | | | 5 | 7.32 | **7.25** | 7.25 | |
| 70 | 2415 | 7 | 1 | 7.23 | **7.23** | 7.23 | 50.0 |
| | | | 2 | 7.12 | **7.08** | 7.08 | |
| | | | 3 | 6.99 | **6.98** | 6.99 | |
| | | | 4 | 7.52 | **7.50** | 7.51 | |
| | | | 5 | 7.27 | **7.25** | 7.25 | |
| 100 | 4950 | 10 | 1 | 7.76 | 7.83 | 7.88 | 200.0 |
| | | | 2 | 7.85 | 7.94 | 7.96 | |
| | | | 3 | 7.90 | 7.98 | 8.00 | |
| | | | 4 | 7.98 | 8.04 | 8.08 | |
| | | | 5 | 8.17 | 8.21 | 8.22 | |
| 250 | 31125 | 15 | 1 | 12.30 | 12.45 | 12.53 | 1500.0 |
| | | | 2 | 12.02 | 12.31 | 12.36 | |
| | | | 3 | 12.04 | 12.13 | 12.18 | |
| | | | 4 | 12.51 | 12.70 | 12.75 | |
| | | | 5 | 12.28 | 12.44 | 12.51 | |
| 500 | 124750 | 20 | 1 | 16.97 | 17.20 | 17.25 | 3000.0 |
| | | | 2 | 16.88 | 17.06 | 17.25 | |
| | | | 3 | 16.98 | 17.22 | 17.28 | |
| | | | 4 | 16.99 | 17.24 | 17.31 | |
| | | | 5 | 16.57 | 16.94 | 17.02 | |
| 1000 | 499500 | 25 | 1 | 25.18 | **24.66** | 24.83 | 4500.0 |
| | | | 2 | 25.02 | **24.46** | 24.77 | |
| | | | 3 | 24.82 | **24.32** | 24.56 | |
| | | | 4 | 25.29 | **24.61** | 24.71 | |
| | | | 5 | 25.03 | **24.27** | 24.50 | |

**Table 7.** Results for the instances of the third test set.


2. N. R. Achuthan, L. Caccetta, P. A. Caccetta, and J. F. Geelen. Algorithms for the minimum weight spanning tree with bounded diameter problem. In K. H. Phua, C. M. Wand, W. Y. Yeong, T. Y. Leong, H. T. Loh, K. C. Tan, and F.S. Chou, editors, *Optimization Techniques and Applications*, volume 1, pages 297–304, Singapore, 1992.
3. N. R. Achuthan, L.Caccetta, P. A. Caccetta, and J. F. Geelen. Computational methods for the diameter restricted minimum weight spanning tree problem. *Australasian Journal of Combinatorics*, 10:51–71, 1994.
4. K. Bala, K. Petropoulos, and T. E. Stern. Multicasting in a linear lightwave network. In *Proceedings of the IEEE INFOCOM'93 Conference on Computer Communications*, volume 3, pages 1350–1358, São Francisco, 1993.

5. J. E. Beasley. OR-Library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41:1069–1072, 1990.
6. A. Bookstein and S.T. Klein. Compression of correlated bitvectors. *Information Systems*, 16:110–118, 2001.
7. N. Deo and A. Abdalla. Computing a diameter-constrained minimum spanning tree in parallel. *Lecture Notes in Computer Science*, 1767:17–31, 2000.
8. A.R. Duarte, C.C. Ribeiro, S. Urrutia, and E.H. Haeusler. Referee assignment in sports leagues. In *Practice and Theory of Automated Timetabling VI*, volume 3867 of *Lecture Notes in Computer Science*, pages 158–173. Springer, 2007.
9. T.A. Feo and M.G.C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.
10. M.R. Garey and D.S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. W. H. Freeman, New York, 1979.
11. L. Gouveia and T. L. Magnanti. Modelling and solving the diameter-constrained minimum spanning tree problem. Technical report, Departamento de Estatística e Investigação Operational, Faculdade de Ciências, Lisboa, 2000.
12. L. Gouveia and T. L. Magnanti. Network flow models for designing diameter-constrained minimum-spanning and steiner trees. *Networks*, 41:159–173, 2003.
13. L. Gouveia, T. L. Magnanti, and C. Requejo. A 2-path approach for odd-diameter-constrained minimum spanning and steiner trees. *Networks*, 44:254–265, 2004.
14. M. Gruber and G. R. Raidl. A new 0-1 ILP approach for the bounded diameter minimum spanning tree problem. In L. Gouveia and C. Mour ao, editors, *Proceedings of the 2nd International Network Optimization Conference*, volume 1, pages 178–185, Lisbon, 2005.
15. M. Gruber and G. R. Raidl. Variable neighborhood search for the bounded diameter minimum spanning tree problem. In P. Hansen, N. Mladenovic, J. A. M. Pérez, B. M. Batista, and J. M. MorenoVega, editors, *Proceedings of the 18th Mini Euro Conference on Variable Neighborhood Search*, pages 1–11, Tenerife, 2005.
16. A.C.B. Guedes and C.C. Ribeiro. A hybrid heuristic for minimizing weighted carry-over effects in round robin tournaments, 2009. Submitted for publication.
17. G.Y. Handler. Minimax location of a facility in an undirected graph. *Transportation Science*, 7:287–293, 1978.
18. P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130:449–467, 2001.
19. P. Hansen and N. Mladenović. Developments of variable neighborhood search. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 415–439. Kluwer Academic Publishers, Boston, 2002.
20. J. P. Hart and A. W. Shogan. Semi-greedy heuristics: An empirical study. *Operations Research Letters*, 6:107–114, 1987.
21. B. Julstrom and G. R. Raidl. A permutation-coded evolutionary algorithm for the bounded-diameter minimum spanning tree problem. In A. Barry, F. Rothlauf, and D. Thierens et al., editors, *The Genetic and Evolutionary Computation Conference's Workshops Proceedings, Workshop on Analysis and Desgn of Representations*, pages 2–7, Chicago, 2003.
22. B. A. Julstrom. Encoding bounded-diameter minimum spanning trees with permutations and with random keys. *Lecture Notes in Computer Science*, 3102:1272–1281, 2004.
23. H. R. Lourenço, O. C. Martin, and T. Stutzle. Iterated local search. In F. Glover and G. Kochenberger, editors, *Handbook of metaheuristics*, pages 321–353. Kluwer Academic Publishers, Boston, 2002.

24. M. Prais and C. C. Ribeiro. Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS Journal on Computing*, 12:164–176, 2000.
25. R. C. Prim. Shortest connection networks and some generalizations. *Bell Systems Technical Journal*, 36:1389–1401, 1957.
26. 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 18th ACM Symposium on Applied Computing*, pages 747–752, Melbourne (USA), 2003.
27. K. Raymond. A tree-based algorithm for distributed mutual exclusion. *ACM Transactions on Computers*, 7:61–77, 1989.
28. C. Requejo. *Modelos para o problema da árvore geradora de suporte com restrição de diâmetro: caso do diâmetro ímpar*. PhD thesis, Universidade de Lisboa, Faculdade de Ciências, Lisboa, 2003.
29. M. G. C. Resende and C. C. Ribeiro. Greedy randomized adaptive search procedures. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 219–249. Kluwer Academic Publishers, Boston, 2003.
30. C. C. Ribeiro and S. Urrutia. Heuristics for the mirrored traveling tournament problem. *European Journal of Operational Research*, 179:775–787, 2007.
31. A. C. Santos, A. Lucena, and C. C. Ribeiro. Solving diameter constrained minimum spanning tree problem in dense graphs. *Lecture Notes in Computer Science*, 3059:458–467, 2004.