Applied Mathematics and Computation 218 (2012) 11597–11612

Contents lists available at SciVerse ScienceDirect

Applied Mathematics and Computation

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

Solving the Quadratic Minimum Spanning Tree Problem

Roberto Cordone [a,*], Gianluca Passeri [b]

[a] Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, Via Comelico 39, 20135-Milano, Italy
[b] Dipartimento di Tecnologie dell'Informazione, Università degli Studi di Milano, Via Bramante 65, 26013-Crema, Italy

ARTICLE INFO

Keywords:
Quadratic Minimum Spanning Tree Problem
Binary quadratic programming
Tabu Search
Variable Neighbourhood Search
Branch-and-bound

ABSTRACT

Given an undirected graph with costs associated to its edges and pairs of edges, the *Quadratic Minimum Spanning Tree Problem* (*QMSTP*) requires to determine a spanning tree of minimum total cost. This is a proper model for network problems in which both routing and interference costs need to be considered. It is $\mathcal{NP}$-hard in the strong sense and not approximable unless $\mathcal{P} = \mathcal{NP}$. This paper describes a Tabu Search algorithm, with two independent and adaptively tuned tabu lists, and a Variable Neighbourhood Search algorithm. Both metaheuristics are based on the same neighbourhood, but the Tabu Search proves more effective and robust than the Variable Neighbourhood Search. To assess the quality of these results, we provide a comparison with the heuristic algorithms proposed in the recent literature and we reimplement, with minor improvements, an exact algorithm drawn from the literature, which confirms the optimality of the results obtained on small instances.

1. Introduction

Consider a connected undirected graph $G = (V, E)$ with $|V| = n$ vertices and $|E| = m$ edges, a linear cost function $c : E \to \mathbb{R}$ defined on the edges and a quadratic cost function $q : E \times E \to \mathbb{R}$ defined on the pairs of edges. The *Quadratic Minimum Spanning Tree Problem* (*QMSTP*) requires to determine a spanning tree $T = (V, X)$ minimizing the sum of the linear costs for all edges in X plus the quadratic costs for all pairs of edges in X. Without loss of generality, we assume $q_{ef} = q_{fe}$ for all $e, f \in E$ and $q_{ee} = 0$ for all $e \in E$.

A natural integer programming formulation for the *QMSTP* is:

$$\min z = \sum_{e \in E} c_e x_e + \sum_{e \in E} \sum_{f \in E} q_{ef} x_e x_f \tag{1a}$$

$$\sum_{e \in E} x_e = n - 1 \tag{1b}$$

$$\sum_{e \in E(S)} x_e \leqslant |S| - 1 \quad S \subset V : |S| \geqslant 3 \tag{1c}$$

$$x_e \in \{0, 1\} \quad e \in E \tag{1d}$$

where $x_e = 1$ if edge e belongs to the solution, $x_e = 0$ otherwise, and $E(S)$ denotes the set of edges with both ends in S.

The first contribution of this paper is a Tabu Search algorithm based on the exchange of edges. The tabu mechanism exploits two independent tabu lists, which manage the insertion and removal of edges with respect to the current solution; their length varies adaptively depending on the results of the search. The second contribution of the paper is a Variable

* Corresponding author.
E-mail address: roberto.cordone@unimi.it (R. Cordone).

0096-3003/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.amc.2012.05.043

Neighbourhood Search algorithm, based on the same neighbourhood as the Tabu Search. In a direct comparison, the Tabu Search algorithm proves more robust. Since the recent literature proposes a number of heuristic algorithms evaluated on different benchmark instances, we compare our Tabu Search with these algorithms on all the available benchmarks [18,5,15,19,16]. Thanks to the efficient data structures used (see Section 4.1), the Tabu Search equals or improves all best known results, usually taking a shorter or equal time with respect to its competitors.

In order to give an absolute assessment of the performance of the algorithms here proposed, we reimplement the Lagrangean branch-and-bound algorithm proposed in [3], with some minor improvements: Lagrangean penalties to fix edges in or out of the solution, a weaker (but faster to solve) combinatorial relaxation and an anticipated stopping rule for the computation of the lower bound. The exact algorithm confirms the optimality of the best solution found by Tabu Search for all instances up to 15 vertices and for the sparse instances with 20 vertices. Moreover, when applied to benchmark instances up to 20 vertices, the combinatorial branching approach provides better bounds than a Lagrangean relaxation recently proposed in the literature, which is theoretically tighter, but much slower to compute.

Section 2 presents some practical applications of the *QMSTP* and surveys the related literature. Section 3 discusses its computational complexity and approximability. The following ones introduce the Tabu Search (Section 4), the Variable Neighbourhood Search (Section 5) and the branch-and-bound algorithm (Section 6). Finally, Section 7 discusses the experimental results.

2. Applications and literature survey

The *QMSTP* has applications in network design, where the linear function models the cost to build or use edges, while the quadratic one models interference costs between the edges in use. This may concern telecommunication networks (in which the edges represent radio or cable links), oil or water transmission networks (in which the edges stand for pipes), transportation networks (in which the edges correspond to roads), etc.. . .. In some cases, the interference costs are limited to pairs of adjacent edges (for example, valves or bending losses in T-joins, conversion devices between cables of different kinds, turn delays), whereas in other cases they may affect any pair of edges in the network (especially when its topology has little relation to the physical layout, as in fiber-optic networks). The *QMSTP* is also equivalent, through a standard transformation in the context of *Chance-Constrained Programming* [4], to a special case of the *Stochastic Minimum Spanning Tree Problem*, as discussed in [3].

Despite the neat structure of this problem, the literature reports a very limited number of approaches until quite recently. In particular, two greedy algorithms were proposed in [21,3,22] and compared to an evolutionary algorithm in [23]. Another evolutionary algorithm was proposed for a fuzzy variation of the *QMSTP* [8,9], representing the solutions with the Prüfer number encoding of trees, whereas the *edge-window decoder* strategy was adopted in [18]. During the review process of the present paper, however, a number of papers have proposed alternative approaches, mostly based on local search. In particular, the Tabu Thresholding algorithm [12] described in [15] alternatively performs local search and random move phases. Iterated Tabu Search is compared to multistart simulated annealing and a hybrid genetic algorithm in [16]. The Artificial Bee Colony algorithm described in [19] represents solutions with the edge-set encoding (also adopted in the hybrid genetic algorithm mentioned above), selects candidate solutions with the binary tournament selection method, generates new ones with a refined mechanism based on random edge swaps and applies local search to improve the best solution found in the end. While several authors test their algorithms on the 6 instances provided by [18] and Palubeckis et al. [16] adopt our benchmark [5], most of the time each algorithm is evaluated on an independent new benchmark.

As for exact methods, the branch-and-bound algorithm described in [3] is based on the use of a linear function to approximate from below the quadratic objective function. Thanks to an update mechanism known as *levelling procedure*, the original linear and quadratic costs are iteratively updated, in such a way that the value of all feasible solutions is unchanged, while the linear approximation becomes monotonically tighter. The process is actually equivalent to a Lagrangean relaxation, with an update mechanism guaranteeing monotonicity. Stronger bounds can be obtained applying the Lagrangean relaxation to an extended formulation with valid inequalities as in [15].

3. Properties

The *QMSTP* is obviously related to the classical Minimum Spanning Tree (*MST*) problem. Contrary to that problem, however, the *QMSTP* is strongly $\mathcal{NP}$-hard. This has been proved in [3] by reduction from the Quadratic Assignment Problem. We here prove it by reduction from the Satisfiability Problem.

Theorem 1. *The QMSTP is $\mathcal{NP}$-hard in the strong sense, even if $c_e = 0$ for all $e \in E$ and $q_{ef} \in \{0,1\}$ for all $(e,f) \in E \times E$.*

Proof. Given an instance of *SAT*, build the following instance of *QMSTP*: graph $G = (V,E)$ has a vertex y_i for each clause and a vertex x_j for each Boolean variable; an edge e is given for each pair of vertices (y_i, x_j) such that literal x_j or $\bar{x}_j$ occurs in clause i and for each pair of vertices (x_j, x_{j+1}) associated to consecutive variables. The linear cost function is identically zero, while $q_{ef} = 1$ for edges $e = (y_i, x_j)$ and $f = (y_l, x_j)$ if variable x_j is affirmed in clause i and negated in clause l or viceversa; $q_{ef} = 0$ for all other pairs of edges.

This instance of *QMSTP* has a solution of zero cost if and only if there is a spanning tree such that all pairs of its edges have zero quadratic cost. The edges (y_i, x_j) can be interpreted as the use of variable x_j to satisfy clause i. Therefore, the zero cost spanning trees exactly correspond to the consistent assignments which satisfy all clauses.

See Fig. 1 for the graph construction corresponding to the Boolean form $(x_1 + x_2 + x_4)\ (\bar{x}_1 + \bar{x}_2)\ \bar{x}_4\ (x_1 + x_3 + \bar{x}_4)$: the bold lines identify the edges of a zero cost spanning tree; the dashed lines correspond to the other edges of the graph. □

From the previous proof, it is possible to derive also that the *QMSTP* is not approximable within a constant factor α in polynomial time, unless in the unlikely circumstance that $\mathcal{P} = \mathcal{NP}$. We remind that a minimization problem is approximable within a constant factor α (in short, it belongs to $\mathcal{APX}$) if and only if there exists a constant value α and a polynomial-time algorithm such that $z \leqslant \alpha z^*$ for all instances of the problem, where z is the value of the solution computed by the algorithm and z^* is the optimal value for the given instance [10].

Corollary 2. *The QMSTP is not approximable, unless* $\mathcal{P} = \mathcal{NP}$.

Proof. Given the graph construction of Theorem 1, the optimum of such a *QMSTP* is zero if and only if the corresponding *SAT* instance is satisfiable. If the optimum is zero, any polynomial algorithm with a constant approximation guarantee equal to α would find a solution of cost $\leqslant \alpha \cdot 0 = 0$, that is an optimal solution. Hence, *QMSTP* $\in \mathcal{APX}$ would imply $\mathcal{P} = \mathcal{NP}$. □

4. Tabu Search

The Tabu Search algorithm *QMST-TS* starts from a random solution $T = (V, X)$, obtained selecting $n - 1$ edges from E, such that they do not form any closed loop. Then, it improves this solution performing moves based on a natural neighbourhood: each move adds to X one of the $(m - n + 1)$ unused edges $e \in E \setminus X$, thus closing a loop L_e of $n' \leqslant n$ edges, and removes an edge f from $L_e \setminus \{e\}$. The number of feasible moves is therefore in $O(mn)$. A classical local search algorithm would replace the current solution with a better one extracted from this neighbourhood and terminate as soon as no such solution exists. Tabu Search, on the contrary, continues the search by visiting also worse solutions. The visit of already explored solutions, however, is prevented by a memory mechanism which forbids some moves labelling them as tabu. The overall process terminates after a number of iterations I specified by the user. We refer the reader to [13] for general information about the Tabu Search methodology. In the following, we briefly describe the specific features of *QMST-TS*, i. e. the tabu mechanism and the data structures which allow to limit its computational complexity.

Basic tabu mechanism. The tabu mechanism derives from the one successfully applied in [1] to the *Maximum Diversity Problem*, which is a binary quadratic optimization problem with some relation to the *QMSTP*. This mechanism exploits two independent tabu lists to forbid both the inclusion of a recently removed edge and the removal of a recently included edge. More specifically, for each edge $e \in E$ we save in an array the last iteration I_e in which edge e was moved into or out of the current solution. When evaluating the insertion of edge e and the removal of edge f, we check whether the current iteration index i is larger than both $I_e + \ell_{in}$ and $I_f + \ell_{out}$; if it is not, the move is declared tabu. Parameters ℓ_{in} and ℓ_{out} denote the length of the prohibition, or *tabu tenure*. The algorithm runs for a total number I of iterations, but it is restarted every I/r iterations from a random solution, so that the number of starting solutions considered is r.

Short term memory. We also adopt a short term memory mechanism to intensify or diversify the search depending on the current results: the two tabu tenures decrease by one if the objective function has improved in the most recent iteration, and they increase by one if the objective function has worsened. The purpose of decreasing the tabu tenure is to intensify the search in those regions which provide improving solutions, and therefore appear more promising. On the contrary, increasing the tabu tenure speeds up the departure from those regions which provide worsening solutions, and therefore probably surround an already visited local optimum. To avoid over-reactions, the values of the tabu tenures are restricted within given ranges, and the updates which would violate these ranges are not performed. More specifically, the tabu tenure ℓ_{in} varies in a

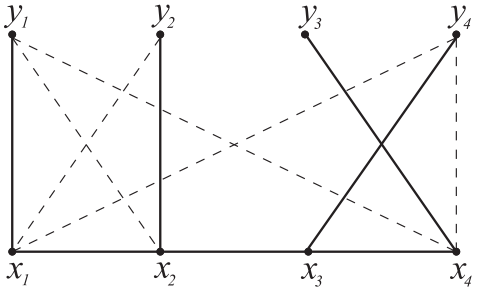

Fig. 1. Graph construction to prove that the QMSTP is $\mathcal{NP}$-hard: the graph corresponds to the Boolean form $(x_1 + x_2 + x_4)\ (\bar{x}_1 + \bar{x}_2)\ \bar{x}_4\ (x_1 + x_3 + \bar{x}_4)$, the bolded edges form a zero cost spanning tree, which corresponds to a satisfying truth assignment.

given range $[\ell_{in}^m, \ell_{in}^M]$, starting from the middle point of this range $\ell_{in}^{(0)} = (\ell_{in}^m + \ell_{in}^M)/2$. As well, ℓ_{out} ranges between ℓ_{out}^m and ℓ_{out}^M, starting from $\ell_{out}^{(0)} = (\ell_{out}^m + \ell_{out}^M)/2$.

Aspiration criterion. If the best feasible move improves the best known result, it is performed, even if it is tabu. In the opposite case, the best non tabu feasible move is performed.

4.1. Worst-case analysis of the neighbourhood exploration

In order to explore the neighbourhood, one needs to check for each pair of edges $e \in E \setminus X$ and $f \in X$ whether it is feasible to introduce e and remove f, i. e. whether f belongs to loop L_e or not, and to evaluate the corresponding variation of the objective function. A trivial implementation would have a relevant computational cost. The feasibility test could be done in $O(n)$ time by visiting the current tree from one of the end vertices of $e = (u, v)$, say u, to see whether f belongs to the unique path between u and v on X. The evaluation of the objective function could also be done in $O(n)$ time, by subtracting the linear and quadratic contributions of f from the objective function and summing those of e. This would lead to an overall complexity of $O(mn^2)$ to explore the whole neighbourhood. Suitable data structures, however, allow to determine efficiently the edges in loop L_e [11] and their contributions to the objective function [1]. The use of these data structures is particularly relevant to explain the performance of the algorithms here proposed with respect to the ones available in the literature.

Proposition 3. *Each solution of the neighbourhood can be evaluated in constant amortized time.*

Proof. The exchange $(e, f) \in (E \setminus X) \times X$, where edge e is inserted and edge f is removed from the solution, is feasible if and only if f belongs to the closed loop L_e. But, instead of testing for each of the $(m - n + 1)\ (n - 1)$ possible pairs whether $f \in L_e$ or not, it is more efficient to determine the loop L_e for each $e \in E \setminus X$ and scan it, thus considering only the feasible exchanges.

It is possible to determine L_e efficiently by representing the current solution as a rooted tree. Given a conventional root vertex r, the rooted tree representation exploits a vector ϕ, which reports for each vertex $v \in V$ the first vertex on the unique path from v to r along the tree. For example, on the left side of Fig. 2, the first vertex on the path from vertex 2 to the root $r = 1$ is $\phi_2 = 5$. The choice of the root is arbitrary and changes during the execution of the algorithm.

When testing the insertion of edge $e = (u, v) \in E \setminus X$, the tree is rerooted in one of its end vertices (say u). This simply amounts to scanning the path from u to the current root r, and reversing it (see Fig. 2, where the root of the tree changes from 1 to 6). Once the tree is rerooted, loop L_e consists of edge e plus the new unique path from v to u along the tree, which vector ϕ easily allows to scan. Thus, it is possible to scan all feasible exchanges for a given $e = (u, v)$ by rerooting the tree in u and exploiting vector ϕ to scan L_e. Both operations require $O(|L_e|) = O(n)$ time, but they allow to test all the $|L_e| - 1$ feasible removed edges. Then, the amortized complexity is constant if it is possible to evaluate in constant time the cost variation induced by each exchange.

To achieve this result, we maintain the following vector D:

$$D_e = c_e + \sum_{g \in X} q_{eg} \quad e \in E$$

whose values can be interpreted as the (actual or potential) contribution of each edge $e \in E$ to the overall cost of the solution. When exchanging edge $e \in E \setminus X$ and $f \in X$, the objective function varies by $\delta_{ef} = D_f - D_e - 2q_{ef}$, which can be computed in constant time. After performing the chosen move, all coefficients D_g must be updated as follows: $D_g := D_g - 2q_{ge} + 2q_{gf}$, which takes $O(m)$ time as a whole, and an amortized $O(1/n)$ time for each solution in the neighbourhood.

The complexity above discussed can be further reduced in the average case by consecutively testing the edges $e \in E \setminus X$ which are incident to the same vertex u. In this way, in fact, the total number of rerootings reduces from $m - n + 1$ to n. □

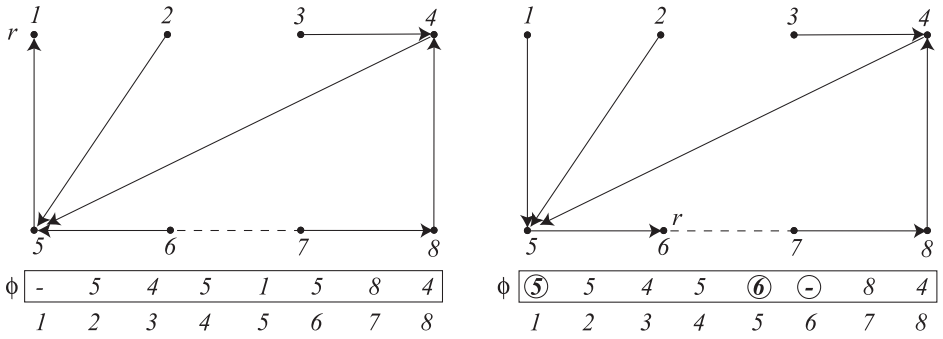

Fig. 2. A tree rooted in vertex 1 (on the left) is rerooted in vertex 6 (on the right) by reversing the unique path (6,5,1) from the new root to the old one and updating the corresponding elements of vector ϕ (circled). Loop L_{67} is now identified by path (7,8,4,5,6).

The overall complexity of a neighbourhood exploration is therefore $O(mn)$, plus an initial $O(n + mn) = O(mn)$ time to initialize vectors ϕ and D.

5. Variable Neighbourhood Search

The Variable Neighbourhood Search (*VNS*) approach requires to define a hierarchy of neighbourhoods $\mathcal{N}_k$, indexed by an integer parameter k, which ranges from $k_{\min}$ to $k_{\max}$ as the size of the neighbourhood increases. In the following, we briefly describe the specific features of our implementation, denoted as *QMST-VNS*; the interested reader can find in [14] general information about the methodology.

Fig. 3 provides a pseudocode of algorithm *QMST-VNS*. This starts from a given solution S^*, generated at random, which is also assumed as a *reference solution*. It sets k to $k_{\min}$ and applies classical local search to the same basic neighbourhood adopted by algorithm *QMST-TS*: the neighbour solutions are the ones obtained with simple edge exchanges, that is by adding an unused edge $e \in E \setminus X$ and removing an edge f from the loop L_e thus created, so that the objective function varies by an amount δ_{ef}. The efficient exploration of this neighbourhood and evaluation of δ_{ef} has been already discussed in Section 4.1. The search terminates when it is no longer possible to find edge exchanges with $\delta_{ef} < 0$. The current solution S is then locally optimal. If it improves the previously best known one, this is updated and k is set back to $k_{\min}$; otherwise, k increases by 1. Every time k exceeds $k_{\max}$, it is set back to $k_{\min}$. Then, a suitable *shaking procedure* extracts a new starting solution S from the neighbourhood $\mathcal{N}_k$ of the reference solution S^*. The process ends after a specified number of local search iterations returning the best known solution.

Neighbourhood hierarchy. Neighbourhood $\mathcal{N}_k$ includes all solutions obtained from the current one by adding at most k new edges and removing the same number of edges, so as to break all the loops thus generated. Notice that necessarily $k \geqslant 1$ and that its maximum reasonable value is $\tilde{k} = \min(n-1, m-n+1)$. In fact, the corresponding neighbourhood $\mathcal{N}_{\tilde{k}}$ coincides with the whole solution space, given that it is not possible to remove more than $n-1$ edges from a spanning tree nor to introduce in it more than $m-n+1$ edges. Therefore, $1 \leqslant k_{\min} < k_{\max} \leqslant \tilde{k}$.

Shaking procedure. This procedure generates a solution in the neighbourhood $\mathcal{N}_k$ of the current one. It selects a random edge (with a uniform distribution) not belonging to the solution and inserts it into the solution, thus closing a loop. Then, it selects a random edge (with a uniform distribution) from the loop and removes it. This process is repeated k times, always avoiding to reinsert a removed edge. If possible, it also avoids removing the freshly inserted edges, but if a whole loop consists of such edges, the second prohibition is relaxed for the sake of simplicity. Consequently, the new starting solution differs from the reference one by *at most*, but possibly less than, k edges. The selection of the newly inserted edge can be performed in constant time keeping in separate parts of a suitable vector the edges in and out of the solution. The selection of the removed edge can be performed in $O(n)$ time thanks to the data structure introduced by Proposition 3. The update of D takes $O(m)$ time. Therefore the overall shaking procedure has $O(km)$ worst-case complexity.

6. Branch-and-bound

The branch-and-bound algorithm *QMST-BB* reimplements the one proposed in [3]. It is based on the relaxation of the quadratic objective function to a linear approximation, so that the relaxed subproblem is a *MST* problem, solved by Kruskal's algorithm.

```
Algorithm QMST-VNS(G, c, q, S*)
S := S*; k := k_min; i := 1;
while (i < I) do
    { Improve S by simple edge exchanges }
    repeat (e, f) := BestEdgeExchange(S, G, c, q);
        if (δ_ef < 0) then S := PerformEdgeExchange(e, f, S, G, c, q); i := i + 1;
    until (δ_ef ≥ 0);
    if c(S) < c(S*) then S* := S; k := k_min; else k := k + 1;
    if k > k_max then k := k_min;
    S := Shaking(S*, k, G, c, q);
return S*;
```

Fig. 3. Pseudocode of *QMST-VNS*.

A combinatorial lower bound. The branching mechanism adopted to generate subproblems fixes a subset X_i of edges into the solution and a subset X_o of edges out of it. Hence, a subproblem can be relaxed by replacing the quadratic objective function with the following linear one:

$$\min \hat{z} = \sum_{e \in E} \hat{c}_e x_e \tag{2}$$

subject to the original constraints (1b)–(1d), plus the additional branching constraints $x_e = 1$ for all $e \in X_i$ and $x_e = 0$ for all $e \in X_o$.

The approximated costs $\hat{c}_e$ are defined as

$$\hat{c}_e = c_e + \sum_{f \in X_i} q_{ef} + \sum_{f \in F_{eX_iX_o}^{(n-2-|X_i|)}} q_{ef} \quad e \in E \tag{3}$$

where $F_{eX_iX_o}^{(k)}$ is the subset of k edges from $E \setminus (\{e\} \cup X_i \cup X_o)$ which do not close loops with $(X_i \cup \{e\})$ and have the minimum quadratic costs with respect to e. The linear estimate $\hat{c}_e$, therefore, includes the linear cost c_e, the quadratic costs with respect to the already fixed edges and the correct number of cheapest quadratic costs with the available unfixed edges which close no loop with e and with the fixed edges. This is a relaxation of the considered subproblem because the feasible solutions are the same (the spanning trees) and in any feasible solution the auxiliary cost $\hat{c}_e$ is not larger than the contribution of edge e to the original objective function. In fact, this contribution would include the first two terms of $\hat{c}_e$ plus a quantity not smaller than the third term. The optimum of the relaxation introduced above can be computed by solving with Kruskal's algorithm a *MST* subproblem with respect to $\hat{c}_e$ and it provides a lower bound on the optimum of the given subproblem. Since the evaluation of $\hat{c}_e$ requires $O(m)$ time for each edge, which dominates the time required to solve the relaxed subproblem, the overall complexity is $O(m^2)$ for each branching node.

A more refined relaxation could be obtained replacing $F_{eX_iX_o}^{(n-2-|X_i|)}$ with the set of $(n-2-|X_i|)$ edges from $E \setminus (\{e\} \cup X_i \cup X_o)$ which have the minimum quadratic costs with respect to e and close no loop with $X_i \cup \{e\}$, *as well as with each other*. This corresponds to computing for each edge e a spanning tree including $X_i \cup \{e\}$ and minimum with respect to cost function q_{ef}. With this approach, the computation of $\hat{c}_e$ requires $O(m \log m)$ time for each edge, dominating the solution of the relaxed problem and yielding an overall $O(m^2 \log m)$ complexity.

The levelling procedure. The following interesting property of all quadratic programming problems with cardinality constraints allows to refine the two bounds described above. This has been done for the latter in [3], whereas we apply it to the former here.

Remark 4 [3]. Given any real function $\pi : E \to \mathbb{R}$, replacing c_e with $c'_e = c_e - (n-2)\pi_e$ and q_{ef} with $q'_{ef} = q_{ef} + \pi_f$ for $e \in E$ and $f \in E \setminus \{e\}$ leaves unchanged the total cost:

$$z'_\pi(x) = \sum_{e \in E} c'_e x_e + \sum_{e \in E} \sum_{f \in E} q'_{ef} x_e x_f = \sum_{e \in E} c_e x_e + \sum_{e \in E} \sum_{f \in E} q_{ef} x_e x_f = z(x)$$

of any solution x subject to Constraints (1b) and (1d).

Therefore, the optimum of the problem is independent from vector π. By contrast, the approximated linear costs $\hat{c}_e$ depend on π and a better choice of π yields a tighter bound. The largest possible bound can be determined by the following iterative procedure. Starting from $\pi_e = 0$ for all $e \in E$, the levelling procedure computes the approximated linear costs $\hat{c}_e(\pi)$ and determines their *deviation* $\delta(\pi) = \max_{e \in E} \hat{c}_e(\pi) - \min_{e \in E} \hat{c}_e(\pi)$, that is the difference between their maximum and minimum values. Then, π_e is updated according to the following formula:

$$\pi_e := \pi_e + \frac{\hat{c}_e(\pi)}{n-1} \quad e \in E$$

It has been proved that the resulting lower bound is nondecreasing, that the deviation is nonincreasing and that the best possible lower bound is always lower than the current lower bound plus $(n-1)$ times the current deviation δ [3]. Therefore, if δ converges to zero, the best possible lower bound has been attained.

Based on our experimental results, we modified two features of the original bounding procedure described in [3]. First, we apply the levelling procedure to compute the simpler bound, in which the linear estimate $\hat{c}_e$ accounts for the correct number of quadratic costs, but neglects the acyclicity requirement, instead of the stronger one. Second, we terminate the bounding procedure as soon as

$$LB(\pi) + (n-1)\delta(\pi) < UB$$

where $LB(\pi)$ is the current value of the lower bound and UB is the best current upper bound on the optimum. In fact, if the upper estimate on the best possible lower bound is strictly lower than the current upper bound, there is no chance that further refinements of the lower bound could lead to prune the current branching node by dominance. The original bounding procedure stopped after a fixed number of iterations (one or two).

Upper bound computation. At each iteration of the levelling procedure, the solution of the relaxed subproblem is a spanning tree, which, evaluated with the original objective function, provides an upper bound. An initial upper bound is also provided by algorithm *QMST-TS* (see Section 4).

Visit strategy and branching rule. We adopt the *best-bound-first* visit strategy: always visit the open branching node with the smallest lower bound.

Branching is performed by fixing an edge in or out of the solution. The branching edge is the cheapest unfixed one which belongs to the solution of the relaxed *MST* problem, as in [3]. The rationale of this choice is that, by fixing that edge into the solution the linear estimated cost of all other edges will include the quadratic cost with respect to it and therefore probably increase; on the other side, by removing that edge, the relaxed solution will certainly be different and more expensive.

Logical reductions. With respect to the original branch-and-bound algorithm in [3], we introduce the following two simple logical tests, which may allow to fix some edges in or out of the optimal solution, respectively augmenting subsets X_i or X_o:

1. if graph $(V, E \setminus (X_o \cup \{e\}))$ is not connected, edge e is added to X_i;
2. if graph $(V, X_i \cup \{e\})$ contains a cycle, edge e is added to X_o.

These tests can be performed in almost linear time with respect to m by using merge-find sets to represent the connected components [7].

Lagrangean penalties. With respect to the original branch-and-bound algorithm in [3], we introduce Lagrangean penalty procedures, which allow to fix in (or out of) the optimal solution an edge belonging to (or not belonging to) the current relaxed solution. Such procedures can be applied to any problem whose relaxation is a *MST* problem (see for example [6]).

- In penalties: for each edge e in the current solution, try to remove it and look for the cheapest edge $e' \neq e$ connecting the two subtrees thus generated; if $LB_\pi - \hat{c}_e + \hat{c}_{e'} \geqslant UB$, then the best solution attainable by removing e cannot improve the best known one: consequently, set $x_e = 1$.
- Out penalties: for each edge e out of the current solution, find the most expensive edge e' in the loop which e would form if added to the solution; if $LB_\pi + \hat{c}_e - \hat{c}_{e'} \geqslant UB$, then the best solution attainable by inserting e cannot improve the best known one: consequently, set $x_e = 0$.

For the sake of computational efficiency, we apply a heuristic version of these procedures: we maintain the cheapest edge out of the current solution and the most expensive one in the solution, and use them instead of e', respectively in the first and second procedure described above.

Useful data structures. To improve the average complexity of all these computations, the edges are stored in a vector, composed of three subvectors: the first and the last one include, respectively, the edges fixed into and out of the solution. The middle subvector includes the unfixed edges and it is managed as a *min-heap* [7], in order to determine in constant time the cheapest edge at each step of Kruskal's procedure without keeping them totally ordered. This reduces the average complexity of the lower bound computation, because Kruskal's procedure can be stopped as soon as a spanning tree has been found, without sorting all of the edges.

Warm start. In order to give the levelling procedure a "warm start", at each node of the branching tree we initialize vector π with the best vector found in the father node ($\pi = 0$ at the root node). This is somewhat contrary to the original spirit of the levelling procedure, which is an ascent approach (i.e., the multipliers always increase). Indeed, the resulting bound is weaker than the one achieved by always starting from scratch, and the total number of branching nodes increases. However, the levelling procedure converges in a lower number of iterations, the branching nodes are solved more quickly and the total computational time is significantly lower.

7. Experimental results

The algorithms described in the previous sections have all been implemented in C language and run on a 2.6 GHz Intel Pentium Core 2 Duo E6700 with 2 GB of RAM. This section describes the benchmark instances. Then it discusses the parameter tuning for algorithms *QMST-TS* and *QMST-VNS* and compares them, showing the superior performance of the former. A further analysis is made on the influence of the starting solution. Then, we present the results of the exact algorithm *QMST-BB*. The following subsection compares *QMST-TS* with the state-of-the-art heuristics proposed in the literature on all the available benchmarks. The final one compares *QMST-BB* with the Lagrangean lower bounding procedure described in [15].

7.1. Benchmark instances

In recent years, several authors have proposed heuristic approaches to the *QMSTP*, validating them on benchmark instances. To the best of our knowledge, the following benchmarks (named after the initials of the authors) are currently available:

- Benchmark SCA [18] includes 6 instances, ranging from 10 to 50 vertices by steps of 10, with complete graphs, vertices uniformly spread in a square of side equal to 500, linear costs equal to the Euclidean distances between the vertices and random quadratic costs uniformly distributed in [0; 20];

- Benchmark CP [5], downloadable from http://www.dti.unimi.it/cordone/research/qmst.html, consists of 108 instances with a number of vertices ranging from 10 to 50 by steps of 5, graphs of different densities ($\rho = 33\%$, 67% or 100%), random linear costs uniformly distributed in [1; 10] or [1; 100] and random quadratic costs uniformly distributed in [1; 10] or [1; 100];
- Benchmark OP1 [15] consists of 480 instances with complete graphs ranging from $n = 6$ to $n = 18$ vertices by steps of 1, and with $n = 20$, $n = 30$ and $n = 50$; they are divided into three subsets:
  1. SYM: the linear costs are uniformly distributed at random in [1; 100], the quadratic ones in [1; 20];
  2. VSYM: the linear costs are uniformly distributed at random in [1; 10 000], the quadratic costs q_{ef} are obtained associating to the vertices random values uniformly distributed in [1; 10] and multiplying the four values associated to the end vertices of edges e and f;
  3. ESYM: having spread the vertices uniformly at random in a square of side 100, the linear costs are the Euclidean distances between the end vertices of each edge and the quadratic costs are the Euclidean distances between the midpoints of the edges;
- Benchmark OP2 [15] consists of the *NUG* and *CHR* benchmarks of the *Quadratic Assignment Problem* (29 instances), converted into *QMSTP* instances by a simple graph transformation which guarantees a one-to-one correspondence between the solutions of the two problems;
- Benchmark SS [19] includes 18 instances with complete graphs of $n = 25, 50, 100, 150, 200$ and 250 vertices, random linear costs uniformly distributed in [1; 100] and random quadratic costs uniformly distributed in [1; 20].

7.2. Parameter tuning for the Tabu Search algorithm

This section considers algorithm *QMST-TS*, described in Section 4, and discusses the tuning of its parameters, which are the total number of iterations I, the number of random starting solutions r and the tabu tenure ranges $[\ell_{in}^m; \ell_{in}^M]$ and $[\ell_{out}^m; \ell_{out}^M]$. These forbid, respectively, the insertion of new edges and the removal of old ones.

In the first experiment, Algorithm *QMST-TS* has been run on Benchmark CP, starting from a single random solution ($r = 1$) for a total number of $I = 100\,000$ iterations. For a very large majority of instances and parameter settings, this is enough to hit the best result known from the whole experimental campaign. The computational time goes from fractions of a second for the instances with $n = 10$ vertices to about 22 s for the complete graphs with $n = 50$ vertices.

Table 1 considers a subset of 30 settings for the tabu tenure ranges, which describes the region in which the algorithm performs best. The rows correspond to 6 ranges of parameter ℓ_{out}, the columns to 5 ranges of parameter ℓ_{in}. In particular, all the ranges considered for ℓ_{out} and the last three ranges considered for ℓ_{in} adapt to the number of vertices n of the instance, whereas the other two ranges impose a fixed value: $\ell_{in} = 0$ and $\ell_{in} = 1$. Of course, $\ell_{in} = 0$ corresponds to allowing the insertion of any edge. Each cell of the table combines the row range for the removal tenure and the column range for the insertion tenure, and reports the average gap of the result thus achieved with respect to the best known one.

The best parameter setting ($\ell_{in} = 1$ and $\ell_{out} \in [0.35n; 0.45n]$) is bolded in Table 1. It corresponds to a 0.006% average gap, with 102 best known results out of 108. Since the average number of iterations required to find the result is much lower than the total one (8 650 versus 100 000), this setting usually computes a nearly optimal solution in a matter of few seconds. Many other settings, however, exhibit very similar gaps and numbers of best results found. Indeed, the difference between the best setting and the other ones shaded in grey in Table 1 are not statistically significant. Specifically, *Wilcoxon's matched-pairs signed-ranks test* [20] indicates that, with a probability $\geqslant 5\%$, such differences are due to random fluctuations. This suggests a predominance of the range $[0.35n; 0.45n]$ for ℓ_{out} and of the shortest ranges for ℓ_{in} (the tabu on insertion can be even completely relaxed). On the other side, an increase of ℓ_{in} can be compensated by a suitable decrease of ℓ_{out}.

These results were unexpected, since they are in contrast with our experience on related problems, such as the *Maximum Diversity Problem*, where the tenure for insertion must be larger than that for removal [1]. The theoretical intuition, as well, would point in the opposite direction. In fact, since the number of edges out of any solution strongly exceeds the number of

Table 1
Gaps with respect to the best known results achieved by *QMST-TS* with different tabu tenure ranges.

$[\ell_{out}^m; \ell_{out}^M]$	$[\ell_{in}^m; \ell_{in}^M]$				
	$[0; 0]$	$[1; 1]$	$[1; \lceil 0.05n \rceil]$	$[1; \lceil 0.10n \rceil]$	$[1; \lceil 0.15n \rceil]$
$[\lceil 0.15n \rceil; \lceil 0.25n \rceil]$	0.540%	0.336%	0.296%	0.183%	0.209%
$[\lceil 0.20n \rceil; \lceil 0.30n \rceil]$	0.330%	0.115%	0.126%	0.098%	0.091%
$[\lceil 0.25n \rceil; \lceil 0.35n \rceil]$	0.077%	0.058%	0.050%	0.025%	0.006%
$[\lceil 0.30n \rceil; \lceil 0.40n \rceil]$	0.018%	0.037%	0.035%	0.015%	0.020%
$[\lceil 0.35n \rceil; \lceil 0.45n \rceil]$	0.006%	**0.006%**	0.014%	0.038%	0.035%
$[\lceil 0.40n \rceil; \lceil 0.40n \rceil]$	0.031%	0.016%	0.018%	0.043%	0.047%

Table 2
Gaps with respect to the best known results achieved by *QMST-TS* with different restart frequencies.

r	I/r	Gap (%)	NF
1	100 000	0.006	6
10	10 000	0.005	6
100	1 000	0.026	10
1000	100	0.151	31

edges inside it ($m - n + 1 \gg n - 1$), the probability to be involved in an exchange should be lower for the inserted edge than for the removed one. If the strength of the two tabus should be comparable, the prohibition on insertion should be longer than that on removal.

Our second experiment evaluates the contribution of random restart to the effectiveness of *QMST-TS*. We have applied the best tenure ranges found in the first experiment ($\ell_{in} = 1$ and $\ell_{out} \in [0.35n; 0.45n]$), and kept the total number of iterations to $I = 100\,000$. However, we have periodically restarted the search from a random solution every I/r iterations, thus obtaining r independent runs. Table 2 reports in the first two columns the number of runs and the number of iterations for each run. The third column contains the average gap of the result with respect to the overall best known result, the fourth column the number of best known results not found by each parameter setting. With $r = 10$ runs of 10 000 iterations there is a slight, though not statistically significant, improvement in the average gap. This is in good accordance with the already mentioned fact that the number of iterations required to find the best result is on average 8 640. Any further increase in the number of restarts negatively affects the search, increasing both the gap and the number of best known results not found.

7.3. Influence of the starting solution

This phase of experiments analyzes the influence on the final result of the starting random solution, which is obtained exploiting the well-known `ranl` pseudorandom number generator [17]. First we have run 10 times algorithm *QMST-TS* with $r = 1$ and $I = 100\,000$, gradually changing the seed of the generator from -1 (the default value) to -10. In 90 instances out of 106, all runs obtained the best known result. In the other instances the gap was always $\leqslant 0.6\%$, apart from a single run on a single instance. The best known result could be found at least once for all instances but one.

To further investigate whether the Tabu Search mechanism is sufficiently robust to provide solutions independent from the random initialization, we focused on the two instances which suffered from the largest gap in our original experiments. They are denoted as `n050c010q100` and `n050c100q100` because both have $n = 50$ vertices and quadratic costs extracted from [1; 100], while the linear costs are extracted, respectively, from [1; 10] and [1; 100]. We have performed 100 runs of *QMST-TS* with different values of the random seed, from -1 to -100. Keeping track of the number of iterations required in each run to find the best known result, we have obtained an *iteration-to-target plot*, which provides the increasing fraction of runs that reach the best known solution as a function of the number of iterations. Fig. 4 reports these plots, together with a theoretical exponential interpolation, for the two instances considered. Though they are the most challenging in the benchmark, the algorithm reaches the best known solution in all 100 runs in a few millions iterations, i.e. in a few minutes. Therefore, in practice a limited recourse to randomization allows to avoid the rare cases in which a bad initialization has a long-lasting influence on the search.

7.4. Parameter tuning for the VNS algorithm

Algorithm *QMST-VNS* has only three parameters: the total number of iterations I, which we will fix to 100 000 as for algorithm *QMST-TS*, and the extreme levels $k_{\min}$ and $k_{\max}$ of the neighbourhood hierarchy. We remind that $1 \leqslant k_{\min} < k_{\max} \leqslant \tilde{k} = \min(n - 1, m - n + 1)$. We therefore consider the following five values, related to the size of the instance $n : 1,\ \lceil 0.2\tilde{k} \rceil,\ \lceil 0.4\tilde{k} \rceil,\ \lceil 0.6\tilde{k} \rceil,\ \lceil 0.8\tilde{k} \rceil,\ \tilde{k}$. Then we build $6 \cdot 5/2 = 15$ alternative parameter settings by assigning these values to $k_{\min}$ and $k_{\max}$ under the constraint that $k_{\min} < k_{\max}$. Table 3 reports the results: each row is associated to a value of $k_{\min}$ and each column to a value of $k_{\max}$. Each cell of the table corresponds to a parameter setting and reports the average gap of the result achieved by that setting with respect to the best known one. The algorithm is initialized with the same random solution used for the Tabu Search. The best parameter setting ($k_{\min} = \lceil 0.6\tilde{k} \rceil$ and $k_{\max} = \tilde{k}$) is bolded, and corresponds to a 0.089% average gap, with 89 best known results out of 108. Some other settings, shaded in grey, have a worse average gap, but Wilcoxon's test estimates that there is a probability $\geqslant 5\%$ that the differences are not significant. Anyway, there is a predominance of the settings in which $k_{\max}$ assumes the largest possible value and $k_{\min}$ a fairly large one. This is in contrast with the fact that most *VNS* algorithms proposed in the literature set $k_{\min} = 1$. The advantage of using a rather large value of $k_{\min}$ is probably related to the need of avoiding that local search could be driven back to its starting point. In fact, the average number of iterations from the starting solution to the local optimum increases from about 4 for $n = 10$ to about 40 for $n = 50$. These values are very close to the best setting for $k_{\min} = \lceil 0.6\tilde{k} \rceil$, which correspondingly increases from 3 to 29. Probably, smaller values of k often lead the search to fall back into an already visited local optimum.

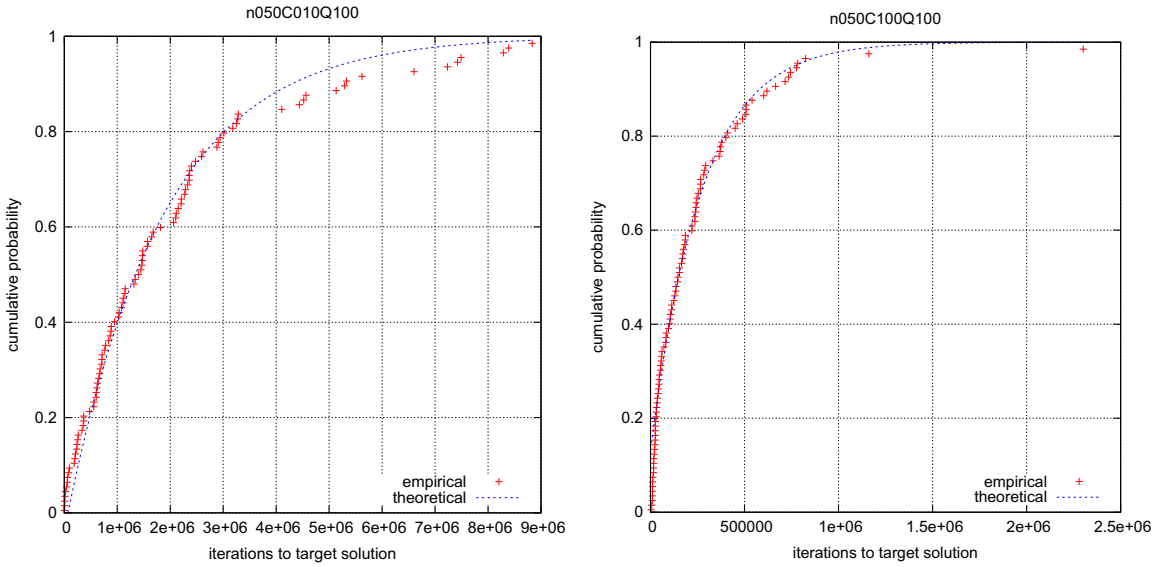

Fig. 4. Rate of convergence of *QMST-TS* to the best known result for different random starting solutions on instances `n050c010q100` (on the left) and `n050c100q100` (on the right).

Table 3
Gaps with respect to the best known results achieved by QMST-VNS with different parameter settings.

k_{min}	k_{max}				
	$\lceil 0.2\tilde{k} \rceil$	$\lceil 0.4\tilde{k} \rceil$	$\lceil 0.6\tilde{k} \rceil$	$\lceil 0.8\tilde{k} \rceil$	$\tilde{k}$
1	1.043%	0.464%	0.250%	0.155%	0.140%
$\lceil 0.2\tilde{k} \rceil$	–	0.599%	0.284%	0.202%	0.142%
$\lceil 0.4\tilde{k} \rceil$	–	–	0.203%	0.152%	0.158%
$\lceil 0.6\tilde{k} \rceil$	–	–	–	0.130%	0.089%
$\lceil 0.8\tilde{k} \rceil$	–	–	–	–	0.114%

The computational time goes from fractions of a second for the instances with $n = 10$ vertices to about 23 s for the complete graphs with $n = 50$ vertices. It is slightly larger than the corresponding time required by *QMST-TS*, due to the shaking procedure, which is called very frequently. Indeed, the difference in time is more marked on the smaller instances, where the restarts are more frequent.

7.5. Comparison between TS and VNS

Algorithms *QMST-TS* and *QMST-VNS* are based on the same neighbourhood, that is the introduction of a new edge and the removal of an old one. *QMST-TS* proceeds beyond local optima through a memory mechanism, *QMST-VNS* through a controlled restart mechanism. As we have seen, given the same number of local search iterations, *QMST-TS* achieves better results in a slightly lower time than *QMST-VNS*. Moreover, the average number of iterations required to find the best known results is around 8 640 for *QMST-TS*, versus the 11 100 iterations required by *QMST-VNS*. This suggests that in a direct comparison *QMST-TS* should outperform *QMST-VNS*.

We have performed such a comparison on an "equal-time" basis, giving both algorithms a total computational time of $T = 1$ second, initializing them with the same random solution. We have also adopted the best parameter setting for each algorithm, with the exception that algorithm *QMST-TS* is never restarted, to stress the difference between its memory mechanism and the restart mechanism of *VNS*. Table 4 compares the results: the first column reports the number of vertices n, the following two columns report the average and the maximum gap with respect to the best known result obtained by *QMST-TS* in the limit time on the twelve instances with n vertices (the minimum gap is always zero). Label "-" stands for a zero gap. The last two columns provide the same information for *QMST-VNS* (also in this case the minimum gap is always zero). Up to $n = 20$ both algorithm have zero maximum gap; on the larger instances, their performance differs, confirming the superiority of *QMST-TS*. Notice that for the larger instances the limit time is much shorter than the computational time required by the previous experiments, which confirms that both algorithms, and in particular *QMST-TS*, are effective also in a short run. The

Table 4
Comparison between algorithms *QMST-TS* and *QMST-VNS* in a limit time of $T = 1$ s.

n	*QMST-TS*		*QMST-VNS*	
	Avg. (%)	Max. (%)	Avg. (%)	Max. (%)
10	–	–	–	–
15	–	–	–	–
20	–	–	–	–
25	–	–	0.05	0.57
30	0.06	0.44	0.17	0.51
35	0.01	0.13	0.11	1.22
40	0.05	0.31	0.56	2.12
45	0.25	1.11	0.78	2.76
50	0.20	0.67	1.01	3.46

superior performance of Tabu Search with respect to Variable Neighbourhood Search was unexpected, given that our previous experience on similar problems, namely the *Maximum Diversity Problem*, pointed exactly in the opposite direction [2].

7.6. Results for the exact algorithm

This section briefly describes the application of the branch-and-bound algorithm *QMST-BB*, described in Section 6, to benchmark CP. Section 7.7 will compare the results of *QMST-BB* to those of a Lagrangean lower bounding procedure on benchmark OP1. The instances of the other benchmarks are too large to be attacked by exact algorithms.

We initialize *QMST-BB* with the solution obtained by *QMST-TS* with the standard parameter setting ($r = 10$ runs of $I/r = 10\,000$ iterations each, with $\ell_{in} = 1$ and $\ell_{out} \in [0.35n; 0.45n]$). Table 5 reports the results: each row contains values averaged over the four instances with the same size and density. We have limited our experience to the instances with $n \leqslant 30$ since the bound on the larger ones is of little significance. The first two columns report the number of vertices n and edges m of the instances. The following triplet of columns reports the percent gap between the upper and the lower bound on the optimum after one hour of computation ("-" if all four instances have been solved exactly within the time limit), the number of branching nodes processed, the computational time in seconds (label "> 3600" indicates that none of the four instances was solved in the given time).

The exact algorithm confirmed the optimality of the best known solutions for all instances with $n = 10$ and $n = 15$ vertices and for the sparse instances with $n = 20$ and $m = 63$. The main limiting feature of the algorithm is memory consumption, which forbids to apply it with meaningful results to instances much larger than $n = 20$ vertices, as it appears from the number of branching nodes processed.

7.7. Comparison with the literature

This section reviews the benchmarks available in the literature on the *QMSTP* and compares the results published on each of them with the results obtained by algorithm *QMST-TS*. We always apply the standard parameter setting tuned on benchmark CP, without adapting it to the specific instances: the tenure ranges are $\ell_{in} = 1$ and $\ell_{out} \in [0.35n; 0.45n]$ while the number of iterations for each run is $I/r = 10\,000$. The number of runs is modified case by case in order to yield a computational time

Table 5
Results of the exact branch-and-bound algorithm *QMST-BB*.

n	m	*QMST-BB*		
		Gap (%)	BN	CPU
10	15	–	16	0.004
	30	–	900	0.008
	45	–	6326	0.059
15	35	–	19204	0.176
	70	–	888244	43.550
	105	–	21538070	471.923
20	63	–	37556550	657.168
	127	28.00	112595181	>3600
	190	49.50	80921765	>3600
25	100	26.88	120245564	>3600
	200	75.18	67526424	>3600
	300	106.58	46388125	>3600
30	145	58.31	76078983	>3600
	290	122.81	43722697	>3600
	435	161.47	30874912	>3600

smaller than that required by the competing algorithms. Our detailed instance-by-instance results on all benchmarks can be retrieved from http://homes.dsi.unimi.it/cordone/research/ResultsQMST.pdf.

Benchmark SCA [18]. This is the only benchmark widely used in the literature: after its introduction in [18], in fact, it has been used in [15,19]. There is a small uncertainty in the data, given by the fact that the original linear costs are Euclidean distances turned into integer numbers. We could not know from the authors of [18,15] the kind of rounding adopted. The authors of [19] rounded them to the closest integer, and we did the same.

The first column of Table 6 provides the size n of the instance. The following ones report the result and the computational time of each competing algorithm. All of them have been run for 20 times, and the table contains the best result found and the total computational time in seconds. Algorithm *EWD* is the best genetic algorithm proposed in [18], where it is denoted as *EWD+ANX with dK-TCR* and runs on a Pentium IV 1.6 GHz CPU. Algorithm *RLS-TT* is the Randomized Local Search with Tabu Thresholding proposed in [15] and runs on a Pentium IV PC with a 3 GHz CPU and 2 GB of RAM. In the original papers, the best results for these two algorithms are expressed as a percent gap with respect to a previously best known value, also reported in a side column. Following [19], we converted this information into absolute values for comparing purposes. Algorithm *ABC* is the best performing version of the Artificial Bee Colony approach proposed in [19] and it runs on a Linux based 3.0 GHz core 2 duo system with 2 GB of RAM. Finally, algorithm *QMST-TS* is our Tabu Search procedure, which runs on a 2.6 GHz Intel Pentium Core 2 Duo E6700 with 2 GB of RAM. We have applied the standard parameter setting with no further tuning, and the number of runs is $r = 20$ as for all other algorithms. The best result on each instance is bolded in Table 6. The performance of algorithms *ABC* and *QMST-TS* are nearly equivalent: for two instances they find the same result; for the other four instances they find two best results each. The computational time of *QMST-TS* is on average 60% of that of *ABC*, but the machines employed are different, though very similar, so that the comparison is not conclusive. The other two algorithms reach worse results in a much larger time on all instances.

Benchmark CP [5]. This is the benchmark used to tune the parameters of our algorithms, and a subset of instances with $n = 40, 45$ and 50 has been adopted as a benchmark in [16], selecting those which required the longest computational times to be solved by the algorithms there presented. We can therefore compare algorithm *QMST-TS* with the best of those algorithms, that is an Iterated Tabu Search (*ITS*) running on a 3.0 GHz PC with an Intel Core 2 Duo CPU. This algorithm has been run 10 times until the best known solution has been found on all selected instances. We have done the same, applying the standard parameter setting, restarting the search every 10 000 iterations and terminating each run as soon as the best known result is obtained.

Table 7 reports the results: its first four columns identify the instance, through the number of vertices n, the number of edges m and the ranges in which the linear costs c and the quadratic costs q are extracted. The following two columns provide the average and the maximum time in seconds required to reach the best known solution in the 10 runs of algorithm *ITS*. The last two columns give the same information for algorithm *QMST-TS*. The last row provides average values over the 23 selected instances. Both algorithms find the best known solution in all 10 runs. The average and maximum time required by *ITS* is on average 45–50% of that required by *QMST*, but also in this case the comparison is not conclusive, given that the two machines employed are different, though very similar.

Benchmark OP1 [15]. This very large benchmark has been used to evaluate the performance of algorithm *RLS-TT* and of a refined Lagrangean lower bounding procedure. Both procedures run on a Pentium IV PC with a 3 GHz CPU and 2 GB of RAM. Average results have been published for each of the 48 groups of 10 instances which belong to the same subclass (SYM, VSYM or ESYM) and have the same number of vertices n; the detailed results are not available. The first two columns of Tables 8–10 identify the groups of instances with the same structure and size (number of vertices n and edges m). The following four columns provide the average lower bound and upper bound (merged into a single value if they coincide), and the computational times in seconds required by the Lagrangean lower bounding procedure to compute the former and by algorithm *RLS-TT* to compute the latter. This information derives from [15], and specifically from columns 6 to 7 of Table 2 for the lower bounds, from columns 6 to 7 of Table 1 for the upper bounds. Unfortunately, some of these results, shaded in dark grey, exhibit internal inconsistencies, probably due to typos. Specifically, the upper bounds reported in Table 8 for n ranging from 13 to 30 have two decimal digits, though they should be the average of 10 integer values, while the upper bounds reported in Table 9 for $n = 18, 20$ and 30 and in Table 10 for $n = 30$ and 50 are smaller than the corresponding lower bounds.

The last four columns provide the same information for our branch-and-bound algorithm *QMST-BB*, initialized by the Tabu Search algorithm *QMST-TS*. When all 10 instances of a group are solved to optimality, the lower and the upper bound

Table 6
Comparison between the state-of-the-art algorithms on benchmark SCA.

n	*EWD* [18]		*RLS-TT* [15]		*ABC* [19]		*QMST-TS*	
	UB	CPU	UB	CPU	UB	CPU	UB	CPU
50	25 339	343.0	25 226	3242.1	**25 200**	87.0	**25 200**	44.5
60	36 086	495.7	35 754	4321.4	35 466	169.0	**35 447**	83.2
70	48 538	716.6	48 536	5738.5	**48 125**	337.2	**48 125**	178.5
80	63 546	1086.7	63 546	7026.3	63 022	417.8	**63 004**	340.4
90	79 627	1337.2	79 922	8623.6	**78 879**	751.8	78 912	579.7
100	98 342	1828.9	98 811	10431.3	**96 750**	1542.4	96 757	789.4

Table 7
Comparison between the state-of-the-art algorithms on benchmark CP.

Instance				*ITS*		*QMST-TS*	
n	m	c	q	t^*_{avg}	t^*_{max}	t^*_{avg}	t^*_{max}
40	522	[1;10]	[1;100]	3.6	10.2	2.5	6.9
40	522	[1;100]	[1;100]	7.8	24.9	8.0	25.0
40	780	[1;10]	[1;100]	5.7	21.3	2.8	8.2
45	663	[1;10]	[1;10]	11.2	24.1	24.2	89.3
45	663	[1;10]	[1;100]	0.6	1.9	1.6	4.8
45	663	[1;100]	[1;10]	0.2	0.6	1.0	5.4
45	663	[1;100]	[1;100]	0.7	1.8	2.9	10.9
45	990	[1;10]	[1;10]	21.0	50.5	33.1	128.7
45	990	[1;10]	[1;100]	8.9	30.3	32.6	96.9
45	990	[1;100]	[1;10]	0.4	0.9	1.2	3.1
45	990	[1;100]	[1;100]	20.9	48.3	43.8	134.3
50	404	[1;10]	[1;10]	0.2	0.7	0.9	2.4
50	404	[1;10]	[1;100]	0.2	0.5	0.5	1.6
50	404	[1;100]	[1;10]	0.0	0.2	0.4	1.0
50	404	[1;100]	[1;100]	0.2	0.9	0.8	2.7
50	820	[1;10]	[1;10]	2.2	8.4	7.2	22.9
50	820	[1;10]	[1;100]	3.7	7.9	14.7	30.2
50	820	[1;100]	[1;10]	0.2	0.3	2.3	10.8
50	820	[1;100]	[1;100]	2.0	7.2	22.4	62.2
50	1225	[1;10]	[1;10]	13.3	46.5	23.6	65.6
50	1225	[1;10]	[1;100]	50.1	151.7	107.3	193.3
50	1225	[1;100]	[1;10]	7.6	30.0	35.4	154.5
50	1225	[1;100]	[1;100]	8.3	41.8	72.5	178.4
Avg.				7.3	22.2	19.2	53.9

are merged into a single value. In fact, *QMST-BB* solved exactly all instances up to $n = 15$ within a limit of 100 000 000 branching nodes. For the larger instances, we reduced the limit to 1 000 000 branching nodes, which still allowed to solve all instances up to $n = 20$ for subclass VSYM and all instances up to $n = 16$ for subclass ESYM. Label "0.0" stands for a computational time lower than 0.05 s. Of course, we were unable to reproduce the inconsistent results and, unfortunately, also part of the consistent ones. The cells shaded in light grey in the three tables correspond to bounds which are contradicted by our results, i. e. either lower bounds larger than our upper bound or upper bounds smaller than our lower bound. In Table 10 we have no result for $n = 20$ because the available files are actually copies of the VSYM instances with $n = 6$. Finally, the last row of each table reports average values over all sizes.

It is clearly not easy to draw sound conclusions in this situation. Focusing on the results which are both self-consistent and compatible with ours, we can observe that the Lagrangean bound is tighter than the combinatorial bound at the root

Table 8
Comparison between the state-of-the-art algorithms on benchmark OP1 SYM.

n	m	*RLS-TT*				*QMST-TS/QMST-BB*			
		LB	UB	t_{LB}	t_{UB}	LB	UB	t_{LB}	t_{UB}
6	15	258.4		0.1	0.2	258.4		0.0	0.1
7	21	312.9	326.8	0.2	0.4	326.8		0.0	0.2
8	28	409.8	438.5	0.4	1.2	438.5		0.0	0.2
9	36	505.6	534.9	0.8	2.1	534.9		0.0	0.3
10	45	547.9	650.1	1.3	3.3	653.9		0.0	0.4
11	55	613.2	785.9	2.0	4.7	785.9		0.1	0.5
12	66	652.1	915.1	2.6	7.2	918.5		0.5	0.6
13	78	713.0	1053.33	3.9	12.2	1067.1		2.7	0.7
14	91	812.1	1235.17	5.9	16.6	1249.8		14.9	0.8
15	105	860.1	1358.75	7.3	29.5	1390.2		79.9	1.0
16	120	937.1	1594.94	14.2	60.2	1387.9	1629.3	27.3	1.2
17	136	1020.3	1801.78	21.0	74.3	1452.8	1823.8	32.5	1.4
18	153	1085.5	2036.68	24.0	90.4	1520.2	2081.0	37.4	1.5
20	190	1220.0	2525.01	28.2	121.4	1611.2	2572.7	50.2	2.0
30	435	1828.0	5929.36	301.0	511.1	1969.8	6015.9	147.9	5.1
50	1225	2845.1	18154.5	3311.7	2615.2	2158.4	17617.0	1333.5	25.5
Avg.		913.8	2475.0	232.8	221.9	1107.8	2460.2	107.9	2.6

Table 9
Comparison between the state-of-the-art algorithms on benchmark OP1 VSYM.

n	m	*RLS-TT*				*QMST-TS/QMST-BB*			
		LB	UB	t_{LB}	t_{UB}	LB	UB	t_{LB}	t_{UB}
6	15	16290.4		0.1	0.3	16273.9		0.0	0.1
7	21	19625.7		0.2	0.4	19625.7		0.0	0.1
8	28	26739.4	27039.4	0.4	1.4	27039.4		0.0	0.2
9	36	22768.1	22769.9	0.7	2.6	22769.9		0.0	0.3
10	45	25750.5		1.2	3.9	25743.8		0.0	0.3
11	55	29295.0	29307.6	1.8	4.8	29307.6		0.0	0.4
12	66	32607.2	32615.8	2.6	8.4	32577.8		0.0	0.5
13	78	40458.1	40488.5	3.8	13.5	40488.5		0.0	0.6
14	91	44310.1	44338.2	5.2	26.3	44240.4		0.0	0.7
15	105	50789.0	50821.6	9.1	32.5	50821.6		0.0	0.8
16	120	41907.2	41940.2	12.1	44.1	41940.2		0.1	1.0
17	136	41779.6	41819.6	15.8	57.3	41819.0		0.2	1.1
18	153	46372.3	45145.2	18.4	70.4	46130.2		0.6	1.3
20	190	55862.2	54610.3	25.3	124.1	55326.2		8.7	1.7
30	435	80177.6	78999.9	178.5	523.3	70534.5	78999.9	141.0	4.2
50	1225	126281.0	164040	1983.6	2380.4	117745.4	165419.9	808.8	14.4
Avg.		43813.3	45975.1	141.2	205.9	42649.0	46157.8	60.0	1.7

Table 10
Comparison between the state-of-the-art algorithms on benchmark OP1 ESYM.

n	m	*RLS-TT*				*QMST-TS/QMST-BB*			
		LB	UB	t_{LB}	t_{UB}	LB	UB	t_{LB}	t_{UB}
6	15	541.1	541.2	0.1	0.2	541.2		0.0	0.1
7	21	782.7	783.7	0.3	0.6	783.7		0.0	0.2
8	28	1020.1		0.5	1.6	1020.1		0.0	0.2
9	36	1345.7	1356.0	0.8	2.1	1356.0		0.0	0.3
10	45	1422.0	1427.1	1.3	4.8	1427.1		0.0	0.4
11	55	1535.6	1545.1	1.8	7.9	1545.1		0.0	0.4
12	66	1893.9	1901.6	2.9	9.6	1901.6		0.1	0.5
13	78	2169.2	2175.3	3.7	14.1	2175.3		0.3	0.6
14	91	2515.3	2527.9	8.2	21.2	2527.9		0.6	0.8
15	105	2578.2	2588.8	7.3	25.7	2588.8		0.6	1.0
16	120	2969.7	2980.1	9.5	33.9	2980.1		2.9	1.2
17	136	3372.1	3372.2	12.6	47.2	3372.1	3372.2	12.9	1.3
18	153	3510.0	3646.9	17.4	69.3	3567.2	3569.0	14.1	1.5
20	190	4329.8	4193.8	22.6	102.3				
30	435	8311.1	7992.7	163.7	515.1	7208.6	8056.7	145.2	5.3
50	1225	16127.0	15654.5	1795.2	2424.1	13114.9	15790.6	1131.0	25.2
Avg.		3401.5	3356.7	128.0	205.0	3074.0	3309.0	87.2	2.6

node, but the additional time it requires often allows the branching mechanism to compensate for the initial disadvantage. In fact, *QMST-BB* usually finds better bounds in a smaller computational time, though the comparison is not conclusive, since the two machines employed have slightly different clock frequencies. On the other side, the Lagrangean bound is preferable on the largest instances, both because it is tighter and because memory consumption becomes a relevant issue for algorithm *QMST-BB*. Concerning the heuristic algorithms, they often find the same solutions, but the computational time of *QMST-TS* is two orders of magnitude smaller. Finally, our results confirm the remark made in [15] that the SYM instances are much more difficult than the ESYM and VSYM instances.

Benchmark OP2 [15]. The instances of this benchmark are quite different from the complete random graphs used in benchmarks SCA, OP1 and SS. In fact, they derive from instances of the *QAP*, suitably transformed in order to guarantee a one-to-one correspondence of the feasible solutions between the two problems. This gives them a rather peculiar structure (for details, see [15]). The first column of Table 11 reports the name of each instance, the second one its optimal value, which is known from the *QAP* literature. The following three columns report the heuristic value, the percent gap with respect to the optimum and the computational time in seconds for algorithm *RLS-TT*. Other three columns report the same information for algorithm *QMST-TS* with the standard parameter setting. Since the computational times of *RLS-TT* are quite large, the number of runs has been increased to $r = 1\,000$. Label "-" corresponds to a zero gap. Algorithm *QMST-TS* finds the optimum for 3 instances out of 29, and its average gap is half that of *RLS-TT* on the NUG subclass and less than half on the CHR subclass. The

Table 11
Comparison between the state-of-the-art algorithms on benchmark OP2.

Instance	Opt.	*RLS-TT*			*QMST-TS*			*QMST-TS* retuned		
		UB	Gap (%)	CPU	UB	Gap (%)	CPU	UB	Gap (%)	CPU
nug12	578	605	4.67	639	582	0.69	293.8	578	–	287.4
nug14	1014	1084	6.90	724	1022	0.79	420.3	1014	–	414.3
nug15	1150	1265	10.00	1348	1188	3.30	498.8	1150	–	492.8
nug16a	1610	1742	8.20	2311	1670	3.73	584.7	1622	0.75	582.6
nug16b	1240	1350	8.87	2936	1288	3.87	589.8	1240	–	578.9
nug17	1732	1874	8.20	3422	1806	4.27	694.9	1750	1.04	685.3
nug18	1930	2056	6.53	3482	2016	4.46	798.8	1942	0.62	793.2
nug20	2570	2860	11.28	5151	2692	4.75	1072.5	2580	0.39	1047.2
nug21	2438	2698	10.66	5184	2570	5.41	1219.1	2488	2.05	1200.8
nug22	3596	3868	7.56	5482	3820	6.23	1444.9	3672	2.11	1370.7
nug24	3488	3874	11.07	5914	3758	7.74	1978.2	3590	2.92	1838.1
nug25	3744	4083	9.05	5983	3958	5.72	2304.9	3874	3.47	2097.9
nug27	5234	5966	13.99	6025	5574	6.50	3062.8	5352	2.25	2787.8
nug28	5166	5819	12.64	6087	5544	7.32	3504.5	5262	1.86	3227.6
nug30	6124	6923	13.05	6227	6702	9.44	4597.0	6364	3.92	4282.7
Avg.			9.51	4061.0	–	4.95	1537.7	–	1.43	1445.8
chr12a	9552	11170	16.94	783	9552	–	289.4	9552	–	287.7
chr12b	9742	10753	10.38	790	9742	–	293.5	9742	–	286.9
chr12c	11156	12712	13.95	783	11156	–	293.2	11156	–	285.7
chr15a	9896	11638	17.60	1239	10304	4.12	498.5	9936	0.40	497.3
chr15b	7990	10145	26.97	1136	9470	18.52	501.2	7990	–	491.9
chr15c	9504	12769	34.35	1254	11224	18.10	499.5	9504	–	492.0
chr18a	11098	12757	14.95	3325	13104	18.08	805.3	11098	–	792.9
chr18b	1534	1676	9.26	3354	1646	7.30	807.4	1534	–	789.2
chr20a	2192	2445	11.54	4968	2780	26.82	1052.2	2192	–	1043.1
chr20b	2298	2730	18.80	4652	2694	17.23	1055.1	2352	2.35	1043.5
chr20c	14142	30124	113.01	4763	18330	29.61	1047.3	14202	0.42	1045.5
chr22a	6156	8760	42.30	5089	6738	9.45	1394.7	6228	1.17	1394.8
chr22b	6194	8402	35.65	4741	6748	8.94	1408.7	6314	1.94	1421.5
chr25a	3796	9658	154.43	5223	5856	54.27	2306.5	3866	1.84	2135.0
Avg.			37.15	3007.1		15.18	875.2		0.58	857.6

computational time is also less than half, though the machine employed has a lower clock frequency. Moreover, while in the other benchmarks the standard tuning of *QMST-TS* is already good, on this benchmark we have obtained strong improvements by retuning the tabu tenure ranges as $\ell_{in} \in [0.35n; 0.45n]$ and $\ell_{out} \in [1; 0.05n]$, that is quite similar to exchanging their values with respect to the standard setting. The last three columns report the results of the retuned version of *QMST-TS* with

Table 12
Comparison between the state-of-the-art algorithms on benchmark SS.

n	Instance	*ABC*		*QMST-TS*	
		UB	CPU	UB	CPU
25	1	**5085**	18.20	**5085**	6.683
25	2	**5081**	20.40	**5081**	6.603
25	3	**4962**	21.00	**4962**	6.905
50	1	**21126**	173.60	**21126**	50.362
50	2	21123	176.80	**21106**	50.402
50	3	**21059**	190.20	**21059**	50.583
100	1	89098	2333.20	**88871**	965.769
100	2	89202	2319.00	**89049**	957.699
100	3	89007	1977.60	**88720**	961.186
150	1	205619	8897.40	**205615**	2928.724
150	2	205874	7486.60	**205509**	2923.025
150	3	205634	8658.60	**205094**	2928.603
200	1	371797	22828.40	**371492**	6320.302
200	2	371864	23112.00	**371698**	6332.131
200	3	372156	25534.20	**371584**	6324.342
250	1	587924	51268.20	**586861**	9572.332
250	2	588068	56818.20	**587607**	9592.874
250	3	587883	46565.80	**587281**	9601.206

the same values of r and I : 12 instances are solved to optimality and the average gap decreases to 1.43% on the NUG subclass and 0.58% on the CHR subclass.

Benchmark SS [19]. This benchmark has been used to test algorithm *ABC* on a 3.0 GHz core 2 duo system with 2 GB of RAM. Algorithm *ABC* has been run 20 times and the same has been done for *QMST-TS*, with $r = 1$ and $I = 10000$ for each run. The first two columns of Table 12 identify each instance. The following two provide the best result found by algorithm *ABC* and the total computational time in seconds, i. e. 20 times the average execution time. The last two columns report the same information for algorithm *QMST-TS*. The best result on each instance is bolded in Table 12. The two algorithms find the same best result for 5 instances out of 18; *QMST-TS* finds a better solution for the other 13 instances. Though the machine employed has a slightly smaller clock frequency, the computational time is one third, but this difference is probably not large enough to be fully conclusive.

8. Conclusions

We have discussed two heuristic and an exact approach for the *QMSTP*, a neat combinatorial model recently come into a certain interest. The Tabu Search heuristic *QMST-TS* proves better and more robust than the Variable Neighbourhood Search heuristic *QMST-VNS*. The branch-and-bound algorithm *QMST-BB* can solve exactly instances up to 20 vertices in less than one hour. For larger instances, the gap between lower and upper bounds and the memory consumption grow steeply with the number of vertices, confirming the theoretical finding that the *QMSTP* is very hard from the computational point of view. A complete experimental comparison on all the available benchmarks shows that algorithm *QMST-TS* is at least competitive with all the state-of-the-art heuristics and nearly always confirms or improves the best known results published in the literature.

Acknowledgements

The authors thank the anonymous referee for providing constructive comments and the authors of [18,15,19] for kindly making their benchmark instances available.

References

[1] R. Aringhieri, R. Cordone, Y. Melzani, Tabu search vs. GRASP for the maximum diversity problem, 4OR: A Quarterly Journal of Operations Research 6 (1) (2008) 45–60.
[2] Roberto Aringhieri, R. Cordone, Comparing local search metaheuristics for the maximum diversity problem, Journal of the Operational Research Society 62 (2) (2011) 266–280.
[3] A. Assad, W. Xu, The quadratic minimum spanning tree problem, Naval Research Logistics 39 (1992) 399–417.
[4] A. Charnes, W.W. Cooper, Deterministic equivalents for optimizing and satisficing under chance constrainst, Operations Research 11 (1964) 18–39.
[5] R. Cordone, G. Passeri, Heuristic and exact algorithms for the quadratic minimum spanning tree problem, in: Proceedings of the 7th Cologne-Twente CTW08 Workshop on Graphs and Combinatorial Optimization, Gargnano, Italy, May 13–15, 2008, pp. 168–171.
[6] R. Cordone, M. Trubian, An exact algorithm for the node weighted Steiner tree problem, 4OR: A Quarterly Journal of Operations Research 4 (2) (2006) 124–144.
[7] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, The MIT Press, 2001.
[8] J. Gao, M. Lu, Fuzzy quadratic minimum spanning tree problem, Applied Mathematics and Computation 164 (3) (2005) 773–788.
[9] J. Gao, M. Lu, L. Li, Chance-constrained programming for fuzzy quadratic minimum spanning tree problem, in: Proceedings of the 2004 IEEE International Conference on Fuzzy Systems, Piscataway, NJ, July 2004, IEEE Press, 2004, pp. 983–987.
[10] M.R. Garey, D.S. Johnson, Computers and Intractability. A Guide to the Theory of NP-Completeness, W.H. Freeman, New York, 1979.
[11] M. Gendreau, J.-F. Larochelle, B. Sansó, A tabu search heuristic for the Steiner tree problem, Networks 34 (2) (1999) 162–172.
[12] F. Glover, Tabu thresholding: improved search by nonmonotonic trajectories, ORSA Journal on Computing 7 (1995) 426–442.
[13] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers., 1997.
[14] P. Hansen, N. Mladenovic, Variable neighborhood search, in: F. Glover, G. Kochenagen (Eds.), Handbook of Metaheuristics, Kluwer Academic Publishers, 2003, pp. 145–184.
[15] T. Öncan, A.P. Punnen, The quadratic minimum spanning tree problem: a lower bounding procedure and an efficient search, Computers & Operations Research 37 (10) (2010) 1762–1773.
[16] G. Palubeckis, D. Rubliauskas, A. Targamadzė, Metaheuristic approaches for the quadratic minimum spanning tree problem, Information Technology and Control 39 (4) (2010) 257–268.
[17] W.H. Press, Numerical Recipes in C, Cambridge University Press, Cambridge, 1990.
[18] S.-M. Soak, D.W. Corne, B.-H. Ahn, The edge-window-decoder representation for tree based problems, IEEE Transactions on Evolutionary Computation 10 (2) (2006) 124–144.
[19] S. Sundar, A. Singh, A swarm intelligence approach to the quadratic minimum spanning tree problem, Information Sciences 180 (17) (2010) 3182–3191.
[20] F. Wilcoxon, Individual comparisons by ranking methods, Biometrics 1 (1945) 80–83.
[21] W. Xu, The Quadratic Minimum Spanning Tree Problem and Other Topics. PhD thesis, University of Maryland, College Park, MD, 1984.
[22] W. Xu, On the quadratic minimum spanning tree problem, in: M.Gen, W.Xu, (Eds.), Japan–China International Workshop on Information Systems, Ashikaga, 1995, pp. 141–148.
[23] G. Zhou, M. Gen, An effective genetic algorithm approach to the quadratic minimum spanning tree problem, Computers & Operations Research 25 (3) (1998) 229–237.