Engineering Applications of Artificial Intelligence 46 (2015) 113–130

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

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

A three-phase search approach for the quadratic minimum spanning tree problem

Zhang-Hua Fu[a,b], Jin-Kao Hao[a,*]

[a] LERIA, Université d'Angers, 2 Bd Lavoisier, 49045 Angers Cedex 01, France
[b] Laboratory for Robotics and Intelligent Manufacturing, The Chinese University of Hong Kong (Shen Zhen), 518172 Shen Zhen, China

ARTICLE INFO

Article history:
Received 11 September 2014
Received in revised form
17 May 2015
Accepted 31 August 2015
Available online 28 September 2015

Keywords:
Spanning tree
Network design
Neighborhood search
Heuristics

ABSTRACT

Given an undirected graph with costs associated with each edge as well as each pair of edges, the quadratic minimum spanning tree problem (QMSTP) consists of determining a spanning tree of minimum cost. QMSTP is useful to model many real-life network design applications. We propose a three-phase search approach named TPS for solving QMSTP, which organizes the search process into three distinctive phases which are iterated: (1) a descent neighborhood search phase using two move operators to reach a local optimum from a given starting solution, (2) a local optima exploring phase to discover nearby local optima within a given regional area, and (3) a perturbation-based diversification phase to jump out of the current regional search area. TPS also introduces a pre-estimation criterion to significantly improve the efficiency of neighborhood evaluation, and develops a new swap-vertex neighborhood (as well as a swap-vertex based perturbation operator) which prove to be quite powerful for solving a series of special instances with particular structures. Computational experiments based on 7 sets of 659 popular benchmarks show that TPS produces highly competitive results compared to the best performing approaches in the literature. TPS discovers improved best known results (new upper bounds) for 33 open instances and matches the best known results for all the remaining instances. Critical elements and parameters of the TPS algorithm are analyzed to understand its behavior.

1. Introduction

Network design is an extremely challenging task in numerous resource distribution systems (e.g., transportation, electricity, telecommunication, and computer networks). Many of these systems can conveniently be modeled as some variants of the spanning or Steiner tree problem (STP). In this paper, we focus on the *quadratic minimum spanning tree problem* (*QMSTP*) which has broad practical applications. Let $G=(V,E)$ be a connected undirected graph with $|V|=n$ vertices and $|E|=m$ edges. Let $c: E \rightarrow \mathbb{R}$ be a linear cost function for the set of edges and $q: E \times E \rightarrow \mathbb{R}$ be a quadratic cost function to weigh each pair of edges (without loss of generality, assume $q_{ee}=0$ for all $e \in E$). QMSTP requires to determine a spanning tree $T=(V,X)$, so as to minimize its total cost $F(T)$, i.e., the sum of the linear costs plus the quadratic costs. Naturally, as in Cordone and Passeri (2012), this problem can be formulated as follows:

$$\text{Minimize } F(T) = \sum_{e \in E} c_e x_e + \sum_{e \in E} \sum_{f \in E} q_{ef} x_e x_f, \quad (1)$$

$$\text{subject to} \sum_{e \in E} x_e = n-1, \quad (2)$$

$$\sum_{e \in E(S)} x_e \le |S|-1, \quad \forall\ S \subset V \text{ with } |S| \ge 3, \quad (3)$$

$$x_e \in \{0,1\}, \quad \forall\ e \in E, \quad (4)$$

where $x_e = 1$ if edge e belongs to the solution, $x_e = 0$ otherwise. S is any possible subset of V (with $|S| \ge 3$) and $E(S)$ denotes the set of edges with both end vertices in S. Eq. (2) requires that the final solution contains $n-1$ edges and Eq. (3) ensures that no cycle exists in the solution. These two constraints together guarantee that the obtained solution is necessarily a spanning tree.

As an extension of the classical *minimum spanning tree problem* (*MST*) in graphs, QMSTP has various practical applications in network design problems, where the linear function models the cost to build or use edges, while the quadratic function models interference costs between pairs of edges. For example, in transportation, telecommunication or oil supply networks, the linear

* Corresponding author.
E-mail addresses: fu@info.univ-angers.fr (Z.-H. Fu), hao@info.univ-angers.fr (J.-K. Hao).

http://dx.doi.org/10.1016/j.engappai.2015.08.012
0952-1976/© 2015 Elsevier Ltd. All rights reserved.

function represents the costs for building each road, communication link or pipe, and the quadratic function represents the extra costs needed for transferring from one road (link, pipe) to another one. Normally, the interference costs are limited to pairs of adjacent edges (which share a common vertex) (Maia et al., 2013, 2014; Pereira et al., 2013b, 2015), but in some special cases, the interference costs also exist between any pair of edges. As discussed in Assad and Xu (1992), Öncan and Punnen (2010), and Palubeckis et al. (2010), QMSTP has several equivalent formulations such as the stochastic minimum spanning tree problem, the quadratic assignment problem, and the unconstrained binary quadratic optimization problem.

During the last two decades, QMSTP has been extensively investigated and many exact and heuristic approaches have been proposed. Since QMSTP is $\mathcal{NP}$-hard and difficult to approximate (Xu, 1995), exact methods are often applied only to solve very small instances. For larger instances, heuristics are preferred to obtain feasible solutions within a reasonable time.

As for exact methods, Assad and Xu (1992) and Xu (1995) propose a Lagrangian branch-and-bound (B&B) method. Öncan and Punnen (2010) combine the Lagrangian relaxation scheme with an extended formulation of valid inequalities to obtain tighter bounds. Cordone and Passeri (2012) improve the Lagrangian B&B procedure in Assad and Xu (1992). Pereira et al. (2013a) introduce a 0-1 programming formulation based on the reformulation–linearization technique and derive an effective Lagrangian relaxation. Using the resulting strong lower bounds and other formulations, they develop two effective parallel B&B algorithms able to solve optimally problem instances with up to 50 vertices. Recently, based on reduced cost computation, Rostami and Malucelli (2014) combine a reformulation scheme with new mixed 0-1 linear formulations and report lower bounds on hundreds of instances with up to 50 vertices. Exact algorithms are also proposed for solving other closely related QMSTP variants. Buchheim and Klein (2013, 2014) propose a B&B approach for QMSTP with one quadratic term in the objective function, of which the polyhedral descriptions are completed in Fischer and Fischer (2013). Pereira et al. (2013b) introduce several exact approaches (brand-and-cut, branch-and-cut-and-price), to obtain strong lower bounds for QMSTP with adjacency costs, for which the interference costs are limited to adjacent edges.

On the other hand, to handle large QMSTP instances, heuristics become the main approaches to obtain good near-optimal solutions within a reasonable time. For example, two greedy algorithms are proposed in Assad and Xu (1992), Xu (1984), and Xu (1995). Several genetic algorithms are implemented by Zhou and Gen (1998) and tested on instances with up to 50 vertices. Another evolutionary algorithm is proposed for a fuzzy variant of QMSTP (Gao and Lu, 2005), using the Prüfer number to encode a spanning tree. Soak et al. (2005, 2006) report remarkable results with an evolutionary algorithm using an edge-window-decoder strategy. In addition to these early methods, even more heuristics have been proposed in recent years, mostly based on neighborhood search. For example, the Tabu Thresholding algorithm (Öncan and Punnen, 2010) alternatively performs local search and random moves. In Palubeckis et al. (2010), an iterated tabu search (ITS) is proposed and compared to a multi-start simulated annealing algorithm and a hybrid genetic algorithm, showing that ITS performs the best. An artificial bee colony algorithm is developed in Sundar and Singh (2010). Cordone and Passeri (2012) adopt a novel data structure and updating technique to reduce the amortized time of neighborhood exploration from $O(mn^2)$ to $O(mn)$, based on which they further propose a tabu search (TS) algorithm and report a number of improved results over previous best known results. Recently, Lozano et al. (2013) propose an iterated greedy (IG) and a strategic oscillation (SO) heuristic, and combine them with the ITS (Palubeckis et al., 2010) algorithm to obtain a powerful hybrid algorithm named HSII. In addition, for the QMSTP variant only with adjacency costs, Maia et al. develop a Pareto local search (Maia et al., 2013) as well as several evolutionary algorithms (Maia et al., 2014).

In this work, we propose a three-phase search approach named TPS for effectively solving QMSTP, whose main contributions are as follows.

- From the perspective of algorithm design, the proposed TPS approach consists of three distinctive and sequential search phases which are iterated: a descent-based neighborhood search phase (to reach a local optimum from a given starting solution), a local optima exploring phase (to discover more nearby local optima within a given regional area), and a perturbation-based diversification phase (to jump out of the current search area and move to unexplored new areas). At a high abstraction level, TPS shares similar ideas with other popular search frameworks such as iterated local search (Lourenco et al., 2003), reactive tabu search (Bastos and Ribeiro, 2002; Cerulli et al., 2005) and breakout local search (Benlic and Hao, 2013a, 2013b; Fu and Hao, 2014). Still the proposed approach promotes the idea of a clear separation of the search process into three distinctive phases which are iterated, each phase focusing on a well-specified goal with dedicated strategies and mechanisms. The proposed TPS approach also includes two original search strategies designed for QMSTP. The first one is a pre-estimation criterion, which boosts the efficiency of local search by discarding a large number of hopeless neighboring solutions (so as to avoid useless computations). The second one is a new swap-vertex neighborhood, which complements the conventional swap-edge neighborhood and proves to be particularly useful for tackling the challenging and special QMSTP instances transformed from the *Quadratic Assignment Problem* (*QAP*).
- From the perspective of computational results, TPS yields highly competitive results with respect to the best known results and best performing algorithms (tested on 7 sets of 659 benchmarks). Respectively, for the 630 conventional instances, TPS (using the same parameter setting) improves within comparative time the best known results (new upper bounds) on 30 instances and matches easily the best known results for all the remaining instances only except three cases (for which TPS also finds improved best known results within the same cutoff time by simply tuning some parameters). For the set of the 29 instances transformed from QAP which are known to be extremely challenging for existing QMSTP algorithms, TPS consistently attains the known optimal values within very short time.

In the rest of the paper, we describe the proposed approach (Section 2), show extensive computational results on the benchmark instances (Sections 3) and study several key ingredients of the algorithm (Section 4). Conclusions are drawn in Section 5, followed by a parameter analysis in the Appendix.

2. A three-phase search approach for QMSTP

2.1. General framework

The proposed three-phase search approach TPS for QMSTP is outlined in Algorithm 1, which is composed of several subroutines. Starting from an initial solution (generated by *Init_Solution*), the first search phase, ensured by *Descent_Neighborhood_Search*, employs a descent-based neighborhood search procedure to attain a local optimal solution from the input solution. The second search phase *Explore_Local_Optima* is then used to discover nearby local optima of better quality within the current regional search space. If no further improvement can be attained, the search turns into a diversified perturbation phase *Diversified_Perturb*, which strongly modifies the incumbent solution to jump out of the current regional search area and move to a more distant new search area. From this point, the search enters into a new round of *Descent_Neighborhood_Search* and *Explore_Local_Optima* followed by *Diversified_Perturb* search phases. This process is iterated until a given terminal criterion is met (cutoff time, allowed number of iterations, etc.).

Algorithm 1. Framework of the proposed TPS approach for QMSTP.

```
Data: Graph $G(V,E)$, linear function $E \to \mathbb{R}$, quadratic function $E \times E \to \mathbb{R}$
Result: The best solution found
1 $T \leftarrow Init\_Solution()$ ;   /* Construct an initial solution, Section 2.3 */
2 $T^{best} \leftarrow T$ ;          /* $T^{best}$ records the best solution found so far */
3 while The terminal criterion is not met do
      // Find a local optimum, Section 2.4
4     $T \leftarrow Descent\_Neighborhood\_Search(T)$ ;
      // Explore nearby local optima, Algorithm 2 and Section 2.6
5     $T \leftarrow Explore\_Local\_Optima(T)$ ;
      // Update $T^{best}$ if an improved solution is found
6     if $F(T) < F(T^{best})$ then
7         $T^{best} \leftarrow T$ ;
      // Strongly perturb the incumbent solution, Section 2.7
8     $T \leftarrow Diversified\_Perturb(T)$ ;
9 return $T^{best}$ ;
```

Fig. 1 illustrates the idea followed by the TPS procedure, where X-axis indicates all the feasible solutions T, and Y-axis indicates the corresponding objective values $F(T)$. As shown in Fig. 1, A, B, C, D, F, G, I, J, K, L, M are local optima of different qualities, while E, H, N are feasible solutions. Starting from a randomly generated initial solution, say N, the search calls *Descent_Neighborhood_Search* to reach a first local optimum M, and then uses the *Explore_Local_Optima* search phase to discover nearby local optima L and K. At this point, the *Diversified_Perturb* phase is executed to jump from K to a faraway enough solution E, which is subsequently optimized by *Descent_Neighborhood_Search* (E → F) and *Explore_Local_Optima* (F → G), to obtain a high-quality solution G.

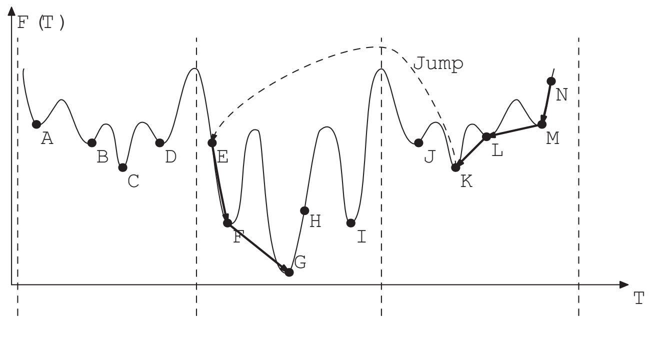

Fig. 1. Procedure of searching a high-quality feasible solution of QMSTP.

2.2. Solution representation

Like the compact tree representation used in Cordone and Passeri (2012) and Fu and Hao (2014), we uniquely represent each feasible solution T as a rooted tree (with vertex 1 fixed as the root vertex, being different from Cordone and Passeri (2012) where the root changes dynamically during the search process), corresponding to a one-dimensional vector $T = \{t_i, i \in V\}$, where t_i denotes the parent vertex of vertex i, with the only exception for the root vertex 1 (let $t_1 = null$). Inversely, given a vector $T = \{t_i, i \in V\}$, the corresponding solution tree can be easily reconstructed.

2.3. Initialization

TPS requires an initial solution to start its search. We use a simple randomized procedure to build initial solutions. Starting from an empty solution T containing only the root vertex, we iteratively select at random one edge from E and add it to T (without leading to a closed loop), until $n-1$ edges are added, meaning that a feasible initial solution (tree) $T = (V, X)$ is generated, where V and $X \subseteq E$ are respectively the vertex set of the graph and the edge set of the tree. In the rest of this paper, we occasionally refer an edge of a directed tree as an arc if needed, to avoid possible confusions.

2.4. Descent-based neighborhood search phase

As the basis of the proposed approach, a descent-based neighborhood search phase *Descent_Neighborhood_Search* is used to reach a local optimum from a given starting solution $T = (V, X)$. For this, we adopt two different move operators to generate neighboring solutions, including a conventional move operator

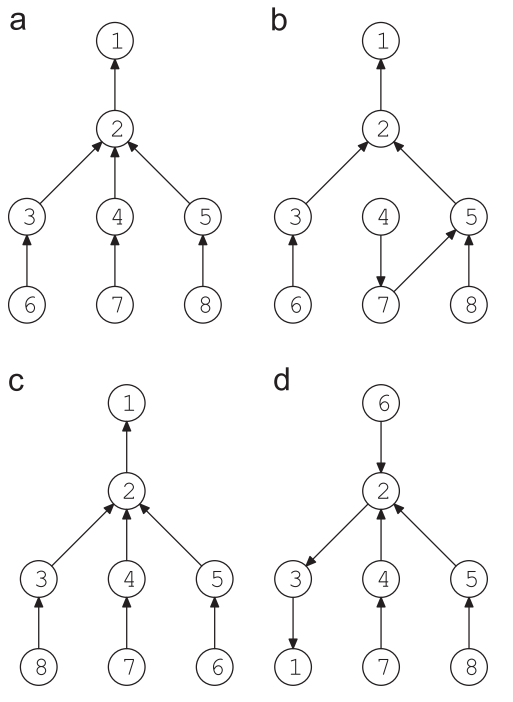

Fig. 2. Move operators for generating neighboring solutions.

(swap-edge) widely used in the literature and an original move operator (swap-vertex) newly introduced in this paper.

1. The first one is the conventional *swap-edge move operator* inherited from Cordone and Passeri (2012), Lozano et al. (2013), and Palubeckis et al. (2010). This operator first adds to X one of the $m-n+1$ unused edges $e \in E \backslash X$, thus closing a loop L_e of $|L_e| \le n$ edges, and then removes an edge f from $L_e \backslash \{e\}$, to obtain a feasible neighboring solution denoted by $T \oplus SwapEdge(e,f)$. The corresponding difference of the objective function (also called move gain) is denoted by δ_{ef}.
2. The above move operator swaps only one pair of edges. It is tempting to introduce a move operator by swapping two pairs of edges to obtain an enlarged neighborhood. Nevertheless, such a move operator induces a neighborhood with a total of $O(m^2n^2)$ neighboring solutions, which is extremely expensive for neighborhood examination. To control the size of the neighborhood, we develop for the first time a restricted *swap-vertex move operator* as follows. Let $V^1 \subseteq V$ denote the subset containing all the vertices with degree in the solution tree equal to 1 (including all the leaf vertices and possibly the root vertex), and for each vertex $i \in V^1$, let r_i denote the related vertex, i.e., the vertex connected to *i*. Then, for each pair of vertices $i,j \in V^1$ with $r_i \ne r_j$ and $\{i,r_j\} \in E, \{j,r_i\} \in E$, a feasible neighboring solution denoted by $T \oplus SwapVertex(i,j)$ could be generated by swapping vertices *i* and *j*, leading to a difference δ_{ij} of the objective function. Note that, if we denote edges (more precisely, arcs) $\{i,r_j\}$, $\{j,r_i\}$, $\{i,r_i\}$, $\{j,r_j\}$ by $e1$, $e2$, $f1$, $f2$ respectively, $SwapVertex(i,j)$ is indeed equivalent to $SwapEdge(e1,f1) \oplus SwapEdge(e2,f2)$. Clearly, $SwapVertex(i,j)$ defines a neighborhood whose size is bounded by $O(n^2)$.

For example, Fig. 2 illustrates several neighboring solutions generated by the above move operators. From the original solution (*a*), solution (*b*) is generated by adding arc $\{5,7\}$ and deleting arc $\{2,4\}$, while solutions (*c*) and (*d*) are obtained by swapping vertices $6-8$ and $1-6$ respectively. Note that, to ensure that vertex 1 is always fixed as the root, before swapping vertex 1 and vertex 6, we should at first traverse all the arcs from vertex 6 to vertex 1 and reverse the parent–child relationship on these arcs (as shown in sub-figure (d)).

Based on these move operators ($SwapEdge(e,f)$ and $SwapVertex(i,j)$), two different neighborhoods $N1$ and $N2$ are defined as follows:

$$N1 = \{T \oplus SwapEdge(e,f) \mid e \in E \backslash X,\ f \in L_e \backslash \{e\}\},$$
$$N2 = \{T \oplus SwapVertex(i,j) \mid i,j \in V^1, r_i \ne r_j, \{i,r_j\} \in E, \{j,r_i\} \in E\}, \quad (5)$$

where $T=(V,X)$ is a feasible solution, and $T \oplus SwapEdge(e,f)$ (respectively, $T \oplus SwapVertex(i,j)$) designates the neighboring solution obtained by applying $SwapEdge(e,f)$ (respectively, $SwapVertex(i,j)$) to T. As shown in Section 4.2, these two basic move operators are effective respectively for solving different types of instances (specifically, the new swap-vertex move is extremely powerful for the challenging instances transformed from QAP with very special structures), thus being adopted in a combined mode as follows.

Typically, at each iteration of *Descent_Neighborhood_Search*, the algorithm first examines in random order neighborhood $N1$ and uses the first met improving neighbor solution (with $\delta_{ef} < 0$) to replace the incumbent solution. If no improving solution exists in $N1$, it turns to examine neighborhood $N2$ in the same manner to accept the first met improving neighbor solution (with $\delta_{ij} < 0$). This process is iterated until no improving solution exists in $N1 \cup N2$, meaning that a local optimum is reached.

Additionally, one observes that $|N1| = O(m-n+1) \times O(\max(|L_e|)) \le O(mn) \le O(n^3)$, and $|N2| = O(|V^1|^2) \le O(n^2)$, being statistically much less than $|N1|$. In the next section, we describe how to quickly evaluate the needed move gains for both neighborhoods. In particular, we further devise a pre-estimation criterion which is able to identify and discard a large number of useless $SwapEdge(e,f)$ moves. As shown in Section 4.1, this pre-estimation criterion ensures a fast exploration of neighborhood $N1$ and considerably reduces the computational complexity of *Descent_Neighborhood_Search*.

2.5. Fast examination technique

Like in Cordone and Passeri (2012), we maintain a vector D, whose components indicate the actual or potential contribution of each edge $g \in E$ to the overall cost of the incumbent solution $T=(V,X)$.

$$D_g = c_g + \sum_{h \in X} (q_{gh} + q_{hg}),\ \forall g \in E. \quad (6)$$

With this vector, for each of the $O(mn)$ possible $SwapEdge(e,f)$ moves of neighborhood $N1$, the change in objective value is given by

$$\delta_{ef} = D_e - D_f - q_{ef} - q_{fe} \quad (7)$$

which can be calculated in constant time $O(1)$ (Cordone and Passeri, 2012). After performing the chosen move $SwapEdge(e,f)$, as in Cordone and Passeri (2012), vector D is updated in $O(m)$ as follows:

$$D_g \leftarrow D_g + q_{ge} + q_{eg} - q_{gf} - q_{fg},\quad \forall g \in E. \quad (8)$$

Clearly, the overall complexity for exploring neighborhood $N1$ at each iteration is bounded by $O(mn) \times O(1) + O(m) = O(mn)$.

Similarly, since each of the $O(n^2)$ possible $SwapVertex(i,j)$ moves of neighborhood $N2$ is equivalent to $SwapEdge(e1,f1) \oplus SwapEdge(e2,f2)$, where $e1$, $e2$, $f1$, $f2$ denote arcs $\{i,r_j\}$, $\{j,r_i\}$, $\{i,r_i\}$, $\{j,r_j\}$ respectively, the difference of the objective function is obtained by

$$\delta_{ij} = D_{e1} + D_{e2} - D_{f1} - D_{f2} + q_{e1e2} + q_{e2e1} + q_{f1f2} + q_{f2f1} - q_{e1f1} - q_{f1e1} - q_{e1f2} - q_{f2e1} - q_{e2f1} - q_{f1e2} - q_{e2f2} - q_{f2e2}. \quad (9)$$

where each term can be evaluated in constant time $O(1)$. Then, vector D is updated in $O(m)$ as follows:

$$D_g \leftarrow D_g + q_{ge1} + q_{e1g} + q_{ge2} + q_{e2g} - q_{gf1} - q_{f1g} - q_{gf2} - q_{f2g},\quad \forall g \in E. \quad (10)$$

Clearly, the computational complexity needed for exploring neighborhood $N2$ at each iteration is at most $O(n^2) \times O(1) + O(m) = O(n^2)$.

Furthermore, we attempt to reduce the computational time needed for examining neighborhood $N1$, which is the most expensive part of the first search phase. As mentioned above, at each iteration of *Descent_Neighborhood_Search*, up to $O(mn)$ legal swap-edge moves are possible. However, many of these moves are definitely hopeless since no improvement over the incumbent solution can be gained. Since *Descent_Neighborhood_Search* only accepts improved solutions with $\delta_{ef} < 0$, it is interesting to identify the hopeless moves with $\delta_{ef} \geq 0$ and discard them directly to avoid irrelevant computations.

Based on this idea, we develop a pre-estimation criterion as follows. Let $\gamma = \text{Max}\{D_g, g \in X\}$ denote the maximum cost value of D_g of all the edges g belonging to the incumbent solution $T=(V,X)$, and let $\lambda = \text{Max}\{q_{hk}+q_{kh},\ h,\ k \in E\}$ denote the maximum possible value of quadratic costs between any pair of edges. Note that γ is a variable which should be updated at each iteration, within an amount of $O(|X|)=O(n)$ extra time, while λ is a constant. Then, it is clear that, for each edge $e \in E \backslash X$, if we add it to X, the objective function would increase by D_e. At this point, one can observe that no matter which edge $f \in L_e \backslash \{e\}$ we choose to remove from X, the decreased cost is strictly bounded within $\gamma+\lambda$. Obviously, if $D_e - \gamma - \lambda \geq 0$, it means that all the possible moves $SwapEdge(e,f), f \in L_e \backslash \{e\}$ lead to a solution no better than the incumbent solution T. In other words, it is definitely impossible to obtain an improved solution by exchanging e against any other edge belonging to the incumbent solution. Consequently, we can directly discard all these moves to avoid useless evaluations, thus reducing the computation time.

While exploring the solutions belonging to neighborhood $N1$, for each edge $e \in E \backslash X$, we first use the above pre-estimation criterion to check if it is possible to gain any improvement by exchanging e against some other edge $f \in L_e \backslash \{e\}$. If this is not the case, we discard all the moves involving e and skip to the next edge in $E \backslash X$. Otherwise, we evaluate one by one the possible legal moves $SwapEdge(e,f), f \in L_e \backslash \{e\}$ to identify an improving neighboring solution. As shown in Section 4.1, the pre-estimation criterion allows the algorithm to discard a high number of hopeless moves, accelerating considerably the neighborhood exploration without sacrifice of solution quality.

2.6. *Local optima exploring phase*

Obviously, the *Descent_Neighborhood_Search* procedure described in Section 2.4 alone cannot go beyond the achieved local optimum. In order to be able to discover nearby local optima which are possibly of better quality and to intensify the search in a given regional search space, we develop a local optima exploring phase (*Explore_Local_Optima*, Algorithm 2), which is based on two directed perturbation operators. Inspired by the idea of breakout local search (Benlic and Hao, 2013a, 2013b), these directed perturbation operators rely on the tabu search principle (Glover and Laguna, 1997), which favors moves with the weakest objective deterioration. In order to distinguish these two perturbation operators from the following diversified perturbation operator (Section 2.7), we call them *Directed_Perturb* operators. Precisely, *Directed_Perturb* takes one of the following two forms.

1. The *swap-edge directed perturbation* operator applies the swap-edge move operator (Section 2.4). For each edge $g \in E$, this perturbation saves in an array the last iteration I_g when edge g is added into or removed from the current solution. With this information, before exchanging edge $e \in E \backslash X$ and edge $f \in L_e \backslash \{e\}$, we first check whether the current iteration index is larger than both $I_e + l_{in}$ and $I_f + l_{out}$, where l_{in} and l_{out} are parameters indicating the length of the prohibition, i.e., the tabu tenures (Glover and Laguna, 1997). If this is not the case, the corresponding move *SwapEdge(e,f)* is marked tabu (otherwise it is declared non-tabu). This prohibition aims to avoid the inclusion of a recently removed edge or the removal of a recently included edge, unless the move meets the aspiration criterion, i.e., leading to a solution better than the overall best found solution. Typically, this perturbation operator examines all the non-tabu legal moves and iteratively applies the best legal move to the incumbent solution (no matter it leads to an improved solution or not), until a given number L_{dir} (called perturbation strength) of such moves are performed.

Algorithm 2. Local optima exploring (*Explore_Local_Optima*) phase.

```
Data: The incumbent local optimal solution $T$, allowed maximum consecutive
      non-improving rounds $\omega_{max}$
Result: The best found local optimal solution near $T$
1  $T^\sharp \leftarrow T$ ;                  /* $T^\sharp$ records the best found local optimum */
2  $\omega \leftarrow 0$ ;  /* $\omega$ counts the number of consecutive non-improving rounds */
3  while $\omega < \omega_{max}$ do
       // Apply a directed perturbation operator to perturb $T$
4      $T \leftarrow Directed\_Perturb(T)$ ;
       // Optimize $T$ to a new local optimum
5      $T \leftarrow Descent\_Neighborhood\_Search(T)$ ;
       // Update the best found solution and the value of $\omega$
6      if $F(T) < F(T^\sharp)$ then
7          $T^\sharp \leftarrow T$ ;
8          $\omega \leftarrow 0$ ;
9      else
10         $\omega \leftarrow \omega + 1$ ;
11 return $T^\sharp$ ;
```

2. The new *swap-vertex directed perturbation* operator is based on the swap-vertex move operator (Section 2.4). For each pair of vertices $i,j \in V$, we save in a two-dimensional array the last iteration I_{ij} when vertex i is swapped with vertex j. Then, before swapping any pair of vertices $i,j \in V^1$, we first check whether the current iteration index is larger than $I_{ij}+l_{swap}$, where l_{swap} is a parameter indicating the tabu tenure. The moves satisfying this condition are marked non-tabu, while the others are declared tabu, unless they meet the same aspiration criterion above. This perturbation operator iteratively applies L_{dir} times the best non-tabu move to the incumbent solution.

In general, these two directed perturbation operators could be used in a combined mode. Nevertheless, as analyzed in Section 4.2, we only apply the swap-edge directed perturbation for solving the conventional instances, and the swap-vertex directed perturbation for solving the special instances transformed from QAP. Whenever a directed perturbation is executed, we call the *Descent_Neighborhood_Search* phase again to the perturbed solution, to obtain a new local optimum. Typically, the local optima exploring phase alternates between *Directed_Perturb* and *Descent_Neighborhood_Search*, until no further improvement is gained after ω_{max} consecutive such rounds (ω_{max} is fixed to be 5 in this paper), meaning that it is difficult to find better local optima within the current search region. At this point, the search turns into a diversified perturbation phase described below, to jump out of the current region.

2.7. Diversified perturbation phase

The diversified perturbation phase aims to jump out of the current regional search area and displace the search to more distancing search areas, while retaining a certain degree of structure information of the incumbent solution. For this, we develop a diversified perturbation operator *Diversified_Perturb*, which iteratively removes at random an edge f from $T=(V,X)$ and subsequently adds the best feasible edge $e \in E \backslash X$ into T, without leading to any closed loop (to ensure the feasibility of the solution after insertion), until a given number L_{div} (parameter for controlling the perturbation strength) of such perturbation moves are performed.

The *Directed_Perturb* and *Diversified_Perturb* operators introduce different degrees of diversification to the search process. Indeed, with tabu principle, *Directed_Perturb* modifies the incumbent solution more gradually and keeps the search within areas close to the incumbent solution. On the other hand, by random moves, *Diversified_Perturb* may disrupt strongly the incumbent solution and leads the search to a more distant new region. By combining these two types of perturbations, it is expected that a better trade-off between intensification and diversification would be reached in the general search procedure.

Finally, as illustrated in Algorithm 1, TPS alternatively calls the above three search phases, until the terminal criterion is satisfied. The best found solution T^{best} is returned as the obtained solution.

2.8. Discussion

TPS borrows ideas from several existing methods like iterated local search (ILS) (Lourenco et al., 2003), reactive tabu search (RTS) (Bastos and Ribeiro, 2002; Cerulli et al., 2005) and breakout local search (BLS) (Benlic and Hao, 2013a, 2013b). We briefly discuss the similarities and differences between TPS and these methods. First, TPS follows the general ILS framework since it alternates between descent phases to locate local optima and perturbation phases to escape from local optima. However, TPS introduces an intermediate phase (*Explore_Local_Optima*) to discover nearby local optima by applying directed perturbations. Second, like RTS, TPS uses the tabu mechanism to forbid visited solutions. However, unlike RTS, TPS uses the tabu mechanism for its directed perturbations and does not adopt any reactive mechanism for its tabu list management. Finally, similar to BLS, TPS distinguishes directed perturbations from diversified perturbations. Yet, TPS organizes these two types of perturbations in a different way: the diversified perturbation phase is triggered only after an *Explore_Local_Optima* phase. By contrast, in BLS, the directed perturbations and diversified perturbations are handled at the same level and managed adaptively. More abstractly, TPS clearly divides the search process into three distinctive phases, while the above methods typically consist of two search phases (intensification phase and diversification phase).

3. Experimental results

In order to evaluate the performance of our TPS algorithm[1] (which is coded in C language and compiled by g++ with the “-O3” option), we test it on *all* the currently existing benchmarks, and compare the results with respect to the state-of-the-art heuristics in the literature. For information, our TPS algorithm is executed on an Intel Xeon E5440 2.83 GHz processor with 2 GB RAM, while a 1.6 GHz Pentium IV processor is used in Soak et al. (2006), a 3 GHz Pentium IV CPU with 2 GB RAM in Öncan and Punnen (2010), a 3.0 GHz core 2 duo system with 2 GB RAM in Sundar and Singh (2010), a 3.0 GHz Intel core 2 duo in Palubeckis et al. (2010), a 2.6 GHz Intel Pentium Core 2 Duo E6700 and 2 GB RAM in Cordone and Passeri (2012), a 3.2 GHz Intel processor with 12 GB RAM in Lozano et al. (2013). One can observe that the clock frequency of our processor is about 80% faster than the computer used in Soak et al. (2006), while being similar to the machines used in Öncan and Punnen (2010), Sundar and Singh (2010), Palubeckis et al. (2010), Cordone and Passeri (2012), and Lozano et al. (2013). Note that our TPS algorithm is a sequential algorithm and runs on one single core of the processor. For the reference algorithms, no precise information is known in this regard.

Given that the reference algorithms are executed on different platforms with different configurations, it seems impossible to exactly compare the computational times. For this reason, we focus our assessment on solution quality achieved by our TPS algorithm (within reasonable runtime), with respect to the existing state-of-the-art algorithms. Nevertheless, following the newest QMSTP references (Cordone and Passeri, 2012; Lozano et al., 2013; Pereira et al., 2013a), we include for indicative purposes the runtimes of the compared algorithms, which could still provide some rough indications about the computational efficiency of each algorithm.

3.1. Benchmark instances

Given the importance of QMSTP, a large number of benchmark instances (to the best of our knowledge, 659 instances in total) are currently available in the literature, corresponding to different problem sizes (from small graphs to large graphs), different network densities (from sparse graphs to complete graphs), and different types of quadratic costs (randomly distributed or artificially distributed). Given the diversity of these instances, they form a reasonable basis to evaluate QMSTP algorithms. One notes that previous algorithms of the literature only report computational results on some of these 659 instances (see Sections 3.3–3.8). To

[1] All the best solutions reported in this paper are available on url http://www.info.univ-angers.fr/pub/hao/qmstp.html, the TPS source code will also be made available upon the publication of the paper.

assess our TPS algorithm thoroughly, we evaluate TPS on the *whole set of all the 659 benchmarks*. For convenience, we classify the QMSTP benchmarks into seven groups as follows[2]:

- Benchmark CP (Cordone and Passeri, 2008) consists of 108 instances, with vertices number n ranging from 10 to 50, and graph density $\rho = 33\%$, 67% or 100%. The linear costs and the quadratic costs are randomly distributed in [1,10] or [1,100].
- Benchmark OP1 (Öncan and Punnen, 2010) consists of 480 complete graphs, with n=6–18, 20, 30, 50 respectively, each group having 30 instances. These instances are further divided into three subclasses:
  - SYM: with linear costs uniformly distributed at random within [1,100], and quadratic ones within [1,20];
  - VSYM: the linear costs are uniformly distributed at random in [1,10,000], for the quadratic costs, each vertex is assigned with a value randomly distributed in [1,10] and the quadratic cost q_{ef} is obtained by multiplying the four values associated with the end vertices of edges e and f;
  - ESYM: the vertices are randomly distributed in a square of side 100, then the linear costs are the Euclidean distances between the end vertices of each edge, and the quadratic costs are the Euclidean distances between the mid-points of the edges.
- Benchmark SCA (Soak et al., 2006) includes 6 complete graphs, with vertices number ranging from 50 to 100, by steps equal to 10. For each instance, the vertices are uniformly spread in a square of side 500, then the linear costs are the Euclidean distances between the vertices and the quadratic costs are uniformly distributed within [0,20].
- Benchmark SS (Sundar and Singh, 2010) consists of 18 complete graphs with $n=$ 25, 50, 100, 150, 200 and 250 (each corresponds to 3 instances), the linear costs are uniformly distributed at random within [1,100] and the quadratic costs are randomly distributed within [1,20].
- Benchmark RAND (Lozano et al., 2013) consists of 9 large instances (with $n=$ 150, 200 or 250) recently generated by Lozano et al., with linear costs uniformly distributed in [1, 100], and quadratic ones uniformly distributed in [1,20].
- Benchmark SOAK (Lozano et al., 2013) includes 9 large instances (with $n=$ 150, 200 or 250), with vertices uniformly distributed at random on a 500×500 grid. The edge costs are the integer Euclidean distances between any pair of vertices, and the quadratic ones are uniformly distributed between [1,20].
- Benchmark QAP-QMSTP (originally named OP2) consists of 29 special QMSTP instances converted from the NUG (Nugent et al., 1968) and CHR (Christofides and Benavent, 1989) benchmarks of the Quadratic Assignment Problem (QAP), using a one-to-one transformation procedure between the two problems (Öncan and Punnen, 2010). Note that, although the original QAP instances have already been solved to optimality by previous QAP algorithms (Burkard et al., 1997), they are difficult for existing QMSTP algorithms to reach the optimal results, due to the special problem structures after transformation. Even the best QMSTP algorithm misses 17 optimal solutions.

The largest QMSTP instances have up to 250 vertices (complete graphs) and are extremely difficult for QMSTP algorithms to reach optimality, due to their $O(|E|) \times O(|E|) = O(|V|^4)$ quadratic costs. Indeed, for a complete graph with 250 vertices, the size of the input file is about 1.15 GB, implying that once the number of vertices increases to 1000, the input file size will increase to be unreasonably large (over 290 GB). On the other hand, for most of the existing instances with more than 100 vertices, the current best known results (reported by previous algorithms) can possibly be further improved (as we show below), confirming the difficulties to reach their optimal solutions.

Table 1
Default setting of each parameter.

Parameter	Description	Default setting
l_{in}	Tabu tenure, Section 2.6	$[1, 3]$
l_{out}	Tabu tenure, Section 2.6	$[0.3n, 0.4n]$
l_{swap}	Tabu tenure, Section 2.6	$[n, 2n]$
L_{dir}	Strength of directed perturbation, Section 2.6	$[0.5n, 2n]$
L_{div}	Strength of diversified perturbation, Section 2.7	$[n, 5n]$

One notes that all the instances of the CP and OP1 groups are small-sized, with up to 50 vertices. For all the 108 instances of group CP and almost all (476 out of 480) the instances of group OP1, our TPS algorithm can easily match the optimal or current best known results. Moreover, for the left 4 instances of group OP1, TPS succeeds in finding improved best solutions. The other five groups of instances are much more challenging, due to their large-scale sizes or special structures. To emphasize the effectiveness of TPS for solving challenging instances, we provide the detailed results of TPS on the five challenging groups with respect to previous state-of-the-art heuristics, while showing the results on groups CP and OP1 in a summarized form.

3.2. Parameters

As described in Section 2, TPS requires several parameters: the tabu tenures l_{in}, l_{out}, l_{swap} used in the directed perturbation operators, as well as the perturbation strengths L_{dir} and L_{div}. Generally, these parameters could be tuned with respect to each benchmark group given that the groups have different characteristics and structures. However, to show the efficiency and the robustness of the proposed approach, we uniformly adopt a fixed set of parameter values for all the test instances even if better results could be achieved by finely tuning some parameters.

Following the analysis detailed in the Appendix, we choose an interval as the default setting of each parameter (shown in Table 1). During the search process of TPS, whenever a parameter value is needed, a value is taken at random within the corresponding interval. Additionally, one notes that in the literature, various stop conditions have been adopted by the QMSTP algorithms for solving different groups of instances. To ensure an assessment of our TPS algorithm as fair as possible, we set the stop criteria for TPS relative to the reference algorithms as follows.

3.3. Results of the CP instances

This group contains 108 instances with up to 50 vertices, among which 42, 23, 36 largest instances are respectively tested by ITS (Palubeckis et al., 2010), QMST-TS (Cordone and Passeri, 2012), and HSII (Lozano et al., 2013) to evaluate their performances. Both ITS and QMST-TS solve each instance 10 times, each run continues until the previous best known solution is reached.[3] Experimental results show that for each of their selected instances, each run of ITS and QMST-TS can reach the best known result, with a mean computing time ranging from less than one second to about two minutes. HSII also executes 10 independent times to

[2] The *CP* benchmarks can be downloaded from url http://www.dti.unimi.it/cordone/research/qmst.html and the RAND and SOAK instances are available at url http://sci2s.ugr.es/qmst/QMSTPInstances.rar. The others can be provided on request to the authors (fu@info.univ-angers.fr or hao@info.univ-angers.fr).

[3] The previous best known results for the 108 CP instances are available at url http://www.dti.unimi.it/cordone/research/qmst.html.

Table 2
Four improved results of the SYM subclass of group OP1.

$\vert V\vert$	$\vert E\vert$	Index	QMST-TS (Cordone and Passeri, 2012)		TPS						
			Best	*T (s)*	*Best*	*T (s)*	*t (s)*	σ	$\surd$	$=$	$\times$
50	1225	2	17,600	25.38	**17,587***	4.88	0.49	0.19	3	0	7
50	1225	7	17,643	25.29	**17,633***	4.00	0.40	0.14	5	0	5
50	1225	8	17,685	25.11	**17,663***	5.32	0.53	0.18	6	0	4
50	1225	10	17,639	25.35	**17,623***	5.71	0.57	0.17	5	0	5
Average	–	–	17,641.8	25.28	**17,626.5***	4.98	0.50	0.17	4.75	0	5.25

Table 3
Results of the SCA instances.

$\vert V\vert$	EWD (Soak et al., 2006)		RLS-TT (Öncan and Punnen, 2010)		ABC (Sundar and Singh, 2010)		QMST-TS (Cordone and Passeri, 2012)		TPS						
	Best	*T (s)*	*Best*	*T (s)*	*Best*	*T (s)*	*Best*	*T (s)*	*Best*	*T (s)*	*t (s)*	σ	$\surd$	$=$	$\times$
50	25,339	343.0	25,226	3242.1	**25,200**	87.0	**25,200**	44.5	**25,200**	5.6	0.3	0.0	0	20	0
60	36,086	495.7	35,754	4321.4	35,466	169.0	**35,447**	83.2	**35,447**	12.6	0.6	0.1	0	20	0
70	48,538	716.6	48,536	5738.5	**48,125**	337.2	**48,125**	178.5	**48,125**	21.3	1.1	0.2	0	20	0
80	63,546	1086.7	63,546	7026.3	63,022	417.8	63,004	340.4	**62,963***	41.1	2.1	0.6	16	0	4
90	79,627	1337.2	79,922	8623.6	**78,879**	751.8	78,912	579.7	**78,879**	55.1	2.8	0.8	0	6	14
100	98,342	1828.9	98,811	10,431.3	**96,750**	1542.4	96,757	789.4	**96,750**	89.2	4.5	1.3	0	10	10
Average	58,579.7	968.0	58,632.5	6563.9	57,907.0	550.9	57,907.5	336.0	**57,894.0***	37.5	0.38	0.5	2.7	12.7	4.7

solve each of its selected instances, with a cutoff time of 10 s for each run. However, for many selected instances, HSII occasionally fails to match the previous best known results within the allowed time. On the other hand, the best existing exact algorithms (Pereira et al., 2013a) can solve all the instances with up to 20 vertices and 127 edges to optimality with a time limit of 100 h. Recently, Rostami and Malucelli (2014) provide lower bounds on all these 108 instances.

To evaluate the performance of our TPS algorithm on this set of 108 instances, we independently run TPS 10 times to solve each instance, each run continues until the optimal result (for instances with known optima) or the best known result (for the remaining instances) is reached. Our results show that, each TPS run unexceptionally succeeds in reaching the optimal or the best known result, with an average time from less than one second to less than one minute, indicating that TPS performs similarly with respect to ITS and QMST-TS for this group of small benchmarks. Since these instances are not challenging enough, we do not list our detailed results.

3.4. *Results of the OP1 instances*

This group consists of three subclasses (SYM, ESYM, VSYM), each includes 160 instances, with $|V|$ ranging from 6 to 50 (a total of 480 instances). These benchmarks have been used to evaluate many algorithms, including several ones which aim to provide optimal solutions or strong lower bounds, i.e., the refined Lagrangian lower bounding procedure in Öncan and Punnen (2010), the B&B algorithm QMST-BB in Cordone and Passeri (2012), the enhanced B&B algorithms in Pereira et al. (2013a), and the reformulation scheme for computing lower bounds in Rostami and Malucelli (2014). Nevertheless, even the best existing exact approach (Pereira et al., 2013a) can only solve a subset of these instances to optimality, i.e., the SYM instances with up to 18 vertices, and the VSYM and ESYM instances with up to 50 vertices, with computational time ranging from less than one second to more than five hours.

There are also two heuristics which aim to provide sub-optimal solutions within reasonable time, i.e., the RLS-TT algorithm in Öncan and Punnen (2010) and the tabu search algorithm QMST-TS in Cordone and Passeri (2012). For the instances of subclass SYM with $20 \le |V| \le 50$, only heuristics are able to produce feasible solutions within reasonable time. Note that the RLS-TT heuristic just provides summarized results on this group of 480 benchmarks, without giving detailed results for each instance. Unfortunately, as pointed out in Cordone and Passeri (2012) and Pereira et al. (2013a), some of the results reported by RLS-TT exhibit internal inconsistencies, probably due to typographical errors. It means that it is impossible to reproduce the results reported by RLS-TT on the inconsistent instances.

To ensure that the computation time required by our TPS approach is comparable to the existing approaches, we independently run TPS 10 times to solve each instance, each run continues until the best found solution cannot be further improved after 10 consecutive rounds of *Descent_Neighborhood_Search* and *Explore_Local_Optima* search phases followed by *Diversified_Perturb*, or up to 50 such rounds have been applied. Our results show that, for all these 480 instances, TPS succeeds in matching the optimal results (for instances with known optima) or finding solutions no worse than the previous best known solutions (for instances with unknown optima),[4] with an accumulated CPU time ranging from less than one second to about five seconds.

Most importantly, for four largest SYM instances with unknown optima (with $|V| = 50$), TPS succeeds in improving the current best solutions (new upper bounds) of the literature. The detailed results of these four instances are given in Table 2, where the first three columns are about the instances, and the next two columns respectively report the best objective value and the accumulated CPU time (in seconds) of QMST-TS. The last seven columns show performance information of TPS, including the best objective value ('Best'), the accumulated CPU time of 10 runs (in seconds, 'T(s)'), the average CPU time of one run (in seconds, column 't(s)'), the standard deviation on runtime (column σ), as well as the times that TPS improves (column $\surd$), matches (column $=$) or misses (column $\times$) the best known result among 10 runs. The best results reported by all the competing algorithms (including our TPS algorithm) are indicated in **bold**, and once our TPS yields a result

[4] All the previous best known results of the OP1 instances could be downloaded from url http://www.dti.unimi.it/cordone/research/qmst.html.

Table 4
Results of the SS instances.

$\|V\|$	Index	ABC (Sundar and Singh, 2010)		QMST-TS (Cordone and Passeri, 2012)		TPS						
		Best	*T (s)*	*Best*	*T (s)*	*Best*	*T (s)*	*t (s)*	σ	$\surd$	=	×
25	1	**5085**	18.2	**5085**	6.7	**5085**	0.4	0.02	0.01	0	20	0
25	2	**5081**	20.4	**5081**	6.6	**5081**	0.6	0.03	0.01	0	19	1
25	3	**4962**	21.0	**4962**	6.9	**4962**	0.4	0.02	0.00	0	20	0
50	1	**21,126**	173.6	**21,126**	50.4	**21,126**	5.5	0.28	0.10	0	14	6
50	2	21,123	176.8	**21,106**	50.4	**21,106**	5.5	0.28	0.09	0	8	12
50	3	**21,059**	190.2	**21,059**	50.6	**21,059**	4.2	0.21	0.06	0	16	4
100	1	89,098	2333.2	88,871	965.8	**88,745⁎**	81.2	4.06	1.12	2	0	18
100	2	89,202	2319.0	89,049	957.7	**88,911⁎**	76.4	3.82	1.62	7	0	13
100	3	89,007	1977.6	88,720	961.2	**88,659⁎**	87.5	4.37	1.76	3	0	17
150	1	205,619	8897.4	205,615	2928.7	**204,995⁎**	510.6	25.53	9.70	12	0	8
150	2	205,874	7486.6	205,509	2923.0	**205,219⁎**	493.7	24.68	10.56	4	0	16
150	3	205,634	8658.6	205,094	2928.6	**205,076⁎**	596.7	29.84	10.73	1	0	19
200	1	371,797	22,828.4	371,492	6320.3	**370,873⁎**	1519.5	75.98	36.28	9	0	11
200	2	371,864	23,112.0	371,698	6332.1	**370,853⁎**	1386.3	69.32	25.17	15	0	5
200	3	372,156	25,534.2	371,584	6324.3	**370,954⁎**	1282.8	64.14	21.80	9	0	11
250	1	587,924	51,268.2	586,861	9572.3	**586,265⁎**	2916.5	145.82	45.36	7	0	13
250	2	588,068	56,818.2	587,607	9592.9	**586,778⁎**	2253.3	112.66	31.68	17	0	3
250	3	587,883	46,565.8	587,281	9601.2	**585,851⁎**	2709.9	135.49	53.58	14	0	6
Average	–	213,475.7	14,355.5	213,211.1	3310.0	**212,866.6⁎**	774.0	38.70	13.87	5.56	5.39	9.06

Table 5
Results of the RAND instances.

Instance	ABC (Sundar and Singh, 2010)		ITS (Palubeckis et al., 2010)		HSII (Lozano et al., 2013)		TPS						
	Best	*T (s)*	*Best*	*T (s)*	*Best*	*T (s)*	*Best*	*T (s)*	*t (s)*	σ	$\surd$	=	×
RAND-150-1	194,294	4000	192,946	4000	192,606	4000	**192,369⁎**	4000	400	0	2	0	8
RAND-150-2	194,218	4000	193,034	4000	192,607	4000	**192,579⁎**	4000	400	0	1	0	9
RAND-150-3	193,882	4000	192,965	4000	192,577	4000	**192,046⁎**	4000	400	0	4	0	6
RAND-200-1	353,163	12,000	351,216	12,000	350,517	12,000	**350,321⁎**	12,000	1200	0	3	0	7
RAND-200-2	353,784	12,000	351,312	12,000	350,389	12,000	350,658[1]	12,000	1200	0	0	0	10
RAND-200-3	353,169	12,000	351,466	12,000	351,057	12,000	**350,601⁎**	12,000	1200	0	7	0	3
RAND-250-1	561,864	20,000	558,451	20,000	556,929	20,000	557,278[2]	20,000	2000	0	0	0	10
RAND-250-2	560,704	20,000	558,820	20,000	557,474	20,000	**556,604⁎**	20,000	2000	0	1	0	9
RAND-250-3	561,497	20,000	559,304	20,000	556,813	20,000	557,060[3]	20,000	2000	0	0	0	10
Average	369,619.4	12,000	367,723.8	12,000	366,774.3	12,000	**366,612.9⁎**	12,000	1200	0	2.0	0.0	8.0

[1] TPS achieves an improved value of **350,231** (within 12,000 s) with re-tuned parameters $L_{dir} \in [0.1n, n]$ and $L_{div} \in [n, 2n]$
[2] TPS achieves an improved value of **556,596** (within 20,000 s) with re-tuned parameter $L_{div} \in [0.5n, 2n]$
[3] TPS achieves an improved value of **556,378** (within 20,000 s) with re-tuned parameter $L_{div} \in [0.5n, 5n]$

Table 6
Results of the SOAK instances.

Instance	ABC (Sundar and Singh, 2010)		ITS (Palubeckis et al., 2010)		HSII (Lozano et al., 2013)		TPS						
	Best	*T (s)*	*Best*	*T (s)*	*Best*	*T (s)*	*Best*	*T (s)*	*t (s)*	σ	$\surd$	=	×
SOAK-150-1	207,652	4000	**206,721**	4000	206,925	4000	**206,721**	4000	400	0	0	1	9
SOAK-150-2	208,206	4000	**206,761**	4000	207,102	4000	**206,761**	4000	400	0	0	5	5
SOAK-150-3	207,533	4000	206,802	4000	206,781	4000	**206,759⁎**	4000	400	0	2	3	5
SOAK-200-1	372,419	12,000	370,137	12,000	370,265	12,000	**369,851⁎**	12,000	1200	0	5	0	5
SOAK-200-2	371,641	12,000	370,028	12,000	369,982	12,000	**369,803⁎**	12,000	1200	0	8	0	2
SOAK-200-3	372,117	12,000	370,046	12,000	370,045	12,000	**369,794⁎**	12,000	1200	0	7	0	3
SOAK-250-1	584,799	20,000	582,282	20,000	581,819	20,000	**581,671⁎**	20,000	2000	0	1	0	9
SOAK-250-2	584,409	20,000	582,145	20,000	581,691	20,000	**581,492⁎**	20,000	2000	0	2	0	8
SOAK-250-3	585,717	20,000	582,708	20,000	581,854	20,000	**581,573⁎**	20,000	2000	0	1	0	9
Average	388,277.0	12,000	386,403.3	12,000	386,273.8	12,000	**386,047.2⁎**	12,000	1200	0	2.9	1.0	6.1

better than the previous best known result, it is indicated with a symbol '⁎'. Finally, the last row indicates the average value of each column (marked as '–', if not applicable).

3.5. Results of the SCA instances

This set include six old instances, which are generated by Soak et al. (2006) and have been widely used by various heuristics (Soak et al., 2006; Öncan and Punnen, 2010; Sundar and Singh, 2010; Cordone and Passeri, 2012). For each of these instances, we independently run TPS 20 times (like QMST-TS Cordone and Passeri, 2012, each run using the same stop criterion as for group OP1. The obtained results are given in Table 3, with respect to the results reported by four reference heuristics: EWD (Soak et al., 2006), RLS-TT (Öncan and Punnen, 2010), ABC (Sundar and Singh, 2010), QMST-TS (Cordone and Passeri, 2012). The first column provides

Table 7
Results of *TPS* on the QAP-QMSTP instances compared with RLS-TT (Öncan and Punnen, 2010), ABC (Sundar and Singh, 2010), ITS (Palubeckis et al., 2010), HSII (Lozano et al., 2013) and QMST-TS with re-tuned parameters (Cordone and Passeri, 2012).

Instance	Opt.	RLS-TT		ABC		ITS		HSII		QMST-TS		TPS					
		Best	*T* (*s*)	*Best*	*T* (*s*)	*Best*	*T* (*s*)	*Best*	*T* (*s*)	*Best*	*T* (*s*)	*Best*	*T* (*s*)	*t* (*s*)	σ	=	×
chr12a	9552	11,170	783	14,290	10,000	16,694	10,000	**9552**	10,000	**9552**	288	**9552**	43	4.3	0.1	10	0
chr12b	9742	10,753	790	21,552	10,000	16,356	10,000	**9742**	10,000	**9742**	287	**9742**	37	3.7	0.0	10	0
chr12c	11,156	12,712	783	15,810	10,000	17,434	10,000	**11,156**	10,000	**11,156**	286	**11,156**	44	4.4	0.2	10	0
chr15a	9896	11,638	1239	24,224	10,000	16,718	10,000	9952	10,000	9936	497	**9896**	92	9.2	0.5	10	0
chr15b	7990	10,145	1136	28,340	10,000	17,208	10,000	8384	10,000	**7990**	492	**7990**	90	9.0	0.3	10	0
chr15c	9504	12,769	1254	25,566	10,000	19,302	10,000	**9504**	10,000	**9504**	492	**9504**	91	9.1	0.4	10	0
chr18a	11,098	12,757	3325	24,954	10,000	22,496	10,000	13,834	10,000	**11,098**	793	**11,098**	170	17.0	1.0	10	0
chr18b	1534	1676	3354	21,60	10,000	**1534**	10,000	**1534**	10,000	**1534**	789	**1534**	148	14.8	0.2	10	0
chr20a	2192	2445	4968	4742	10,000	2232	10,000	2276	10,000	**2192**	1043	**2192**	356	35.6	8.0	6	4
chr20b	2298	2730	4652	3704	10,000	2440	10,000	2462	10,000	2352	1044	**2298**	482	48.2	15.2	3	7
chr20c	14,142	30,124	4763	49,842	10,000	36,558	10,000	20,206	10,000	14,202	1046	**14,142**	260	26.0	0.8	10	0
chr22a	6156	8760	5089	8688	10,000	6390	10,000	6334	10,000	6228	1395	**6156**	450	45.0	4.8	10	0
chr22b	6194	8402	4741	8908	10,000	6314	10,000	6396	10,000	6314	1422	**6194**	581	58.1	15.0	3	7
chr25a	3796	9658	5223	8540	10,000	4300	10,000	4310	10,000	3866	2135	**3796**	841	84.1	24.0	7	3
nug12	578	605	639	656	10,000	**578**	10,000	**578**	10,000	**578**	287	**578**	40	4.0	0.1	10	0
nug14	1014	1084	724	1140	10,000	**1014**	10,000	1026	10,000	**1014**	414	**1014**	67	6.7	0.1	10	0
nug15	1150	1265	1348	1404	10,000	**1150**	10,000	1152	10,000	**1150**	493	**1150**	78	7.8	0.0	10	0
nug16a	1610	1742	2311	1944	10,000	1638	10,000	1634	10,000	1622	583	**1610**	102	10.2	0.3	10	0
nug16b	1240	1350	2936	1480	10,000	1248	10,000	1246	10,000	**1240**	579	**1240**	97	9.7	0.1	10	0
nug17	1732	1874	3422	2066	10,000	1768	10,000	1774	10,000	1750	685	**1732**	123	12.3	0.3	10	0
nug18	1930	2056	3482	2224	10,000	1964	10,000	1984	10,000	1942	793	**1930**	152	15.2	0.3	10	0
nug20	2570	2860	5151	2900	10,000	2644	10,000	2662	10,000	2580	1047	**2570**	228	22.8	0.3	10	0
nug21	2438	2698	5184	3042	10,000	2502	10,000	2540	10,000	2488	1201	**2438**	248	24.8	1.6	10	0
nug22	3596	3868	5482	4580	10,000	3712	10,000	3750	10,000	3672	1371	**3596**	317	31.7	0.6	10	0
nug24	3488	3874	5914	4340	10,000	3648	10,000	3688	10,000	3590	1838	**3488**	410	41.0	1.4	10	0
nug25	3744	4083	5983	4522	10,000	3954	10,000	3940	10,000	3874	2098	**3744**	475	47.5	5.2	10	0
nug27	5234	5966	6025	6284	10,000	5456	10,000	5534	10,000	5352	2788	**5234**	651	65.1	7.6	10	0
nug28	5166	5819	6087	6238	10,000	5406	10,000	5484	10,000	5262	3228	**5166**	771	77.1	6.8	10	0
nug30	6124	6923	6227	7688	10,000	6506	10,000	6528	10,000	6364	4283	**6124**	1251	125.1	21.5	10	0
Average	5064.3	6614.0	3552.2	10,063.0	10,000.0	7902.2	10,000.0	5488.3	10,000.0	5108.4	1161.9	**5064.3⁎**	299.8	30.0	4.0	9.3	0.7

Table 8
Improve percentage on each group of instances.

Algorithms\instances	SCA (%)	SS (%)	RAND (%)	SOAK (%)	QAP-QMSTP (%)
TPS ↔ EWD	1.11	–	–	–	–
TPS ↔ RLS-TT	1.02	–	–	–	15.34
TPS ↔ ABC	0.02	0.21	0.84	0.56	31.79
TPS ↔ ITS	–	–	0.30	0.07	15.17
TPS ↔ QMST-TS	0.02	0.10	–	–	0.99
TPS ↔ HSII	–	–	0.06	0.07	4.52

the problem size $|V|$, while the following eight columns indicate the best found results and the accumulated CPU times (in seconds) of each competing algorithm, and the last seven columns show the results of our TPS algorithm, with information similar to the last seven columns of Table 2. The average of each column is listed in the last row.

Table 3 discloses that for all these 6 instances, TPS steadily (indicated by a small value of σ) improves or matches the previous best results within a short time. Moreover, for the instance with $|V|=80$, TPS repeatedly (16 times out of the 20 independent runs) improves the best known result. It also improves 6, 6, 2, 3 results compared to EWD, RLS-TT, ABC, QMST-TS, respectively. Finally, TPS yields a better averaged objective value, indicating its effectiveness on these old benchmarks.

3.6. Results of the SS instances

These 18 instances have been used to test ABC (Sundar and Singh, 2010) and QMST-TS (Cordone and Passeri, 2012). For comparison, for each of these instances, we independently run TPS 20 times (like QMST-TS), each run stops if the best found solution cannot be further improved for 5 consecutive rounds of the three search phases or up to 40 such rounds have been applied. The obtained results are listed in Table 4, where the first two columns identify each instance, the next four columns indicate the best results and the total computational time in seconds of ABC and QMST-TS, and the last seven columns report the information for our TPS algorithm like in Table 3. The averaged results are also listed in the last row.

Table 4 discloses that for all the 12 instances with $|V| \geq 100$, TPS can repeatedly find improved results over the best known results, while for the left 6 smaller instances with $|V| \leq 50$, TPS can easily reach the best known results. Unsurprisingly, TPS obtains a best averaged objective value. Moreover, TPS requires reasonable (short) total run-times with small standard deviations.

3.7. Results of the RAND and SOAK instances

Recently, Lozano et al. (2013) propose a hybrid heuristic named HSII and evaluate its performance using two groups (RAND and SOAK) of 18 newly generated benchmarks, in comparison with two previous heuristics, i.e., ITS (Palubeckis et al., 2010) and ABC (Sundar and Singh, 2010).[5] For each instance, the above three algorithms are respectively executed 10 independent times, each run stops using a time limit that varies according to the problem size (400, 1200, 2000 s respectively for instances with $|V|=150$,

[5] ITS and ABC did not report results on groups RAND, SOAK, and QAP-QMSTP. In order to test ITS and ABC on these benchmarks, Lozano et al. use the source code of ITS from url http://www.soften.ktu.lt/gintaras/qmstp.html and re-implement the ABC algorithm, and then compare the obtained results with the HSII algorithm using the same computing platform.

Table 9
Friedman test results on each group of instances, based on the best objective values.

Algorithms\instances	SCA	SS	RAND	SOAK	QAP-QMSTP
TPS $\leftrightarrow$ EWD	1.43×10^{-2}	–	–	–	–
TPS $\leftrightarrow$ RLS-TT	1.43×10^{-2}	–	–	–	7.24×10^{-8}
TPS $\leftrightarrow$ ABC	1.57×10^{-1}	3.12×10^{-4}	2.70×10^{-3}	2.70×10^{-3}	7.24×10^{-8}
TPS $\leftrightarrow$ ITS	–	–	2.71×10^{-3}	8.15×10^{-3}	5.73×10^{-7}
TPS $\leftrightarrow$ QMST-TS	8.32×10^{-2}	5.32×10^{-4}	–	–	3.74×10^{-5}
TPS $\leftrightarrow$ HSII	–	–	3.17×10^{-1}	2.69×10^{-3}	1.62×10^{-6}

200, 250, uniformly on a 3.2 GHz Intel processor with 12 GB RAM). To evaluate our TPS algorithm under a comparable condition, we also independently run TPS 10 times to solve each instance, using the same cutoff time like in Lozano et al. (2013) for each run (we use a computer with an Intel Xeon E5440 2.83 GHz processor and 2 GB RAM). The obtained results are provided in Tables 5 and 6 (with the averaged values given in the last row), where the meanings of the columns are similar to those in previous tables.

On one hand, as listed in Table 5, for six out of the nine instances of group RAND, TPS succeeds in finding an improved solution over the compared algorithms, while for the left three instances (*Rand-200-2*, *Rand-250-1* and *Rand-250-3*, marked as "†" in the table), TPS fails to match the previous best known results within the limited runtime. However, as shown at the bottom of Table 5, for each of these three instances, an improved solution over the best known result can be found (within the same cutoff time) by TPS with re-tuned parameters. On the other hand, for the nine SOAK instances (Table 6), TPS improves seven best known results and matches the left two results. Finally, considering the averaged objective value, we observe that TPS performs the best on both these two groups, indicating its overall competitiveness in terms of solution quality with respect to the existing algorithms.

3.8. *Results of the QAP-QMSTP instances*

This group of 29 special QMSTP instances are transformed from QAP (including 14 CHR ones (Christofides and Benavent, 1989) and 15 NUG ones (Nugent et al., 1968), while guaranteeing a one-to-one correspondence of the feasible solutions after transformation (Öncan and Punnen, 2010). For each of these 29 instances, we independently run TPS 10 times, each run continues until the best found solution cannot be further improved after 500 consecutive rounds of the three search phases, to ensure that the accumulated runtime remains comparable with respect to the compared heuristics.

Table 7 lists in detail the obtained results. The first two columns show the instance name and its optimal value known from the QAP literature (Burkard et al., 1997).[6] The next 10 columns report the best results and the accumulated CPU times (in seconds) of each compared algorithm, i.e., RLS-TT (Öncan and Punnen, 2010), ABC (Sundar and Singh, 2010), ITS (Palubeckis et al., 2010), HSII (Lozano et al., 2013) and QMST-TS with re-tuned parameters (Cordone and Passeri, 2012). Note that, like for the RAND and SOAK instances, the results of ABC and ITS are reproduced by Lozano et al. (2013). The last six columns indicate the results of our TPS algorithm with information similar to previous tables (column $\surd$ is removed from the table, since all these 29 instances have known optimal results, thus being obviously impossible to be improved). Again, the last row gives the averaged value of each column.

[6] The optimal solutions of the original QAP instances are available online at the QAPLIB: url http://www.seas.upenn.edu/qaplib.

Table 10
Importance of the pre-estimation criterion.

Group	Total ($\times10^3$)	Discarded ($\times10^3$)	Ratio (%)
CP	185,163	164,617	88.9
OP1	20,088	14,427	71.8
SCA	38,567	35,869	93.0
SS	1,064,156	1,033,653	97.1
RAND	5,339,866	5,198,101	97.3
SOAK	5,062,492	4,921,408	97.2
QAP-QMSTP	14,840	14,419	97.2

From Table 7, one observes that the previous QMSTP algorithms RLS-TT, ABC, ITS, HSII, QMST-TS respectively miss 29, 29, 25, 23, 17 optimal solutions. On the contrary, for all these instances, our TPS algorithm can consistently match (within very short time) the optimal solutions. This is the first time a QMSTP algorithm manages to attain all the optimal values of these particular instances, clearly indicating its effectiveness and competitiveness.

3.9. *Summarized results*

We now summarize the overall performance of TPS on each group of instances, with respect to the reference algorithms. First, we calculate the mean improvement percentage of TPS over each compared algorithm, which is defined as $(\sum_{I\in Gr}(F1(I){-}F2(I))/F1(I))/|\mathrm{Gr}|$, where Gr, I, $F1(T)$, $F2(T)$ are respectively a benchmark group, an instance of group Gr, the best objective value obtained by the compared algorithm as well as our TPS algorithm. The values of all the possible mean improvement percentages are given in Table 8, where the columns indicate the benchmark groups (excluding groups CP and OP1 which are too easy for the competing algorithms), and the rows indicate pairs of competing algorithms (our TPS algorithm against each previously existing algorithm). The unavailable items are marked as '-', meaning that the compared algorithm did not report results on the corresponding benchmark group.

Table 8 shows that, compared to each reference algorithm (if applicable), TPS yields a positive mean improvement percentage (ranging from 0.02% to 31.79%) on each benchmark group, corresponding to various problem sizes, network densities and types of quadratic costs. Specifically, TPS is extremely competitive on the instances transformed from QAP (see more discussions in Section 4.2).

Furthermore, to check the statistical significance of the observed differences, we apply the Friedman test to compare TPS and each compared algorithm (in terms of the best found objective values) on each benchmark group. The p values of this test are given in Table 9 where the columns and rows indicate the same information as in Table 8. This test confirms that the differences are statistically significant (with a p value smaller than 5×10^{-2}) on almost every group of instances, with only three exceptions, i.e., TPS $\leftrightarrow$ ABC, TPS $\leftrightarrow$ QMST-TS on group SCA, and TPS $\leftrightarrow$ HSII on group RAND.

Table 11
Results corresponding to different directed perturbation operators.

Instance	TPS-V1 (only swap-edge)			TPS-V2 (combined)			TPS-V3 (only swap-vertex)		
	Best	*Average*	*T (s)*	*Best*	*Average*	*T (s)*	*Best*	*Average*	*T (s)*
SS-100-1	**88,730**	**88,790.3**	3600	88,780	88,865.0	3600	88,780	88,874.1	3600
SS-100-2	88,772	**88,806.9**	3600	**88,725**	88,833.5	3600	88,855	88,915.8	3600
SS-100-3	**88,619**	**88,619.0**	3600	**88,619**	88,656.1	3600	**88,619**	88,762.8	3600
SS-150-1	205,011	**205,069.3**	3600	**204,984**	205,258.5	3600	205,350	205,641.0	3600
SS-150-2	**204,873**	205,178.4	3600	204,991	**205,143.5**	3600	205,337	205,736.9	3600
SS-150-3	**204,770**	**205,034.5**	3600	204,992	205,233.4	3600	205,429	205,700.7	3600
SS-200-1	**370,506**	**370,941.1**	3600	370,587	371,264.7	3600	371,532	372,135.9	3600
SS-200-2	370,827	**371,059.6**	3600	**370,825**	371,178.3	3600	371,727	372,080.9	3600
SS-200-3	**370,582**	**371,159.2**	3600	370,990	371,472.9	3600	371,840	372,278.4	3600
SS-250-1	**586,812**	**587,174.0**	3600	586,968	587,586.5	3600	588,403	589,189.4	3600
SS-250-2	**586,689**	**587,157.5**	3600	586,980	587,445.2	3600	588,743	589,361.5	3600
SS-250-3	586,717	**587,004.1**	3600	**586,269**	587,168.3	3600	588,805	589,377.7	3600
RAND-150-1	**192,369**	**192,754.6**	4000	192,560	192,909.9	4000	192,827	193,388.0	4000
RAND-150-2	**192,579**	**192,856.1**	4000	192,587	192,931.8	4000	193,297	193,508.4	4000
RAND-150-3	**192,046**	**192,583.6**	4000	192,496	192,876.0	4000	193,067	193,420.6	4000
RAND-200-1	**350,321**	**350,901.1**	12,000	350,676	351,147.8	12,000	351,703	352,196.6	12,000
RAND-200-2	**350,658**	**350,880.5**	12,000	350,711	351,184.9	12,000	351,828	352,114.6	12,000
RAND-200-3	350,601	**350,980.1**	12,000	**350,506**	351,152.2	12,000	351,680	352,305.1	12,000
RAND-250-1	**557,278**	**557,823.4**	20,000	557,626	558,193.0	20,000	558,656	559,514.1	20,000
RAND-250-2	**556,604**	**557,822.8**	20,000	557,789	558,200.4	20,000	559,387	560,003.1	20,000
RAND-250-3	**557,060**	**557,762.0**	20,000	557,399	558,045.7	20,000	558,690	559,650.3	20,000
SOAK-150-1	**206,721**	206,900.7	4000	**206,721**	**206,890.4**	4000	206,895	207,030.9	4000
SOAK-150-2	**206,761**	**206,894.5**	4000	206,922	207,176.1	4000	206,922	207,425.3	4000
SOAK-150-3	**206,759**	**206,839.9**	4000	206,785	206,889.4	4000	206,898	207,164.0	4000
SOAK-200-1	**369,851**	**370,220.1**	12,000	369,990	370,377.8	12,000	370,741	371,148.6	12,000
SOAK-200-2	**369,803**	**369,932.6**	12,000	369,822	370,263.8	12,000	370,590	371,117.7	12,000
SOAK-200-3	**369,794**	**370,010.2**	12,000	370,089	370,182.8	12,000	370,403	370,709.1	12,000
SOAK-250-1	**581,671**	582,364.4	20,000	581,959	**582,357.1**	20,000	582,935	583,791.0	20,000
SOAK-250-2	**581,492**	**581,903.0**	20,000	581,671	582,154.7	20,000	581,884	583,431.3	20,000
SOAK-250-3	**581,573**	582,323.8	20,000	581,596	**582,290.2**	20,000	582,641	583,270.8	20,000
chr20a	**2192**	2231.2	3600	**2192**	**2192.0**	3600	**2192**	**2192.0**	3600
chr20b	2334	2373.2	3600	**2298**	**2298.0**	3600	**2298**	2302.6	3600
chr20c	14,214	14,929.8	3600	**14,142**	**14,142.0**	3600	**14,142**	**14,142.0**	3600
chr22a	6384	6438.2	3600	**6156**	**6156.0**	3600	**6156**	**6156.0**	3600
chr22b	6364	6465.0	3600	**6194**	**6198.4**	3600	**6194**	6201.2	3600
chr25a	3970	4248.8	3600	**3796**	**3796.0**	3600	**3796**	**3796.0**	3600
nug20	2602	2637.0	3600	**2570**	**2570.0**	3600	**2570**	**2570.0**	3600
nug21	2550	2574.8	3600	**2438**	**2438.0**	3600	**2438**	**2438.0**	3600
nug22	3714	3745.8	3600	**3596**	**3596.0**	3600	**3596**	**3596.0**	3600
nug24	3686	3721.6	3600	**3488**	**3488.0**	3600	**3488**	**3488.0**	3600
nug25	3872	3961.6	3600	**3744**	**3744.0**	3600	**3744**	**3744.0**	3600
nug27	5446	5488.4	3600	**5234**	**5234.0**	3600	**5234**	**5234.0**	3600
nug28	5358	5420.8	3600	**5166**	**5166.0**	3600	**5166**	**5166.0**	3600
nug30	6470	6543.4	3600	**6124**	**6124.0**	3600	**6124**	**6124.0**	3600

These results demonstrate that TPS competes very favorably with the existing QMSTP approaches on different types of benchmark instances.

4. Analysis of search components

4.1. Impact of the pre-estimation criterion

The TPS algorithm employs a pre-estimation criterion to discard useless swap-edge moves while applying the swap-edge operator (Section 2.5). To highlight the importance of this fast examination technique, we realize the following experiment. While solving each group of instances, we record the total number of all the possible edges $e \in E \backslash X$, associated with the number of the useless edges discarded by the pre-estimation criterion, as detailed in Table 10. Table 10 reveals that for benchmarks CP, OP1, SCA, SS, RAND, SOAK, QAP-QMSTP, the pre-estimation criterion can respectively identify and discard 88.9%, 71.8%, 93.0%, 97.1%, 97.3%, 97.2%, 97.2% useless edges among all the possible edges.

Furthermore, we study the influence of the pre-estimation criterion in terms of the problem size. For this purpose, we classify all the 659 instances into two categories, i.e., small-sized instances with $|V| \leq 100$ (including all the instances of groups CP, OP1, SCA, QAP-QMSTP and nine instances of group SS), and large-sized instances with $|V| > 100$ (including all the instances of groups RAND, SOAK and nine instances of group SS). Similarly, we record the ratio between the number of useless edges discarded by the pre-estimation criterion and the total number of all the possible edges $e \in E \backslash X$. The reduction ratio is 89.3%, 97.3% on the small and large-sized instances respectively.

This experiment confirms the relevance of the pre-estimation criterion to the proposed algorithm for solving various instances of different types and sizes.

4.2. Impact of the directed perturbation operators

Our TPS algorithm relies on two tabu-based directed perturbation operators (swap-edge and swap-vertex). To analyze the impact of these directed perturbation operators, we implement as follows three variants of TPS by varying the directed perturbation

Table 12
Results corresponding to different diversified perturbation operators.

Instance	Standard TPS			TPS-V4 (random restart)			TPS-V5 (directed perturb instead)		
	Best	*Average*	*T (s)*	*Best*	*Average*	*T (s)*	*Best*	*Average*	*T (s)*
SS-100-1	**88,730**	**88,790.3**	3600	88,829	88,919.4	3600	88,780	88,819.5	3600
SS-100-2	**88,772**	**88,806.9**	3600	88,862	88,947.6	3600	88,773	88,829.9	3600
SS-100-3	**88,619**	**88,619.0**	3600	88,671	88,762.2	3600	**88,619**	88,664.8	3600
SS-150-1	205,011	**205,069.3**	3600	205,436	205,632.3	3600	**204,936**	205,266.5	3600
SS-150-2	**204,873**	205,178.4	3600	205,492	205,629.6	3600	204,911	**205,134.1**	3600
SS-150-3	**204,770**	**205,034.5**	3600	205,232	205,556.7	3600	204,860	205,072.3	3600
SS-200-1	**370,506**	**370,941.1**	3600	370,938	371,760.9	3600	370,827	371,103.8	3600
SS-200-2	**370,827**	**371,059.6**	3600	371,283	371,731.6	3600	370,937	371,169.5	3600
SS-200-3	**370,582**	**371,159.2**	3600	371,210	371,779.6	3600	370,892	371,192.2	3600
SS-250-1	586,812	587,174.0	3600	587,332	587,873.1	3600	**586,302**	**586,831.1**	3600
SS-250-2	586,689	587,157.5	3600	587,263	587,692.4	3600	**586,411**	**586,921.3**	3600
SS-250-3	586,717	587,004.1	3600	587,257	587,651.8	3600	**586,298**	**586,604.9**	3600
RAND-150-1	**192,369**	**192,754.6**	4000	193,287	193,502.5	4000	192,721	192,866.9	4000
RAND-150-2	**192,579**	**192,856.1**	4000	193,204	193,462.4	4000	192,603	192,891.6	4000
RAND-150-3	**192,046**	**192,583.6**	4000	193,026	193,470.8	4000	192,495	192,678.3	4000
RAND-200-1	**350,321**	**350,901.1**	12,000	351,575	352,126.3	12,000	350,610	350,954.7	12,000
RAND-200-2	350,658	350,880.5	12,000	351,352	351,934.2	12,000	**350,328**	**350,754.4**	12,000
RAND-200-3	350,601	350,980.1	12,000	351,449	352,057.9	12,000	**350,462**	**350,780.5**	12,000
RAND-250-1	**557,278**	**557,823.4**	20,000	559,206	559,607.1	20,000	557,551	557,894.9	20,000
RAND-250-2	**556,604**	557,822.8	20,000	558,400	559,512.1	20,000	557,122	**557,677.2**	20,000
RAND-250-3	557,060	**557,762.0**	20,000	559,029	559,454.8	20,000	**556,714**	557,858.1	20,000
SOAK-150-1	**206,721**	206,900.7	4000	**206,721**	207,064.1	4000	**206,721**	**206,784.7**	4000
SOAK-150-2	**206,761**	**206,894.5**	4000	207,076	207,356.2	4000	206,905	207,089.4	4000
SOAK-150-3	**206,759**	**206,839.9**	4000	206,914	207,087.2	4000	**206,759**	206,906.0	4000
SOAK-200-1	**369,851**	**370,220.1**	12,000	370,701	371,004.8	12,000	370,060	370,277.0	12,000
SOAK-200-2	369,803	**369,932.6**	12,000	370,056	370,766.4	12,000	**369,736**	370,086.0	12,000
SOAK-200-3	**369,794**	**370,010.2**	12,000	370,035	370,674.5	12,000	369,808	370,124.0	12,000
SOAK-250-1	581,671	582,364.4	20,000	582,829	583,544.8	20,000	**581,639**	**582,051.7**	20,000
SOAK-250-2	**581,492**	**581,903.0**	20,000	582,817	583,228.2	20,000	581,556	582,043.6	20,000
SOAK-250-3	**581,573**	582,323.8	20,000	583,196	58,3571.7	20,000	581,775	**582,158.7**	20,000
chr20a	**2192**	**2192.0**	3600	**2192**	**2192.0**	3600	**2192**	**2192.0**	3600
chr20b	**2298**	2302.6	3600	**2298**	2307.2	3600	**2298**	**2298.0**	3600
chr20c	**14,142**	**14,142.0**	3600	**14,142**	**14,142.0**	3600	**14,142**	**14,142.0**	3600
chr22a	**6156**	**6156.0**	3600	**6156**	**6156.0**	3600	**6156**	**6156.0**	3600
chr22b	**6194**	6201.2	3600	**6194**	**6194.0**	3600	**6194**	**6194.0**	3600
chr25a	**3796**	**3796.0**	3600	**3796**	**3796.0**	3600	**3796**	**3796.0**	3600
nug20	**2570**	**2570.0**	3600	**2570**	**2570.0**	3600	**2570**	**2570.0**	3600
nug21	**2438**	**2438.0**	3600	**2438**	**2438.0**	3600	**2438**	**2438.0**	3600
nug22	**3596**	**3596.0**	3600	**3596**	**3596.0**	3600	**3596**	**3596.0**	3600
nug24	**3488**	**3488.0**	3600	**3488**	**3488.0**	3600	**3488**	**3488.0**	3600
nug25	**3744**	**3744.0**	3600	**3744**	**3744.0**	3600	**3744**	**3744.0**	3600
nug27	**5234**	**5234.0**	3600	**5234**	**5234.0**	3600	**5234**	**5234.0**	3600
nug28	**5166**	**5166.0**	3600	**5166**	**5166.0**	3600	**5166**	**5166.0**	3600
nug30	**6124**	**6124.0**	3600	**6124**	**6124.0**	3600	**6124**	**6124.0**	3600

operator. Respectively, TPS-V1 always applies the swap-edge directed perturbation operator, and TPS-V3 always applies the swap-vertex directed perturbation operator, while TPS-V2 applies each operator with probability 50%. All the remaining ingredients and parameters keep in accordance with the standard TPS (described in Sections 2 and 3.2).

For this study, we select a subset of 44 most challenging instances (among all the 659 instances) as sample instances, including 30 conventional instances (i.e., the 12 instances with $|V| \geq 100$ of group SS, together with all the 18 instances of groups RAND and SOAK) and 14 special instances transformed from QAP (i.e., the 14 instances with $|V| \geq 40$ of group QAP-QMSTP). For each instance, we independently run each variant 10 times, each run continues until a given cutoff time is elapsed. Respectively, for groups RAND and SOAK, we use the same cutoff time as in Section 3.7, and for groups SS and QAP-QMSTP, each run continues for six minutes, thus a total time of one hour is allowed for 10 independent runs. The obtained results are provided in Table 11, including the best and average objective values of the 10 independent runs, and the accumulated CPU times (in seconds). The result in **bold** indicates that the corresponding variant performs the best on this instance, in terms of best or average objective value.

We use the Friedman test to examine the statistical differences between different variants. Respectively, on the 30 conventional instances, the Friedman test detects significant differences in terms of both best and average objective values (with p values of 2.04×10^{-11} and 7.13×10^{-13} respectively). On the 14 QAP-QMSTP instances, the Friedman test also detects significant differences in terms of both best and average objective values (with p values of 2.84×10^{-5} and 7.43×10^{-6} respectively). These observations indicate the importance of the directed perturbation to the performance of TPS on various types of instances.

For the 30 conventional instances, Table 11 indicates that TPS-V1 clearly dominates TPS-V3 on almost all the test instances, in terms of both best and average objective values. Furthermore, we find TPS-V1 is also superior to TPS-V2. In terms of best objective values, TPS-V1 yields 23 better and 2 equal results compared to TPS-V2. In terms of average objective values, TPS-V1 dominates TPS-V2 on 26 out of the 30 test instances. These comparisons imply that the swap-edge directed perturbation operator is suitable for tackling the conventional instances. On the other hand, for the 14 QAP-QMSTP instances, one finds that TPS-V3 performs similar to TPS-V2 (without significant difference detected by

Friedman test), while they both dominate TPS-V1 on almost all the test instances (in terms of both best and average objective values), implying that the swap-vertex directed perturbation is very useful for solving these particular instances.

The excellent performance of TPS with the swap-vertex operator on the QAP-QMSTP instances (transformed from QAP) could be explained as follows. In fact, QAP is to uniquely assign k source vertices to k destination vertices of minimal cost (including linear and quadratic terms) (Christofides and Benavent, 1989). According to the transformation rules described in Öncan and Punnen (2010), any transformed QAP-QMSTP instance has the following features: (1) the input graph also consists of k source vertices and k destination vertices. (2) In the optimal solution (a tree), every source vertex must be a leaf vertex, being connected to a unique destination vertex (otherwise the total cost would become unreasonably high). (3) The objective value of a feasible solution tree completely depends on its edges that connect source vertices and destination vertices (call these edges as bridge edges), regardless of other edges between destination vertices. According to these features, given a feasible solution tree $T=(V,X)$ of reasonable quality (with each source vertex being a leaf vertex connected to a unique destination vertex), we can find that swapping any two bridge edges $e \in E \backslash X$ and $f \in X$ would always lead to an unfeasible solution (not a tree) or a solution tree of unreasonably high objective value (with two source vertices connected to the same destination vertex), thus disqualifying the conventional swap-edge operator. By contrast, by applying the swap-vertex operator which is equivalent to swapping two pairs of bridge edges, the search has more opportunity to reach feasible solutions of reasonable quality, thus being able to jump out of the current local optimum.

These observations also explain why the standard TPS algorithm applies respectively the swap-edge directed perturbation and the swap-vertex perturbation to solve the conventional QMSTP instances and the special QAP instances.

4.3. Impact of the diversified perturbation operator

Now we analyze the impact of the diversified perturbation operator. For this purpose, besides the standard TPS algorithm described in Section 2, we implement for comparisons two TPS variants by varying the diversified perturbation operator. Respectively, variant TPS-V4 adopts a random restarting strategy which uses the randomized initialization procedure of Section 2.3 instead of the original diversified perturbation operator of Section 2.7, and variant TPS-V5 uses the directed perturbation of Section 2.6 instead. All the other ingredients and parameters keep in accordance with the standard TPS (with default parameters of Section 3.2). Again, we use the 44 most challenging instances to evaluate the performances of these two variants as well as the standard TPS algorithm. For each instance, we run each variant 10 times, using the same cutoff time as stop condition as before (Section 4.2). The results are detailed in Table 12, with the same meaning of each column as in Table 11.

We use the Friedman test to examine if there exist significant statistical differences. Respectively, on the 30 conventional instances, the Friedman test reveals a p value of 3.41×10^{-10} in terms of best objective values, and a p value of 2.46×10^{-10} in terms of average values, indicating the importance of the diversified perturbation operator on these instances. By contrast, on the 14 QAP-QMSTP instances, the Friedman test does not detect a significant difference in terms of both best objective values (all the three compared variants could reach the optimal solutions of these

Table A1
Results obtained by varying parameter l_{in} on the 30 conventional instances.

Instances	TPS with varied values (intervals) of parameter l_{in}							
	[1,3]	[3,10]	[10,100]	[0.1n, 0.2n]	[0.2n, 0.3n]	[0.3n, 0.4n]	[0.4n, 0.5n]	[0.5n, n]
SS-100-1	88,808.9	88,809.9	88,817.3	88,800.1	**88,786.9**	88,804.2	888,00.6	88,800.5
SS-100-2	88,824.1	88,830.8	88,814.4	88,851.3	88,818.5	88,815.5	**88,810.8**	88,839.6
SS-100-3	88,625.3	88,629.7	**88,623.4**	88,671.5	88,631.0	88,641.3	88,629.8	88,660.1
SS-150-1	205,196.5	**205,037.9**	205,141.8	205,113.8	205,135.9	205,171.8	205,095.5	205,220.8
SS-150-2	205,306.6	**205,077.1**	205,226.1	205,122.5	205,154.6	205,107.8	205,179.3	205,174.9
SS-150-3	205,178.0	**204,989.4**	205,090.3	205,069.6	205,072.1	205,136.0	205,021.7	205,180.4
SS-200-1	371,036.6	371,010.6	371,205.3	370,930.7	**370,748.8**	370,952.8	370,968.6	371,175.8
SS-200-2	371,040.2	371,165.3	371,216.7	371,150.3	371,304.0	371,126.3	**370,972.7**	371,242.3
SS-200-3	371,209.3	371,242.9	371,271.8	371,244.2	371,340.3	371,134.3	**371,087.2**	371,469.4
SS-250-1	587,177.0	587,050.9	587,227.2	**586,820.5**	587,165.4	587,104.5	587,356.0	587,438.2
SS-250-2	**586,690.1**	587,041.8	586,820.2	586,947.2	586,717.9	586,871.2	587,035.5	587,095.4
SS-250-3	586,711.6	586,867.6	586,995.5	586,849.7	**586,629.6**	586,633.7	586,873.2	587,089.8
RAND-150-1	**192,705.9**	192,770.7	192,766.1	192,839.6	192,745.1	192,801.0	192,927.1	192,830.2
RAND-150-2	192,842.1	192,964.3	192,861.0	192,933.9	192,946.7	192,978.7	192,960.4	**192,779.8**
RAND-150-3	**192,698.9**	192,856.8	192,783.2	192,757.6	192,811.5	192,796.7	192,870.8	192,783.0
RAND-200-1	**351,546.4**	351,762.5	351,594.6	351,561.7	351,622.4	351,588.0	352,004.7	351,884.5
RAND-200-2	351,457.4	351,918.2	351,524.9	351,643.4	**351,282.8**	351,689.8	351,801.5	351,836.7
RAND-200-3	351,658.7	351,695.6	351,491.9	**351,238.4**	351,555.0	351,639.5	351,937.2	351,875.1
RAND-250-1	**558,750.8**	558,920.7	559,327.7	558,778.0	559,220.4	558,873.9	559,253.8	559,313.4
RAND-250-2	559,296.6	558,854.4	559,148.9	**558,784.3**	559,222.4	559,004.5	559,149.2	559,286.7
RAND-250-3	**558,678.6**	558,714.0	558,689.5	558,876.5	559,431.2	559,366.5	559,266.4	559,697.3
SOAK-150-1	206,835.0	**206,830.0**	206,938.0	206,880.2	206,917.9	206,888.7	206,912.6	206,965.9
SOAK-150-2	207,004.8	207,003.7	207,030.9	**206,958.8**	207,083.6	207,076.4	206,978.5	207,029.1
SOAK-150-3	**206,877.2**	206,888.4	206,904.9	206,896.5	206,944.4	206,901.5	206,910.2	206,966.8
SOAK-200-1	**370,572.6**	370,679.9	370,592.9	370,670.0	370,752.4	370,835.3	370,679.8	371,032.7
SOAK-200-2	**370,488.9**	370,680.2	370,658.0	370,529.8	370,545.8	370,667.1	370,589.9	370,955.6
SOAK-200-3	370,799.4	370,474.3	370,413.5	**370,076.2**	370,479.1	370,519.2	370,615.4	370,613.2
SOAK-250-1	583,541.8	583,073.7	583,152.0	583,057.5	**582,963.5**	583,201.5	583,483.9	583,895.6
SOAK-250-2	583,119.7	582,798.1	**582,740.6**	583,304.2	583,209.8	583,049.8	583,244.6	583,423.2
SOAK-250-3	583,315.1	**582,916.5**	583,239.0	583,226.0	583,703.4	583,708.4	583,157.6	583,557.7
ν_i	**9**	5	2	5	5	0	3	1
λ_i	**3.67**	4.03	4.43	**3.33**	4.40	4.53	4.87	6.73

Table A2
Results obtained by varying parameter l_{out} on the 30 conventional instances.

Instances	TPS with varied values (intervals) of parameter l_{out}							
	[1,3]	[3,10]	[10,100]	[0.1n, 0.2n]	[0.2n, 0.3n]	[0.3n, 0.4n]	[0.4n, 0.5n]	[0.5n, n]
SS-100-1	88,833.6	88,828.2	88,847.4	88,855.8	**88,770.2**	88,818.4	88,810.9	88,787.8
SS-100-2	88,848.0	88,871.8	88,865.9	88,831.3	88,818.2	88,834.0	**88,812.9**	88,840.1
SS-100-3	88,654.6	88,695.3	88,742.1	88,697.3	88,644.9	88,654.7	**88,643.6**	88,680.1
SS-150-1	205,188.3	205,237.5	205,714.0	205,123.2	205,147.7	**205,060.0**	205,129.2	205,224.5
SS-150-2	205,249.6	205,289.7	205,726.3	205,166.4	205,126.4	**205,124.3**	205,142.3	205,229.4
SS-150-3	205,300.0	205,236.5	205,513.5	205,097.4	205,018.7	205,064.2	**204,997.1**	205,093.0
SS-200-1	371,135.4	371,181.9	371,880.3	370,865.4	371,214.7	**370,778.3**	370,989.9	371,210.7
SS-200-2	371,254.9	371,394.1	372,056.5	**370,894.1**	371,174.9	370,903.7	371,101.4	371,276.3
SS-200-3	371,535.3	371,310.9	372,048.0	370,847.7	371,115.0	**370,823.5**	371,423.7	371,390.7
SS-250-1	587,993.4	587,188.1	588,617.0	**586,477.4**	587,142.6	586,609.2	587,093.8	587,274.5
SS-250-2	587,525.4	586,587.8	588,199.9	586,624.3	**586,559.9**	586,877.6	587,152.4	586,784.7
SS-250-3	587,455.5	586,739.5	588,323.1	**58,6357.3**	586,631.1	586,610.2	587,131.9	586,802.4
RAND-150-1	192,879.7	192,843.5	193,364.0	192,850.4	192,862.1	192,949.8	192,883.1	**192,781.6**
RAND-150-2	193,222.1	192,948.8	193,456.8	192,872.9	**192,796.1**	192,803.4	192,953.1	192,971.6
RAND-150-3	192,895.7	192,873.1	193,404.8	192,827.6	**192,681.7**	192,718.3	192,933.4	192,997.8
RAND-200-1	352,226.4	351,540.1	353,068.5	351,475.1	351,499.2	**351,312.0**	351,635.1	352,184.6
RAND-200-2	352,177.3	351,413.9	352,832.2	351,380.9	351,398.4	**351,327.6**	351,699.0	351,943.5
RAND-200-3	352,034.9	351,433.9	352,856.9	351,310.6	351,380.7	**351,277.2**	351,506.7	351,778.1
RAND-250-1	559,519.1	558,925.9	561,506.1	**558,415.2**	558,756.3	558,537.4	559,115.9	559,378.6
RAND-250-2	560,533.0	559,190.7	561,395.0	**558,398.5**	558,818.0	559,087.1	559,183.5	559,246.1
RAND-250-3	560,002.2	558,947.6	561,363.8	**5582,78.9**	558,869.4	558,543.3	559,034.8	559,162.4
SOAK-150-1	206,993.6	206,989.0	207,179.3	**206,878.3**	206,911.5	206,907.7	206,918.4	206,991.4
SOAK-150-2	207,284.8	207,160.4	207,404.7	207,118.9	207,147.2	**206,911.8**	206,943.2	207,198.5
SOAK-150-3	206,998.3	206,950.4	207,168.2	206,861.7	206,855.8	206,841.5	**206,841.0**	206,928.5
SOAK-200-1	371,093.3	370,850.4	371,872.6	**370,505.7**	370,743.7	370,572.9	370,589.0	371,060.1
SOAK-200-2	371,123.4	370,831.0	371,700.9	370,580.5	370,594.2	**370,345.1**	370,608.1	370,667.7
SOAK-200-3	370,758.5	370,538.0	371,414.9	370,475.8	370,516.5	370,456.3	370,529.0	**370,412.4**
SOAK-250-1	583,595.1	583,239.0	585,526.4	582,996.2	583,127.4	**582,756.5**	583,228.8	583,656.8
SOAK-250-2	583,382.6	583,298.4	585,262.2	**582,891.7**	582,959.2	582,931.5	582,917.8	583,314.6
SOAK-250-3	58,3694.4	58,3726.0	585,592.5	583,234.1	583,016.2	**582,930.4**	583 ,189.3	58,3682.1
ν_i	0	0	0	9	4	**11**	4	2
λ_i	6.33	5.03	7.93	2.60	2.93	**2.20**	3.77	5.20

Table A3
Results obtained by varying parameter l_{swap} on the 14 QAP-QMSTP instances.

Instances	TPS with varied values (intervals) of parameter l_{swap}							
	[1,10]	[10,100]	[100,500]	[n, 2n]	[2n, 3n]	[3n, 4n]	[4n, 5n]	[5n, 10n]
chr20a	2192.4	**2192.0**	**2192.0**	**2192.0**	**2192.0**	**2192.0**	2192.4	**2192.0**
chr20b	2311.8	**2298.0**	2323.4	**2298.0**	2312.0	2314.4	2322.8	2307.2
chr20c	**14,142.0**	**14,142.0**	**14,142.0**	**14,142.0**	**14,142.0**	**14,142.0**	**14,142.0**	**14,142.0**
chr22a	**6156.0**	**6156.0**	**6156.0**	**6156.0**	**6156.0**	**6156.0**	**6156.0**	**6156.0**
chr22b	6197.6	**6194.0**	6232.8	**6194.0**	6210.8	6214.4	6219.0	6208.6
chr25a	3918.4	**3796.0**	3817.0	**3796.0**	3817.0	3819.6	3820.6	3843.2
nug20	**2570.0**	**2570.0**	**2570.0**	**2570.0**	**2570.0**	**2570.0**	**2570.0**	**2570.0**
nug21	**2438.0**	**2438.0**	**2438.0**	**2438.0**	**2438.0**	**2438.0**	**2438.0**	**2438.0**
nug22	**3596.0**	**3596.0**	**3596.0**	**3596.0**	**3596.0**	**3596.0**	**3596.0**	**3596.0**
nug24	**3488.0**	**3488.0**	**3488.0**	**3488.0**	**3488.0**	**3488.0**	**3488.0**	**3488.0**
nug25	**3744.0**	**3744.0**	**3744.0**	**3744.0**	**3744.0**	**3744.0**	**3744.0**	**3744.0**
nug27	**5234.0**	**5234.0**	**5234.0**	**5234.0**	**5234.0**	**5234.0**	**5234.0**	**5234.0**
nug28	**5166.0**	**5166.0**	**5166.0**	**5166.0**	**5166.0**	**5166.0**	**5166.0**	**5166.0**
nug30	**6124.0**	**6124.0**	**6124.0**	**6124.0**	**6124.0**	**6124.0**	**6124.0**	6124.4
ν_i	10	**14**	11	**14**	11	11	10	11
λ_i	2.29	**1.00**	2.14	**1.00**	1.71	2.00	2.64	2.29

14 instances, leading to a p value of 1) and average objective values (corresponding to a p value of 3.75×10^{-1}).

Furthermore, concerning the 30 conventional instances, Table 12 shows that the standard TPS algorithm clearly dominates TPS-V4 on almost all the test instances, in terms of both best and average objective values. The standard TPS algorithm also competes favorably with TPS-V5. In terms of best objective values, the standard TPS algorithm yields respectively 18 better, 3 equal and 9 worse results compared to TPS-V5. In terms of average objective values, the standard TPS algorithm dominates TPS-V5 on 20 out of the 30 test instances and performs worse on the remaining 10 instances. These comparisons provide some insights about the importance of the diversified perturbation operator to the performance of TPS on challenging conventional instances.

5. Conclusion

We have originally proposed a three-phase heuristic approach named TPS for the quadratic minimum spanning tree problem (QMSTP), which could be used to model a number of network design problems. The proposed approach consists of a descent-

Table A4
Results obtained by varying parameter L_{dir} on all the 44 challenging instances.

Instances	TPS with varied values (intervals) of parameter L_{dir}							
	[0.1n, 0.5n]	[0.1n, n]	[0.5n, n]	[0.5n, 2n]	[n, 2n]	[n, 5n]	[2n, 5n]	[2n, 10n]
SS-100-1	88,801.8	88,807.5	88,798.8	88,796.2	88,808.7	88,816.9	**88,788.6**	88,826.9
SS-100-2	**88,807.0**	88,833.5	88,837.8	88,828.6	88,834.0	88,829.4	88,842.3	88,865.3
SS-100-3	88,655.7	88,649.3	88,630.1	88,650.1	88,628.6	88,629.2	**88,628.0**	88,635.1
SS-150-1	205,156.8	205,140.3	205,146.7	**205,037.8**	205,227.9	205,154.8	205,131.2	205,247.9
SS-150-2	205,165.2	205,243.5	205,227.7	**205,071.3**	205,177.2	205,136.9	205,181.0	205,157.9
SS-150-3	205,122.5	205,079.1	205,087.5	**204,985.1**	205,106.3	205,097.4	205,152.5	205,166.2
SS-200-1	370,928.4	371,009.5	370,917.6	371,003.1	**370,812.9**	371,014.2	371,137.5	371,145.2
SS-200-2	370,945.5	370,838.6	**370,832.8**	370,999.6	370,971.0	371,191.3	371,306.0	371,414.9
SS-200-3	**370,845.8**	371,220.0	371,169.7	371, 044.2	371,200.1	371,329.1	371,236.0	371,737.0
SS-250-1	**586,878.5**	586,889.3	587,001.1	587,054.4	587,168.1	587,315.0	587,170.4	587,510.2
SS-250-2	586,820.7	586,792.1	586,731.9	**586,705.8**	586,867.4	587,089.2	587,332.8	587,554.5
SS-250-3	**586,272.9**	586,714.4	586,562.8	586,790.0	586,619.3	586,904.3	586,976.9	587,376.6
RAND-150-1	192,713.6	192,789.8	192,784.0	**192,676.3**	192,679.7	192,900.4	192,921.7	193,132.5
RAND-150-2	**192,759.3**	192,835.7	192,887.7	192,869.1	192,982.0	192,923.7	193,004.8	193,033.2
RAND-150-3	192,784.4	**192,676.2**	192,683.4	192,748.2	192,838.9	192,987.6	192,999.8	192,923.3
RAND-200-1	**351,245.3**	351,651.6	351,486.3	351,399.6	351,637.2	352,031.6	351,778.3	351,958.4
RAND-200-2	351,332.4	351,478.7	351,547.3	**351,310.9**	351,848.9	351,824.6	351,799.2	351,933.1
RAND-200-3	351,357.3	351,783.5	351,371.4	351,516.3	**351,281.3**	351,525.1	351,852.0	351,914.0
RAND-250-1	558,795.9	559,053.2	**558,602.2**	558,818.1	558,799.9	559,334.2	559,450.4	559,380.3
RAND-250-2	558,620.9	558,854.2	558,931.8	**558,552.1**	558,947.8	559,194.0	559,427.0	559,551.8
RAND-250-3	558,757.1	558,892.3	558,884.8	**55,8639.3**	559,085.4	559,243.4	558,927.8	559,469.0
SOAK-150-1	206,876.9	206,869.0	**206,811.0**	206,893.3	206,873.3	206,919.8	206,967.4	206,970.1
SOAK-150-2	207,060.7	206,958.0	207,084.7	206,905.4	207,070.5	**206,855.7**	206,958.3	206,933.0
SOAK-150-3	206,866.2	**206,845.1**	206,855.9	2068,48.4	206,922.4	206,931.4	206,913.5	206,949.1
SOAK-200-1	370,560.0	370,647.3	370,570.1	**370,471.2**	370,697.5	370,753.3	370,986.7	370,861.5
SOAK-200-2	370,481.6	370,423.2	**370,359.9**	370,528.7	370,538.2	370,577.6	370,613.5	370,747.1
SOAK-200-3	370,293.0	370,436.1	370,329.0	**370,211.8**	370,322.2	370,463.6	370,536.1	370,816.1
SOAK-250-1	582,844.9	583,013.1	582,866.8	**582,778.0**	583,009.8	583,351.6	583,447.5	583,671.5
SOAK-250-2	582,884.7	582,978.5	**582,811.8**	582,861.5	582,901.2	58,3203.0	582,969.3	583,516.8
SOAK-250-3	583,115.3	582,970.6	583,002.5	**582,861.5**	582,887.9	583,357.2	583,180.3	58,3673.0
chr20a	**2192.0**	**2192.0**	**2192.0**	**2192.0**	**2192.0**	**2192.0**	**2192.0**	**2192.0**
chr20b	**2298.0**	**2298.0**	2311.8	**2298.0**	**2298.0**	**2298.0**	**2298.0**	**2298.0**
chr20c	**14,142.0**	**14,142.0**	**14,142.0**	**14,142.0**	**14,142.0**	**14,142.0**	**14,142.0**	**14,142.0**
chr22a	**6156.0**	**6156.0**	**6156.0**	**6156.0**	**6156.0**	**6156.0**	**6156.0**	**6156.0**
chr22b	6197.6	6201.2	6201.2	**6194.0**	6197.6	**6194.0**	6201.2	6198.4
chr25a	**3796.0**	3803.0	**3796.0**	**3796.0**	**3796.0**	**3796.0**	**3796.0**	**3796.0**
nug20	**2570.0**	**2570.0**	**2570.0**	**2570.0**	**2570.0**	**2570.0**	**2570.0**	**2570.0**
nug21	**2438.0**	**2438.0**	**2438.0**	**2438.0**	**2438.0**	**2438.0**	**2438.0**	**2438.0**
nug22	**3596.0**	**3596.0**	**3596.0**	**3596.0**	**3596.0**	**3596.0**	**3596.0**	**3596.0**
nug24	**3488.0**	**3488.0**	**3488.0**	**3488.0**	**3488.0**	**3488.0**	**3488.0**	**3488.0**
nug25	**3744.0**	**3744.0**	**3744.0**	**3744.0**	**3744.0**	**3744.0**	**3744.0**	**3744.0**
nug27	**5234.0**	**5234.0**	**5234.0**	**5234.0**	**5234.0**	**5234.0**	**5234.0**	**5234.0**
nug28	**5166.0**	**5166.0**	**5166.0**	**5166.0**	**5166.0**	**5166.0**	**5166.0**	**5166.0**
nug30	**6124.0**	**6124.0**	**6124.0**	**6124.0**	**6124.0**	**6124.0**	**6124.0**	**6124.0**
ν_i	19	14	17	**26**	15	15	15	13
λ_i	2.45	3.20	2.77	**2.00**	3.32	4.18	4.55	5.45

based neighborhood search phase for local optimization, a local optima exploring phase for intensive search in a given regional search space, and a diversified perturbation phase for jumping out of the current regional search space. Particularly, TPS integrates a novel pre-estimation criterion to avoid useless computations, a new swap-vertex move operator as well as its application for perturbations which prove to be particularly useful for solving the special instances transformed from QAP.

Extensive experimental comparisons on all the available 659 benchmarks show that TPS produces highly competitive results with respect to the state-of-the-art QMSTP approaches. For the 630 conventional instances, TPS succeeds in discovering, with a reasonable computational time, improved best known solutions (new upper bounds) for 33 challenging instances, while matching the best known results for the remaining instances. Finally, for the 29 special instances transformed from QAP, TPS can reach all the known optimal solutions within a short time, while the previous best QMSTP algorithm can only reach 12 optimal solutions within high computing times. TPS proves to be quite robust by providing excellent results with a standard default setting of its parameters. Additional analysis helps understand the influences of several key ingredients of the proposed algorithm, including the pre-estimation criterion, the perturbation operators (Section 4), as well as the main parameters (detailed in the Appendix).

Acknowledgments

We are grateful to the referees and the editors for their insightful comments and suggestions. We thank Dr. Roberto Cordone for kindly making their instances available and answering our questions. This work was partially supported by the Pays de la Loire Region (France) (RaDaPop, 2009–2013, N° 08513 and LigeRO, 2010–2014, 07-LigeRO), Jacques Hadamard Mathematical Foundation (PGMO, 2014–2015, N° 0024H), and a Post-doc fellowship from Angers Loire Metropole (France).

Table A5
Results obtained by varying parameter L_{div} on all the 44 challenging instances.

Instances	TPS with varied values (intervals) of parameter l_{div}							
	[0.1n, 0.5n]	[0.1n, n]	[0.5n, n]	[0.5n, 2n]	[n, 2n]	[n, 5n]	[2n, 5n]	[2n, 10n]
SS-100-1	88,911.7	88,873.1	88,870.6	88,814.9	88,853.0	**88,787.2**	88,798.4	88,793.4
SS-100-2	88,919.5	88,908.7	88,893.6	88,872.6	88,836.2	88,838.7	88,829.2	**88,810.0**
SS-100-3	88,804.2	88,717.7	88,713.8	88,732.1	88,713.1	88,635.7	88,640.5	**88,619.0**
SS-150-1	205,423.6	205,210.8	205,235.3	205,159.6	205,377.8	205,128.8	205,120.2	**205,080.1**
SS-150-2	205,355.8	205,291.3	205,325.9	205,222.2	205,158.6	205,168.5	205,165.3	**205,084.4**
SS-150-3	205,227.5	205,334.9	205,138.1	205,099.9	205,113.5	205,081.0	205,120.0	**204,957.5**
SS-200-1	371,401.8	371,087.6	371,049.3	371,058.6	371,077.5	371,007.2	**370,952.7**	371,058.6
SS-200-2	371,278.0	371,308.1	371,089.0	370,999.1	371,105.2	**370,982.6**	371,021.9	371,179.9
SS-200-3	371,386.7	371,233.2	371,274.6	**371,009.0**	371,017.3	371,165.8	371,086.1	371,220.7
SS-250-1	586,970.1	586,929.7	586,971.8	**586,835.3**	587,058.2	587,211.0	587,184.3	587,128.7
SS-250-2	587,027.8	586,721.0	587,094.7	**586,508.3**	587,088.0	586,913.8	587,037.6	587,071.2
SS-250-3	587,150.0	586,687.8	586,748.5	**586,530.9**	587,086.1	586,782.6	586,753.2	586,764.6
RAND-150-1	193,159.3	193,067.1	192,884.4	192,835.1	192,847.1	**192,734.8**	192,812.7	192,890.5
RAND-150-2	193,177.8	193,183.9	193,119.0	192,918.8	192,937.2	1929,14.3	**192,816.5**	192,994.7
RAND-150-3	193,021.5	192,958.3	192,930.4	192,872.8	192,733.5	192,728.7	**192,705.4**	192,995.1
RAND-200-1	351,631.5	351,553.9	351,711.8	351,695.4	351,553.8	351,555.1	**351,247.1**	351,724.7
RAND-200-2	351,763.7	351,715.7	351,631.7	351,723.5	351,627.7	**351,399.6**	351,552.1	351,831.4
RAND-200-3	351,613.4	351,706.6	351,606.3	351,655.4	351,511.9	**351,434.2**	351,595.5	351,760.9
RAND-250-1	559,418.3	559,187.9	558,842.1	**558,833.8**	558,856.7	558,927.4	558,930.4	559,381.5
RAND-250-2	559,280.0	559,285.0	559,100.4	**558,877.6**	558,930.2	559,314.8	559,189.1	559,377.7
RAND-250-3	558,997.6	558,981.3	**558,721.1**	558,761.8	559,104.2	559,137.9	559,096.0	559,364.4
SOAK-150-1	207,093.9	207,026.2	207,060.8	206,955.0	207,034.5	**206,855.7**	206,889.5	206,885.8
SOAK-150-2	207,401.5	207,261.5	207,234.8	207,098.2	207,092.3	207,003.9	206,966.8	**206,943.6**
SOAK-150-3	206,980.5	206,869.8	**206,845.5**	206,867.8	206,865.9	206,893.2	206,847.2	206,871.0
SOAK-200-1	370,901.0	370,753.8	370,875.1	370,559.3	370,754.7	370,837.9	370,657.7	**370,514.2**
SOAK-200-2	371,060.1	370,922.0	370,775.3	370,693.6	**370,421.7**	370,515.6	370,761.3	3704,87.8
SOAK-200-3	370,661.1	370,710.8	370,525.2	370,334.9	370,585.5	370,384.9	370,518.4	**370,117.6**
SOAK-250-1	583,780.1	**582,928.5**	583,481.8	583,162.7	583,330.8	583,164.3	583,416.6	583,107.5
SOAK-250-2	583,316.3	583,413.6	583,167.5	583,001.1	582,855.4	**582,855.1**	582,988.1	582,938.2
SOAK-250-3	583,525.5	583,852.2	583,409.2	583,737.2	583,538.0	583,255.8	**583,222.3**	583,344.2
chr20a	**2192.0**	**2192.0**	**2192.0**	**2192.0**	**2192.0**	**2192.0**	**2192.0**	**2192.0**
chr20b	**2298.0**	2302.6	**2298.0**	**2298.0**	**2298.0**	**2298.0**	**2298.0**	**2298.0**
chr20c	**14,142.0**	**14,142.0**	**14,142.0**	**14,142.0**	**14,142.0**	**14,142.0**	**14,142.0**	**14,142.0**
chr22a	**6156.0**	**6156.0**	**6156.0**	**6156.0**	**6156.0**	**6156.0**	**6156.0**	**6156.0**
chr22b	**6194.0**	**6194.0**	**6194.0**	**6194.0**	**6194.0**	**6194.0**	**6194.0**	**6194.0**
chr25a	**3796.0**	**3796.0**	**3796.0**	**3796.0**	**3796.0**	**3796.0**	**3796.0**	**3796.0**
nug20	**2570.0**	**2570.0**	**2570.0**	**2570.0**	**2570.0**	**2570.0**	**2570.0**	**2570.0**
nug21	**2438.0**	**2438.0**	**2438.0**	**2438.0**	**2438.0**	**2438.0**	**2438.0**	**2438.0**
nug22	**3596.0**	**3596.0**	**3596.0**	**3596.0**	**3596.0**	**3596.0**	**3596.0**	**3596.0**
nug24	**3488.0**	**3488.0**	**3488.0**	**3488.0**	**3488.0**	**3488.0**	**3488.0**	**3488.0**
nug25	**3744.0**	**3744.0**	**3744.0**	**3744.0**	**3744.0**	**3744.0**	**3744.0**	**3744.0**
nug27	**5234.0**	**5234.0**	**5234.0**	**5234.0**	**5234.0**	**5234.0**	**5234.0**	**5234.0**
nug28	**5166.0**	**5166.0**	**5166.0**	**5166.0**	**5166.0**	**5166.0**	**5166.0**	**5166.0**
nug30	**6124.0**	**6124.0**	**6124.0**	**6124.0**	**6124.0**	**6124.0**	**6124.0**	**6124.0**
ν_i	14	14	16	20	15	**21**	19	**22**
λ_i	5.07	4.34	3.75	2.73	3.14	**2.57**	**2.48**	3.16

Appendix A. Parameters tuning

We discuss how we tune the parameters of the TPS algorithm. Given that the benchmark instances have very different structures, it is extremely difficult to obtain a set of parameter values which yield uniformly the best result on every instance. Thus we only determine parameter values within reasonable ranges in order to obtain a globally good performance. As mentioned in Section 3.2, our preliminary experiments show that $l_{in} \in [1, 3]$, $l_{out} \in [0.3n, 0.4n]$, $l_{swap} \in [n, 2n]$, $L_{dir} \in [0.5n, 2n]$, $L_{div} \in [n, 5n]$ perform reasonably well on each group of instances (l_{in} and l_{out} are disabled for the instances transformed from QAP, and l_{swap} is disabled for the remaining conventional instances). In order to confirm these choices, for each parameter, we implement eight scenarios by varying the chosen parameter within a reasonable range (shown at the head of the tables of this Appendix), while fixing the other parameters in accordance with the default settings (i.e., the above parameter values).

We use the 44 challenging instances mentioned in Section 4.2 (including 30 conventional instances and 14 QAP-QMSTP instances) as sample instances to evaluate the performances of the compared scenarios. Corresponding to each parameter, we only use the instances on which the performance of TPS might be affected by this parameter. It means, we only use the 30 conventional instances to tune parameters l_{in} and l_{out}, and only use the 14 QAP-QMSTP instances to tune parameter l_{swap}, while using all these 44 instances to tune parameters L_{dir} and L_{div}.

Given the eight scenarios of each parameter, we independently run each scenario 10 times to solve each sample instance (each run is given six minutes, thus a total time of one hour is allowed for each scenario to solve each instance), and then record the average objective values of 10 independent runs in Tables A1–A5. Moreover, in order to evaluate the overall performance of each scenario for solving all the tested sample instances, we adopt two evaluation criteria as follows. First, for each scenario i, we count the number of sample instances (denoted by ν_i) on which scenario i performs the best among all the eight competing scenarios (corresponding to the same parameter).

The second criterion is the mean rank of each scenario for solving all the tested sample instances, which is precisely defined as follows: let $\lambda_{ij} = \chi + 1$ be the rank of scenario i for solving instance j, where χ denotes the number of scenarios (among the eight compared scenarios) which achieve a better average objective value than scenario i on instance j, then the mean rank of scenario i for solving all the sample instances is defined as $\lambda_i = \sum_{j=1}^{|SI|} \lambda_{ij}/|SI|$, where $|SI|$ denotes the number of sample instances selected to tune a specified parameter. Intuitively, a larger value of ν_i or a lower value of λ_i generally indicates a better overall performance of scenario i, with respect to other compared scenarios.

The detailed results (averaged objective values of 10 independent runs of each scenario for solving each instance) corresponding to each parameter are given in the following tables, in each table the rows and columns respectively indicate the instances and the scenarios, while the final two rows give the values of ν_i and λ_i of each scenario, in order to evaluate their overall performances. Prior to detailed comparisons, we first use the Friedman test to check the statistical differences between the competing scenarios.

Parameter l_{in} (Table A1): The Friedman test reveals a p value of 9.45×10^{-7}, indicating that l_{in} is somewhat sensitive to the performance of TPS. Furthermore, we observe the scenario with $l_{in} \in [1, 3]$ achieves the largest value of $\nu_i = 9$ and the second lowest value of $\lambda_i = 3.67$, indicating a good overall performance among all the eight scenarios.

Parameter l_{out} (Table A2): The Friedman test detects a significant difference (with a p value $< 2.21 \times 10^{-16}$) between these eight scenarios, meaning that parameter l_{out} should be carefully tuned. Furthermore, one observes that the scenario with $l_{out} \in [0.3n, 0.4n]$ achieves the largest value of $\nu_i = 11$ and the lowest value of $\lambda_i = 2.20$, indicating its excellent overall performance with respect to other compared scenarios.

Parameter l_{swap} (Table A3): For parameter l_{swap}, no significant difference is detected by the Friedman test (with a p value of 7.96×10^{-2}). Furthermore, there are two scenarios ($l_{swap} \in [10, 100]$ and $l_{swap} \in [n, 2n]$) both achieving the largest value of $\nu_i = 14$ and the lowest value of $\lambda_i = 1.00$. We choose $l_{swap} \in [n, 2n]$ as default setting in the final version of the TPS algorithm.

Parameter L_{dir} (Table A4): We decide to tune parameter L_{dir} with a uniform setting for solving each instance, but we examine the statistical differences respectively on the conventional instances and the instances transformed from QAP. The Friedman tests show that L_{dir} is sensitive on the 30 conventional instances (with a p value of 2.72×10^{-14}) and is quite robust on the 14 QAP-QMSTP instances (with a p value of 3.69×10^{-1}). Furthermore, among the eight compared scenarios, the scenario with $L_{dir} \in [0.5n, 2n]$ yields the largest value of $\nu_i = 26$ and the lowest value of $\lambda_i = 2.00$, thus being adopted in the final version of the TPS algorithm.

Parameter L_{div} (Table A5): Like parameter L_{dir}, we observe from the Friedman tests that parameter L_{div} is quite sensitive on the 30 conventional instances (with a p value of 4.05×10^{-8}) and is rather robust on the 14 QAP-QMSTP instances (with a p value of 4.82×10^{-1}). Additionally, we find the scenario with $L_{div} \in [n, 5n]$ competes favorably with other scenarios, corresponding to the second largest value of $\nu_i = 21$ and the second lowest value of $\lambda_i = 2.57$, thus being adopted in the final version of TPS algorithm.

References

Assad, A., Xu, W., 1992. The quadratic minimum spanning tree problem. Naval Res. Logist. 39, 399–417.

Bastos, M.P., Ribeiro, C.C., 2002. Reactive tabu search with path-relinking for the Steiner problem in graphs. In: Essays and Surveys in Metaheuristics, Operations Research/Computer Science Interfaces Series, vol. 15, pp. 39–58.

Benlic, U., Hao, J.K., 2013a. Breakout local search for the quadratic assignment problem. Appl. Math. Comput. 219 (9), 4800–4815.

Benlic, U., Hao, J.K., 2013b. Breakout local search for the max-cut problem. Eng. Appl. Artif. Intell. 26 (3), 1162–1173.

Buchheim, C., Klein, L., 2013. The spanning tree problem with one quadratic term. In: 12th Cologne-Twente Workshop (CTW2013) on Graphs and Combinatorial Optimization, Enschede, Netherlands, pp. 31–34.

Buchheim, C., Klein, L., 2014. Combinatorial optimization with one quadratic term: spanning trees and forests. Discrete Appl. Math. 177, 34–52.

Burkard, R.E., Karisch, S.E., Rendl, F., 1997. QAPLIB—a quadratic assignment problem library. J. Glob. Optim. 10, 391–403.

Cerulli, R., Fink, A., Gentili, M., Voss, S., 2005. Metaheuristics comparison for the minimum labelling spanning tree problem. In: The Next Wave in Computing, Optimization, and Decision Technologies, Operations Research/Computer Science Interfaces Series, vol. 29, pp. 93–106.

Christofides, N., Benavent, E., 1989. An exact algorithm for the quadratic assignment problem. Oper. Res. 37 (5), 760–768.

Cordone, R., Passeri, G., 2008. 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, pp. 168–171.

Cordone, R., Passeri, G., 2012. Solving the quadratic minimum spanning tree problem. Appl. Math. Comput. 218, 11597–11612.

Fischer, A., Fischer, F., 2013. Complete description for the spanning tree problem with one linearised quadratic term. Oper. Res. Lett. 41, 701–705.

Fu, Z.H., Hao, J.K., 2014. Breakout local search for the Steiner tree problem with revenue, budget and hop constraints. Eur. J. Oper. Res. 232 (1), 209–220.

Gao, J., Lu, M., 2005. Fuzzy quadratic minimum spanning tree problem. Appl. Math. Comput. 164 (3), 773–788.

Glover, F., Laguna, M., 1997. Tabu Search. Kluwer Academic Publishers, Norwell, MA, USA.

Lourenco, H.R., Martin, O., Stützle, T., 2003. Iterated local search, Handbook of Meta-heuristics. Springer-Verlag, Norwell, MA, USA.

Lozano, M., Glover, F., García-Martínez, C., Rodríguez, F.J., Martí, R., 2013. Tabu search with strategic oscillation for the quadratic minimum spanning tree. IIE Trans. 46 (4), 414–428.

Maia, S.M.D.M., Goldbarg, E.F.G., Goldbarg, M.C., 2013. On the biobjective adjacent only quadratic spanning tree problem. Electron. Notes Discrete Math. 41, 535–542.

Maia, S.M.D.M., Goldbarg, E.F.G., Goldbarg, M.C., 2014. Evolutionary algorithms for the bi-objective adjacent only quadratic spanning tree. Int. J. Innov. Comput. Appl. 6 (2), 63–72.

Nugent, C.E., Vollman, T.E., Ruml, J., 1968. An experimental comparison of techniques for the assignment of facilities to locations. Oper. Res. 16 (1), 150–173.

Öncan, T., Punnen, A.P., 2010. The quadratic minimum spanning tree problem: a lower bounding procedure and an efficient search. Comput. Oper. Res. 37 (10), 1762–1773.

Palubeckis, G., Rubliauskas, D., Targamadzè, A., 2010. Metaheuristic approaches for the quadratic minimum spanning tree problem. Inf. Technol. Control 39 (4), 257–268.

Pereira, D.L., Gendreau, M., Cunha, A.S.D., 2013a. Lower bounds and exact algorithms for the quadratic minimum spanning tree problem. Available online at url 〈http://www.optimization-online.org/DB_FILE/2013/12/4166.pdf〉.

Pereira, D.L., Gendreau, M., Cunha, A.S.D., 2013b. Stronger lower bounds for the quadratic minimum spanning tree problem with adjacency costs. Electron. Notes Discrete Math. 41, 229–236.

Pereira, D.L., Gendreau, M., Cunha, A.S.D., 2015. Branch-and-cut and branch-and-cut-and-price algorithms for the adjacent only quadratic minimum spanning tree problem. Networks 65 (4), 367–379.

Rostami, B., Malucelli, F., 2014. Lower bounds for the quadratic minimum spanning tree problem based on reduced cost computation. Available online at url 〈http://www.optimization-online.org/DB_FILE/2014/09/4555.pdf.

Soak, S.M., Corne, D.W., Ahn, B.H., 2005. A new evolutionary algorithm for spanning-tree based communication network design. IEICE Trans. Commun. 10, 4090–4093 E88-B.

Soak, S.M., Corne, D.W., Ahn, B.H., 2006. The edge-window-decoder representation for tree based problems. IEEE Trans. Evol. Comput. 10 (2), 124–144.

Sundar, S., Singh, A., 2010. A swarm intelligence approach to the quadratic minimum spanning tree problem. Inf. Sci. 180 (17), 3182–3191.

Xu, W., 1984. The Quadratic Minimum Spanning Tree Problem and Other Topics (Ph. D. thesis). University of Maryland, College Park, MD.

Xu, W., 1995. On the quadratic minimum spanning tree problem. In: Japan–China International Workshop on Information Systems, Ashikaga, pp. 141–148.

Zhou, G., Gen, M., 1998. An effective genetic algorithm approach to the quadratic minimum spanning tree problem. Comput. Oper. Res. 25 (3), 229–237.