# A Three-Phase Search Approach for the Quadratic Minimum Spanning Tree Problem

Zhang-Hua Fu [a,b] and 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*

***Engineering Applications of Artificial Intelligence 46:113-130, 2015***

**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.

*Keywords*: Spanning tree; network design; neighborhood search; heuristics.

* Corresponding author.

*Email addresses:* fu@info.univ-angers.fr (Zhang-Hua Fu), hao@info.univ-angers.fr (Jin-Kao Hao).

## 1 Introduction

Network design is an extremely challenging task in numerous resource distribution systems (e.g., transportation, electricity, telecommunication, computer networks, etc.). 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 $c : E \to \mathbb{R}$ be a linear cost function for the set of edges and $q : E \times E \to \mathbb{R}$ be a quadratic cost function to weight 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 [11], this problem can be formulated as follows:

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

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

$$\sum_{e \in E(S)} x_e \leq |S| - 1, \forall\ S \subset V \text{ with } |S| \geq 3, \tag{3}$$

$$x_e \in \{0, 1\}, \forall\ e \in E. \tag{4}$$

where $x_e = 1$ if edge $e$ belongs to the solution, $x_e = 0$ otherwise. $S$ is any possible subset of $V$ (with $|S| \geq 3$) 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 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) [18, 19, 24, 25], but in some special cases, the interference costs also exist between any pair of edges. As discussed in [1, 21, 22], 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 [31], 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 [1, 31] propose a Lagrangian branch-and-bound (B&B) method. Öncan and Punnen [21] combine the Lagrangian relaxation scheme with an extended formulation of valid inequalities to obtain tighter bounds. Cordone and Passeri [11] improve the Lagrangian B&B procedure in [1]. Pereira et al. [23] 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 [26] 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 [5, 6] propose a B&B approach for QMSTP with one quadratic term in the objective function, of which the polyhedral descriptions are completed in [12]. Pereira et al. [24] 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 [1, 30, 31]. Several genetic algorithms are implemented by Zhou and Gen [32] and tested on instances with up to 50 vertices. Another evolutionary algorithm is proposed for a fuzzy variant of QMSTP [14], using the Prüfer number to encode a spanning tree. Soak et al. [27, 28] 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 [21] alternatively performs local search and random moves. In [22], 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 [29]. Cordone and Passeri [11] 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. [17] propose an iterated greedy (IG) and a strategic oscillation (SO) heuristic, and combine them with the ITS [22] 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 [18] as well as several evolutionary algorithms [19].

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 [16], reactive tabu search [2, 8] and breakout local search [3, 4, 13]. 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 $E \rightarrow \mathbb{R}$, quadratic function $E \times E \rightarrow \mathbb{R}$
**Result**: The best solution found

```
1 T ← Init_Solution() ;   /* Construct an initial solution, Section 2.3 */
2 T^best ← 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 ← Descent_Neighborhood_Search(T) ;
      // Explore nearby local optima, Algorithm 2 and Section 2.6
5     T ← Explore_Local_Optima(T) ;
      // Update T^best if an improved solution is found
6     if F(T) < F(T^best) then
7         T^best ← T ;
      // Strongly perturb the incumbent solution, Section 2.7
8     T ← 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` $\rightarrow$ `F`) and *Explore_Local_Optima* (`F` $\rightarrow$ `G`), to obtain a high-quality solution `G`.

### *2.2 Solution representation*

Like the compact tree representation used in [11, 13], we uniquely represent each feasible solution $T$ as a rooted tree (with vertex 1 fixed as the root vertex, being different from [11] 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.

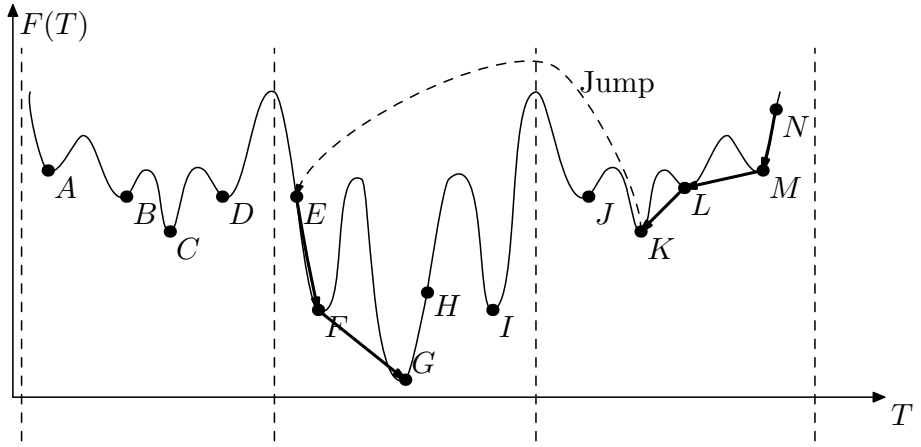
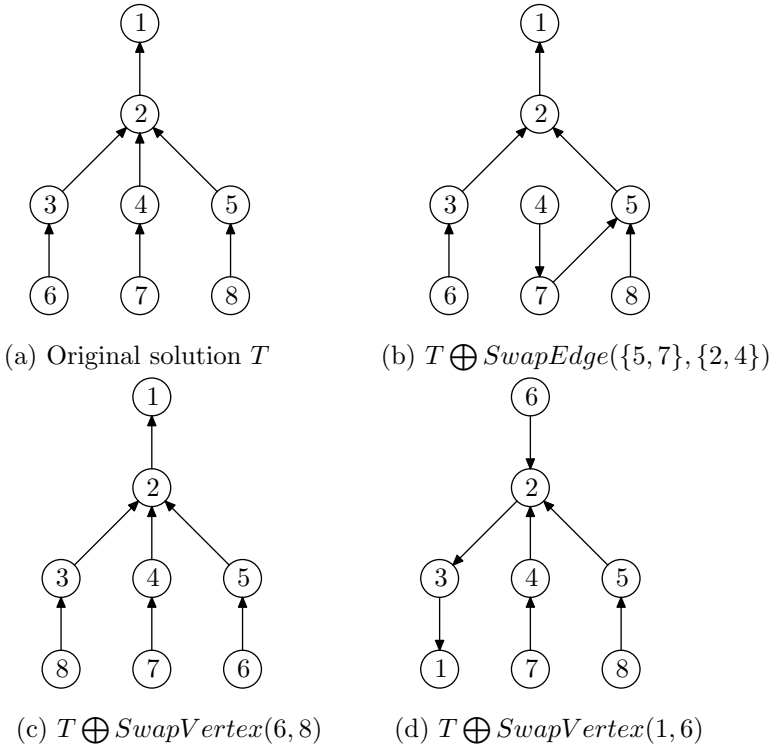



Fig. 1. Procedure of searching a high-quality feasible solution of QMSTP

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




(a) Original solution $T$ (b) $T \bigoplus SwapEdge(\{5,7\},\{2,4\})$

(c) $T \bigoplus SwapVertex(6,8)$ (d) $T \bigoplus SwapVertex(1,6)$

Fig. 2. Move operators for generating neighboring solutions

(1) The first one is the conventional *swap-edge move operator* inherited from [11,17,22]. 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 $|L_e| \leq n$ edges, and then removes an edge $f$ from $L_e \backslash \{e\}$, to obtain a feasible neighboring solution denoted by $T \bigoplus SwapEdge(e, f)$. The corresponding difference of the objective function (also called move gain) is denoted by $\delta_{ef}$.

(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 $r_i \neq r_j$ and $\{i, r_j\} \in E, \{j, r_i\} \in E$, a feasible neighboring solution denoted by $T \bigoplus SwapVertex(i, j)$ could be generated by swapping vertices $i$ and $j$, leading to a difference $\delta_{ij}$ 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 $SwapEdge(e1, f1) \bigoplus SwapEdge(e2, f2)$. 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:

$$\begin{aligned} N1 &= \{T \bigoplus SwapEdge(e, f) \mid e \in E \backslash X,\ f \in L_e \backslash \{e\}\}, \\ N2 &= \{T \bigoplus SwapVertex(i, j) \mid i, j \in V^1, r_i \neq r_j, \{i, r_j\} \in E, \{j, r_i\} \in E\}. \end{aligned} \tag{5}$$

where $T = (V, X)$ is a feasible solution, and $T \bigoplus SwapEdge(e, f)$ (respectively, $T \bigoplus SwapVertex(i, j)$) 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 $|N1| = O(m - n + 1) \times O(max(|L_e|)) \leq O(mn) \leq O(n^3)$, and $|N2| = O(|V^1|^2) \leq O(n^2)$, 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 [11], 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 $T = (V, X)$.

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

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:

$$\delta_{ef} = D_e - D_f - q_{ef} - q_{fe}. \tag{7}$$

which can be calculated in constant time $O(1)$ [11]. After performing the chosen move $SwapEdge(e, f)$, as in [11], vector $D$ is updated in $O(m)$ as follows:

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

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 $SwapEdge(e1, f1) \bigoplus SwapEdge(e2, f2)$, 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:

$$\begin{aligned} \delta_{ij} = {} & D_{e1} + D_{e2} - D_{f1} - D_{f2} + q_{e1e2} + q_{e2e1} + q_{f1f2} + q_{f2f1} \\ & - q_{e1f1} - q_{f1e1} - q_{e1f2} - q_{f2e1} - q_{e2f1} - q_{f1e2} - q_{e2f2} - q_{f2e2}. \end{aligned} \tag{9}$$

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

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

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 $\gamma = Max\{D_g, g \in X\}$ denote the maximum cost value of $D_g$ of all the edges $g$ belonging to the incumbent solution $T = (V, X)$, and let $\lambda = Max\{q_{hk} + q_{kh},\ h,\ k \in E\}$ denote the maximum possible value of quadratic costs between any pair of edges. Note that $\gamma$ is a variable which should be updated at each iteration, within an amount of $O(|X|) = O(n)$ extra time, while $\lambda$ 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 [3, 4], these directed perturbation operators rely on the tabu search principle [15], 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 [15]. 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♯ ← T ;                          /* T♯ records the best found local optimum */
2  ω ← 0 ; /* ω counts the number of consecutive non-improving rounds */
3  while ω < ω_max do
       // Apply a directed perturbation operator to perturb T
4      T ← Directed_Perturb(T) ;
       // Optimize T to a new local optimum
5      T ← Descent_Neighborhood_Search(T) ;
       // Update the best found solution and the value of ω
6      if F(T) < F(T♯) then
7          T♯ ← T ;
8          ω ← 0 ;
9      else
10         ω ← ω + 1 ;
11 return T♯ ;
```

(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 $\omega_{max}$ consecutive such rounds ($\omega_{max}$ 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) [16], reactive tabu search (RTS) [2, 8] and breakout local search (BLS) [3, 4]. 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 tabu mechanism to forbid visited solutions. However, unlike RTS, TPS uses 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 [28], a 3 GHz Pentium IV CPU with 2 GB RAM in [21], a 3.0 GHz core 2 duo system with 2 GB RAM in [29], a 3.0 GHz Intel core 2 duo in [22], a 2.6 GHz Intel Pentium Core 2 Duo E6700 and 2 GB RAM in [11], a 3.2 GHz Intel processor with 12 GB RAM in [17]. One can observe that the clock frequency of our processor is about 80% faster than the computer used in [28], while being similar to the machines used in [11, 17, 21, 22, 29]. 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.

[1] All the best solutions reported in this paper are available on `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.

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 [11,17,23], 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 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 [10] 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 [21] 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:

(1) SYM: with linear costs uniformly distributed at random within [1,100], and quadratic ones within [1,20];
(2) VSYM: the linear costs are uniformly distributed at random in [1,10000], 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 $f$;
(3) 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.

[2] The *CP* benchmarks can be downloaded from `http://www.dti.unimi.it/cordone/research/qmst.html` and the RAND and SOAK instances are available at `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).

- Benchmark SCA [28] 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 [29] 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 [17] 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 [17] 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 [20] and CHR [9] benchmarks of the Quadratic Assignment Problem (QAP), using a one-to-one transformation procedure between the two problems [21]. Note that, although the original QAP instances have already been solved to optimality by previous QAP algorithms [7], 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.

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.

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 | $[n, 5n]$ |

### *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 [22], QMST-TS [11], and HSII [17] 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 solve each of its selected instances, with a cutoff time of 10 seconds 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 [23] can

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

solve all the instances with up to 20 vertices and 127 edges to optimality with a time limit of 100 hours. Recently, Rostami and Malucelli [26] 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 [21], the B&B algorithm QMST-BB in [11], the enhanced B&B algorithms in [23], and the reformulation scheme for computing lower bounds in [26]. Nevertheless, even the best existing exact approach [23] 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 [21] and the tabu search algorithm QMST-TS in [11]. For the instances of subclass SYM with $20 \leq |V| \leq 50$, 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 [11,23], 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 can not be further improved after 10 consecutive rounds of *Descent_Neighborhood_Search*

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

| $\|V\|$ | $\|E\|$ | Index | QMST-TS [11] | | TPS | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | *Best* | $T(s)$ | *Best* | $T(s)$ | $t(s)$ | $\sigma$ | $\surd$ | $=$ | $\times$ |
| 50 | 1225 | 2 | 17600 | 25.38 | **17587**$*$ | 4.88 | 0.49 | 0.19 | 3 | 0 | 7 |
| 50 | 1225 | 7 | 17643 | 25.29 | **17633**$*$ | 4.00 | 0.40 | 0.14 | 5 | 0 | 5 |
| 50 | 1225 | 8 | 17685 | 25.11 | **17663**$*$ | 5.32 | 0.53 | 0.18 | 6 | 0 | 4 |
| 50 | 1225 | 10 | 17639 | 25.35 | **17623**$*$ | 5.71 | 0.57 | 0.17 | 5 | 0 | 5 |
| Average | - | - | 17641.8 | 25.28 | **17626.5**$*$ | 4.98 | 0.50 | 0.17 | 4.75 | 0 | 5.25 |

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 $\sigma$), 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 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. [28] and have been widely used by various heuristics [11, 21, 28, 29]. For each of these instances, we independently run TPS 20 times (like QMST-TS [11]), 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 [28], RLS-TT [21], ABC [29], QMST-TS [11]). The first column provides the problem size $|V|$, while the following eight columns indicate the best found results and the accumulated CPU times (in seconds) of each competing

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

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 $\sigma$) 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.

Table 3
Results of the SCA instances

| $\|V\|$ | EWD [28] | | RLS-TT [21] | | ABC [29] | | QMST-TS [11] | | TPS | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Best* | $T(s)$ | *Best* | $T(s)$ | *Best* | $T(s)$ | *Best* | $T(s)$ | *Best* | $T(s)$ | $t(s)$ | $\sigma$ | $\surd$ | $=$ | $\times$ |
| 50 | 25339 | 343.0 | 25226 | 3242.1 | **25200** | 87.0 | **25200** | 44.5 | **25200** | 5.6 | 0.3 | 0.0 | 0 | 20 | 0 |
| 60 | 36086 | 495.7 | 35754 | 4321.4 | 35466 | 169.0 | **35447** | 83.2 | **35447** | 12.6 | 0.6 | 0.1 | 0 | 20 | 0 |
| 70 | 48538 | 716.6 | 48536 | 5738.5 | **48125** | 337.2 | **48125** | 178.5 | **48125** | 21.3 | 1.1 | 0.2 | 0 | 20 | 0 |
| 80 | 63546 | 1086.7 | 63546 | 7026.3 | 63022 | 417.8 | 63004 | 340.4 | **62963**$*$ | 41.1 | 2.1 | 0.6 | 16 | 0 | 4 |
| 90 | 79627 | 1337.2 | 79922 | 8623.6 | **78879** | 751.8 | 78912 | 579.7 | **78879** | 55.1 | 2.8 | 0.8 | 0 | 6 | 14 |
| 100 | 98342 | 1828.9 | 98811 | 10431.3 | **96750** | 1542.4 | 96757 | 789.4 | **96750** | 89.2 | 4.5 | 1.3 | 0 | 10 | 10 |
| Average | 58579.7 | 968.0 | 58632.5 | 6563.9 | 57907.0 | 550.9 | 57907.5 | 336.0 | **57894.0**$*$ | 37.5 | 0.38 | 0.5 | 2.7 | 12.7 | 4.7 |

### *3.6 Results of the SS instances*

These 18 instances have been used to test ABC [29] and QMST-TS [11]. For comparison, for each of these instances, we independently run TPS 20 times (like QMST-TS), each run stops if the best found solution can not 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. [17] propose a hybrid heuristic named HSII and evaluate its performance using two groups (RAND and SOAK) of 18 newly gen-

Table 4
Results of the SS instances

| $\|V\|$ | Index | ABC [29] | | QMST-TS [11] | | TPS | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | *Best* | $T(s)$ | *Best* | $T(s)$ | *Best* | $T(s)$ | $t(s)$ | $\sigma$ | $\surd$ | $=$ | $\times$ |
| 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 | **21126** | 173.6 | **21126** | 50.4 | **21126** | 5.5 | 0.28 | 0.10 | 0 | 14 | 6 |
| 50 | 2 | 21123 | 176.8 | **21106** | 50.4 | **21106** | 5.5 | 0.28 | 0.09 | 0 | 8 | 12 |
| 50 | 3 | **21059** | 190.2 | **21059** | 50.6 | **21059** | 4.2 | 0.21 | 0.06 | 0 | 16 | 4 |
| 100 | 1 | 89098 | 2333.2 | 88871 | 965.8 | **88745**$*$ | 81.2 | 4.06 | 1.12 | 2 | 0 | 18 |
| 100 | 2 | 89202 | 2319.0 | 89049 | 957.7 | **88911**$*$ | 76.4 | 3.82 | 1.62 | 7 | 0 | 13 |
| 100 | 3 | 89007 | 1977.6 | 88720 | 961.2 | **88659**$*$ | 87.5 | 4.37 | 1.76 | 3 | 0 | 17 |
| 150 | 1 | 205619 | 8897.4 | 205615 | 2928.7 | **204995**$*$ | 510.6 | 25.53 | 9.70 | 12 | 0 | 8 |
| 150 | 2 | 205874 | 7486.6 | 205509 | 2923.0 | **205219**$*$ | 493.7 | 24.68 | 10.56 | 4 | 0 | 16 |
| 150 | 3 | 205634 | 8658.6 | 205094 | 2928.6 | **205076**$*$ | 596.7 | 29.84 | 10.73 | 1 | 0 | 19 |
| 200 | 1 | 371797 | 22828.4 | 371492 | 6320.3 | **370873**$*$ | 1519.5 | 75.98 | 36.28 | 9 | 0 | 11 |
| 200 | 2 | 371864 | 23112.0 | 371698 | 6332.1 | **370853**$*$ | 1386.3 | 69.32 | 25.17 | 15 | 0 | 5 |
| 200 | 3 | 372156 | 25534.2 | 371584 | 6324.3 | **370954**$*$ | 1282.8 | 64.14 | 21.80 | 9 | 0 | 11 |
| 250 | 1 | 587924 | 51268.2 | 586861 | 9572.3 | **586265**$*$ | 2916.5 | 145.82 | 45.36 | 7 | 0 | 13 |
| 250 | 2 | 588068 | 56818.2 | 587607 | 9592.9 | **586778**$*$ | 2253.3 | 112.66 | 31.68 | 17 | 0 | 3 |
| 250 | 3 | 587883 | 46565.8 | 587281 | 9601.2 | **585851**$*$ | 2709.9 | 135.49 | 53.58 | 14 | 0 | 6 |
| Average | - | 213475.7 | 14355.5 | 213211.1 | 3310.0 | **212866.6*** | 774.0 | 38.70 | 13.87 | 5.56 | 5.39 | 9.06 |

Table 5
Results of the RAND instances

| Instance | ABC [29] | | ITS [22] | | HSII [17] | | TPS | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Best* | $T(s)$ | *Best* | $T(s)$ | *Best* | $T(s)$ | *Best* | $T(s)$ | $t(s)$ | $\sigma$ | $\surd$ | $=$ | $\times$ |
| RAND-150-1 | 194294 | 4000 | 192946 | 4000 | 192606 | 4000 | **192369**$*$ | 4000 | 400 | 0 | 2 | 0 | 8 |
| RAND-150-2 | 194218 | 4000 | 193034 | 4000 | 192607 | 4000 | **192579**$*$ | 4000 | 400 | 0 | 1 | 0 | 9 |
| RAND-150-3 | 193882 | 4000 | 192965 | 4000 | 192577 | 4000 | **192046**$*$ | 4000 | 400 | 0 | 4 | 0 | 6 |
| RAND-200-1 | 353163 | 12000 | 351216 | 12000 | 350517 | 12000 | **350321**$*$ | 12000 | 1200 | 0 | 3 | 0 | 7 |
| RAND-200-2 | 353784 | 12000 | 351312 | 12000 | 350389 | 12000 | 350658$^{\dagger 1}$ | 12000 | 1200 | 0 | 0 | 0 | 10 |
| RAND-200-3 | 353169 | 12000 | 351466 | 12000 | 351057 | 12000 | **350601**$*$ | 12000 | 1200 | 0 | 7 | 0 | 3 |
| RAND-250-1 | 561864 | 20000 | 558451 | 20000 | 556929 | 20000 | 557278$^{\dagger 2}$ | 20000 | 2000 | 0 | 0 | 0 | 10 |
| RAND-250-2 | 560704 | 20000 | 558820 | 20000 | 557474 | 20000 | **556604**$*$ | 20000 | 2000 | 0 | 1 | 0 | 9 |
| RAND-250-3 | 561497 | 20000 | 559304 | 20000 | 556813 | 20000 | 557060$^{\dagger 3}$ | 20000 | 2000 | 0 | 0 | 0 | 10 |
| Average | 369619.4 | 12000 | 367723.8 | 12000 | 366774.3 | 12000 | **366612.9*** | 12000 | 1200 | 0 | 2.0 | 0.0 | 8.0 |

†1: TPS achieves an improved value of **350231** (within 12000 seconds) with re-tuned parameters $L_{dir} \in [0.1n, n]$ and $L_{div} \in [n, 2n]$
†2: TPS achieves an improved value of **556596** (within 20000 seconds) with re-tuned parameter $L_{div} \in [0.5n, 2n]$
†3: TPS achieves an improved value of **556378** (within 20000 seconds) with re-tuned parameter $L_{div} \in [0.5n, 5n]$

Table 6
Results of the SOAK instances

| Instance | ABC [29] | | ITS [22] | | HSII [17] | | TPS | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Best* | $T(s)$ | *Best* | $T(s)$ | *Best* | $T(s)$ | *Best* | $T(s)$ | $t(s)$ | $\sigma$ | $\surd$ | $=$ | $\times$ |
| SOAK-150-1 | 207652 | 4000 | **206721** | 4000 | 206925 | 4000 | **206721** | 4000 | 400 | 0 | 0 | 1 | 9 |
| SOAK-150-2 | 208206 | 4000 | **206761** | 4000 | 207102 | 4000 | **206761** | 4000 | 400 | 0 | 0 | 5 | 5 |
| SOAK-150-3 | 207533 | 4000 | 206802 | 4000 | 206781 | 4000 | **206759**$*$ | 4000 | 400 | 0 | 2 | 3 | 5 |
| SOAK-200-1 | 372419 | 12000 | 370137 | 12000 | 370265 | 12000 | **369851**$*$ | 12000 | 1200 | 0 | 5 | 0 | 5 |
| SOAK-200-2 | 371641 | 12000 | 370028 | 12000 | 369982 | 12000 | **369803**$*$ | 12000 | 1200 | 0 | 8 | 0 | 2 |
| SOAK-200-3 | 372117 | 12000 | 370046 | 12000 | 370045 | 12000 | **369794**$*$ | 12000 | 1200 | 0 | 7 | 0 | 3 |
| SOAK-250-1 | 584799 | 20000 | 582282 | 20000 | 581819 | 20000 | **581671**$*$ | 20000 | 2000 | 0 | 1 | 0 | 9 |
| SOAK-250-2 | 584409 | 20000 | 582145 | 20000 | 581691 | 20000 | **581492**$*$ | 20000 | 2000 | 0 | 2 | 0 | 8 |
| SOAK-250-3 | 585717 | 20000 | 582708 | 20000 | 581854 | 20000 | **581573**$*$ | 20000 | 2000 | 0 | 1 | 0 | 9 |
| Average | 388277.0 | 12000 | 386403.3 | 12000 | 386273.8 | 12000 | **386047.2*** | 12000 | 1200 | 0 | 2.9 | 1.0 | 6.1 |

erated benchmarks, in comparison with two previous heuristics, i.e., ITS [22] and ABC [29] [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 seconds respectively for instances with $|V|$=150, 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 [17] 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 the 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 [9] and 15 NUG ones [20]), while guaranteeing a one-to-one correspondence of the feasible solutions after transformation [21]. For each of these 29 instances, we independently run TPS 10 times, each run continues until the best found solution can not 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 [7] [6] . The

[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 `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.

[6] The optimal solutions of the original QAP instances are available online at the

Table 7
Results of $TPS$ on the QAP-QMSTP instances compared with RLS-TT [21], ABC [29], ITS [22], HSII [17] and QMST-TS with re-tuned parameters [11].

| 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* | $T(s)$ | $t(s)$ | $\sigma$ | $=$ | $\times$ |
| *chr*12*a* | 9552 | 11170 | 783 | 14290 | 10000 | 16694 | 10000 | **9552** | 10000 | **9552** | 288 | **9552** | 43 | 4.3 | 0.1 | 10 | 0 |
| *chr*12*b* | 9742 | 10753 | 790 | 21552 | 10000 | 16356 | 10000 | **9742** | 10000 | **9742** | 287 | **9742** | 37 | 3.7 | 0.0 | 10 | 0 |
| *chr*12*c* | 11156 | 12712 | 783 | 15810 | 10000 | 17434 | 10000 | **11156** | 10000 | **11156** | 286 | **11156** | 44 | 4.4 | 0.2 | 10 | 0 |
| *chr*15*a* | 9896 | 11638 | 1239 | 24224 | 10000 | 16718 | 10000 | 9952 | 10000 | 9936 | 497 | **9896** | 92 | 9.2 | 0.5 | 10 | 0 |
| *chr*15*b* | 7990 | 10145 | 1136 | 28340 | 10000 | 17208 | 10000 | 8384 | 10000 | **7990** | 492 | **7990** | 90 | 9.0 | 0.3 | 10 | 0 |
| *chr*15*c* | 9504 | 12769 | 1254 | 25566 | 10000 | 19302 | 10000 | **9504** | 10000 | **9504** | 492 | **9504** | 91 | 9.1 | 0.4 | 10 | 0 |
| *chr*18*a* | 11098 | 12757 | 3325 | 24954 | 10000 | 22496 | 10000 | 13834 | 10000 | **11098** | 793 | **11098** | 170 | 17.0 | 1.0 | 10 | 0 |
| *chr*18*b* | 1534 | 1676 | 3354 | 2160 | 10000 | **1534** | 10000 | **1534** | 10000 | **1534** | 789 | **1534** | 148 | 14.8 | 0.2 | 10 | 0 |
| *chr*20*a* | 2192 | 2445 | 4968 | 4742 | 10000 | 2232 | 10000 | 2276 | 10000 | **2192** | 1043 | **2192** | 356 | 35.6 | 8.0 | 6 | 4 |
| *chr*20*b* | 2298 | 2730 | 4652 | 3704 | 10000 | 2440 | 10000 | 2462 | 10000 | 2352 | 1044 | **2298** | 482 | 48.2 | 15.2 | 3 | 7 |
| *chr*20*c* | 14142 | 30124 | 4763 | 49842 | 10000 | 36558 | 10000 | 20206 | 10000 | 14202 | 1046 | **14142** | 260 | 26.0 | 0.8 | 10 | 0 |
| *chr*22*a* | 6156 | 8760 | 5089 | 8688 | 10000 | 6390 | 10000 | 6334 | 10000 | 6228 | 1395 | **6156** | 450 | 45.0 | 4.8 | 10 | 0 |
| *chr*22*b* | 6194 | 8402 | 4741 | 8908 | 10000 | 6314 | 10000 | 6396 | 10000 | 6314 | 1422 | **6194** | 581 | 58.1 | 15.0 | 3 | 7 |
| *chr*25*a* | 3796 | 9658 | 5223 | 8540 | 10000 | 4300 | 10000 | 4310 | 10000 | 3866 | 2135 | **3796** | 841 | 84.1 | 24.0 | 7 | 3 |
| *nug*12 | 578 | 605 | 639 | 656 | 10000 | **578** | 10000 | **578** | 10000 | **578** | 287 | **578** | 40 | 4.0 | 0.1 | 10 | 0 |
| *nug*14 | 1014 | 1084 | 724 | 1140 | 10000 | **1014** | 10000 | 1026 | 10000 | **1014** | 414 | **1014** | 67 | 6.7 | 0.1 | 10 | 0 |
| *nug*15 | 1150 | 1265 | 1348 | 1404 | 10000 | **1150** | 10000 | 1152 | 10000 | **1150** | 493 | **1150** | 78 | 7.8 | 0.0 | 10 | 0 |
| *nug*16*a* | 1610 | 1742 | 2311 | 1944 | 10000 | 1638 | 10000 | 1634 | 10000 | 1622 | 583 | **1610** | 102 | 10.2 | 0.3 | 10 | 0 |
| *nug*16*b* | 1240 | 1350 | 2936 | 1480 | 10000 | 1248 | 10000 | 1246 | 10000 | **1240** | 579 | **1240** | 97 | 9.7 | 0.1 | 10 | 0 |
| *nug*17 | 1732 | 1874 | 3422 | 2066 | 10000 | 1768 | 10000 | 1774 | 10000 | 1750 | 685 | **1732** | 123 | 12.3 | 0.3 | 10 | 0 |
| *nug*18 | 1930 | 2056 | 3482 | 2224 | 10000 | 1964 | 10000 | 1984 | 10000 | 1942 | 793 | **1930** | 152 | 15.2 | 0.3 | 10 | 0 |
| *nug*20 | 2570 | 2860 | 5151 | 2900 | 10000 | 2644 | 10000 | 2662 | 10000 | 2580 | 1047 | **2570** | 228 | 22.8 | 0.3 | 10 | 0 |
| *nug*21 | 2438 | 2698 | 5184 | 3042 | 10000 | 2502 | 10000 | 2540 | 10000 | 2488 | 1201 | **2438** | 248 | 24.8 | 1.6 | 10 | 0 |
| *nug*22 | 3596 | 3868 | 5482 | 4580 | 10000 | 3712 | 10000 | 3750 | 10000 | 3672 | 1371 | **3596** | 317 | 31.7 | 0.6 | 10 | 0 |
| *nug*24 | 3488 | 3874 | 5914 | 4340 | 10000 | 3648 | 10000 | 3688 | 10000 | 3590 | 1838 | **3488** | 410 | 41.0 | 1.4 | 10 | 0 |
| *nug*25 | 3744 | 4083 | 5983 | 4522 | 10000 | 3954 | 10000 | 3940 | 10000 | 3874 | 2098 | **3744** | 475 | 47.5 | 5.2 | 10 | 0 |
| *nug*27 | 5234 | 5966 | 6025 | 6284 | 10000 | 5456 | 10000 | 5534 | 10000 | 5352 | 2788 | **5234** | 651 | 65.1 | 7.6 | 10 | 0 |
| *nug*28 | 5166 | 5819 | 6087 | 6238 | 10000 | 5406 | 10000 | 5484 | 10000 | 5262 | 3228 | **5166** | 771 | 77.1 | 6.8 | 10 | 0 |
| *nug*30 | 6124 | 6923 | 6227 | 7688 | 10000 | 6506 | 10000 | 6528 | 10000 | 6364 | 4283 | **6124** | 1251 | 125.1 | 21.5 | 10 | 0 |
| Average | 5064.3 | 6614.0 | 3552.2 | 10063.0 | 10000.0 | 7902.2 | 10000.0 | 5488.3 | 10000.0 | 5108.4 | 1161.9 | **5064.3*** | 299.8 | 30.0 | 4.0 | 9.3 | 0.7 |

next 10 columns report the best results and the accumulated CPU times (in seconds) of each compared algorithm, i.e., RLS-TT [21], ABC [29], ITS [22], HSII [17] and QMST-TS with re-tuned parameters [11]. Note that, like for the RAND and SOAK instances, the results of ABC and ITS are reproduced by Lozano et al. [17]. 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.

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.

QAPLIB: `http://www.seas.upenn.edu/qaplib`.

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% |

### *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 $\frac{\sum_{I \in Gr} \frac{F1(I)-F2(I)}{F1(I)}}{|Gr|}$, 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 $5 \times 10^{-2}$) on almost every group of instances, with only three exceptions, i.e., TPS ↔ ABC, TPS ↔ QMST-TS on group SCA, and TPS ↔ HSII on group RAND.

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

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 ↔ EWD | $1.43 \times 10^{-2}$ | - | - | - | - |
| TPS ↔ RLS-TT | $1.43 \times 10^{-2}$ | - | - | - | $7.24 \times 10^{-8}$ |
| TPS ↔ ABC | $1.57 \times 10^{-1}$ | $3.12 \times 10^{-4}$ | $2.70 \times 10^{-3}$ | $2.70 \times 10^{-3}$ | $7.24 \times 10^{-8}$ |
| TPS ↔ ITS | - | - | $2.71 \times 10^{-3}$ | $8.15 \times 10^{-3}$ | $5.73 \times 10^{-7}$ |
| TPS ↔ QMST-TS | $8.32 \times 10^{-2}$ | $5.32 \times 10^{-4}$ | - | - | $3.74 \times 10^{-5}$ |
| TPS ↔ HSII | - | - | $3.17 \times 10^{-1}$ | $2.69 \times 10^{-3}$ | $1.62 \times 10^{-6}$ |

Table 10
Importance of the pre-estimation criterion

| Group | Total ($\times 10^3$) | Discarded ($\times 10^3$) | 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% |

## 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 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 Section 2 and Section 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 $2.04 \times 10^{-11}$ and $7.13 \times 10^{-13}$ 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 $2.84 \times 10^{-5}$ and $7.43 \times 10^{-6}$ 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

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 | **88730** | **88790.3** | 3600 | 88780 | 88865.0 | 3600 | 88780 | 88874.1 | 3600 |
| SS-100-2 | 88772 | **88806.9** | 3600 | **88725** | 88833.5 | 3600 | 88855 | 88915.8 | 3600 |
| SS-100-3 | **88619** | **88619.0** | 3600 | **88619** | 88656.1 | 3600 | **88619** | 88762.8 | 3600 |
| SS-150-1 | 205011 | **205069.3** | 3600 | **204984** | 205258.5 | 3600 | 205350 | 205641.0 | 3600 |
| SS-150-2 | **204873** | 205178.4 | 3600 | 204991 | **205143.5** | 3600 | 205337 | 205736.9 | 3600 |
| SS-150-3 | **204770** | **205034.5** | 3600 | 204992 | 205233.4 | 3600 | 205429 | 205700.7 | 3600 |
| SS-200-1 | **370506** | **370941.1** | 3600 | 370587 | 371264.7 | 3600 | 371532 | 372135.9 | 3600 |
| SS-200-2 | 370827 | **371059.6** | 3600 | **370825** | 371178.3 | 3600 | 371727 | 372080.9 | 3600 |
| SS-200-3 | **370582** | **371159.2** | 3600 | 370990 | 371472.9 | 3600 | 371840 | 372278.4 | 3600 |
| SS-250-1 | **586812** | **587174.0** | 3600 | 586968 | 587586.5 | 3600 | 588403 | 589189.4 | 3600 |
| SS-250-2 | **586689** | **587157.5** | 3600 | 586980 | 587445.2 | 3600 | 588743 | 589361.5 | 3600 |
| SS-250-3 | 586717 | **587004.1** | 3600 | **586269** | 587168.3 | 3600 | 588805 | 589377.7 | 3600 |
| RAND-150-1 | **192369** | **192754.6** | 4000 | 192560 | 192909.9 | 4000 | 192827 | 193388.0 | 4000 |
| RAND-150-2 | **192579** | **192856.1** | 4000 | 192587 | 192931.8 | 4000 | 193297 | 193508.4 | 4000 |
| RAND-150-3 | **192046** | **192583.6** | 4000 | 192496 | 192876.0 | 4000 | 193067 | 193420.6 | 4000 |
| RAND-200-1 | **350321** | **350901.1** | 12000 | 350676 | 351147.8 | 12000 | 351703 | 352196.6 | 12000 |
| RAND-200-2 | **350658** | **350880.5** | 12000 | 350711 | 351184.9 | 12000 | 351828 | 352114.6 | 12000 |
| RAND-200-3 | 350601 | **350980.1** | 12000 | **350506** | 351152.2 | 12000 | 351680 | 352305.1 | 12000 |
| RAND-250-1 | **557278** | **557823.4** | 20000 | 557626 | 558193.0 | 20000 | 558656 | 559514.1 | 20000 |
| RAND-250-2 | **556604** | **557822.8** | 20000 | 557789 | 558200.4 | 20000 | 559387 | 560003.1 | 20000 |
| RAND-250-3 | **557060** | **557762.0** | 20000 | 557399 | 558045.7 | 20000 | 558690 | 559650.3 | 20000 |
| SOAK-150-1 | **206721** | 206900.7 | 4000 | **206721** | **206890.4** | 4000 | 206895 | 207030.9 | 4000 |
| SOAK-150-2 | **206761** | **206894.5** | 4000 | 206922 | 207176.1 | 4000 | 206922 | 207425.3 | 4000 |
| SOAK-150-3 | **206759** | **206839.9** | 4000 | 206785 | 206889.4 | 4000 | 206898 | 207164.0 | 4000 |
| SOAK-200-1 | **369851** | **370220.1** | 12000 | 369990 | 370377.8 | 12000 | 370741 | 371148.6 | 12000 |
| SOAK-200-2 | **369803** | **369932.6** | 12000 | 369822 | 370263.8 | 12000 | 370590 | 371117.7 | 12000 |
| SOAK-200-3 | **369794** | **370010.2** | 12000 | 370089 | 370182.8 | 12000 | 370403 | 370709.1 | 12000 |
| SOAK-250-1 | **581671** | 582364.4 | 20000 | 581959 | **582357.1** | 20000 | 582935 | 583791.0 | 20000 |
| SOAK-250-2 | **581492** | **581903.0** | 20000 | 581671 | 582154.7 | 20000 | 581884 | 583431.3 | 20000 |
| SOAK-250-3 | **581573** | 582323.8 | 20000 | 581596 | **582290.2** | 20000 | 582641 | 583270.8 | 20000 |
| 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 | 14214 | 14929.8 | 3600 | **14142** | **14142.0** | 3600 | **14142** | **14142.0** | 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 |

instances, one finds that TPS-V3 performs similarly 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) [9]. According to the transformation rules described in [21], 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 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

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 | **88730** | **88790.3** | 3600 | 88829 | 88919.4 | 3600 | 88780 | 88819.5 | 3600 |
| SS-100-2 | **88772** | **88806.9** | 3600 | 88862 | 88947.6 | 3600 | 88773 | 88829.9 | 3600 |
| SS-100-3 | **88619** | **88619.0** | 3600 | 88671 | 88762.2 | 3600 | **88619** | 88664.8 | 3600 |
| SS-150-1 | 205011 | **205069.3** | 3600 | 205436 | 205632.3 | 3600 | **204936** | 205266.5 | 3600 |
| SS-150-2 | **204873** | 205178.4 | 3600 | 205492 | 205629.6 | 3600 | 204911 | **205134.1** | 3600 |
| SS-150-3 | **204770** | **205034.5** | 3600 | 205232 | 205556.7 | 3600 | 204860 | 205072.3 | 3600 |
| SS-200-1 | **370506** | **370941.1** | 3600 | 370938 | 371760.9 | 3600 | 370827 | 371103.8 | 3600 |
| SS-200-2 | **370827** | **371059.6** | 3600 | 371283 | 371731.6 | 3600 | 370937 | 371169.5 | 3600 |
| SS-200-3 | **370582** | **371159.2** | 3600 | 371210 | 371779.6 | 3600 | 370892 | 371192.2 | 3600 |
| SS-250-1 | 586812 | 587174.0 | 3600 | 587332 | 587873.1 | 3600 | **586302** | **586831.1** | 3600 |
| SS-250-2 | 586689 | 587157.5 | 3600 | 587263 | 587692.4 | 3600 | **586411** | **586921.3** | 3600 |
| SS-250-3 | 586717 | 587004.1 | 3600 | 587257 | 587651.8 | 3600 | **586298** | **586604.9** | 3600 |
| RAND-150-1 | **192369** | **192754.6** | 4000 | 193287 | 193502.5 | 4000 | 192721 | 192866.9 | 4000 |
| RAND-150-2 | **192579** | **192856.1** | 4000 | 193204 | 193462.4 | 4000 | 192603 | 192891.6 | 4000 |
| RAND-150-3 | **192046** | **192583.6** | 4000 | 193026 | 193470.8 | 4000 | 192495 | 192678.3 | 4000 |
| RAND-200-1 | **350321** | **350901.1** | 12000 | 351575 | 352126.3 | 12000 | 350610 | 350954.7 | 12000 |
| RAND-200-2 | 350658 | 350880.5 | 12000 | 351352 | 351934.2 | 12000 | **350328** | **350754.4** | 12000 |
| RAND-200-3 | 350601 | 350980.1 | 12000 | 351449 | 352057.9 | 12000 | **350462** | **350780.5** | 12000 |
| RAND-250-1 | **557278** | **557823.4** | 20000 | 559206 | 559607.1 | 20000 | 557551 | 557894.9 | 20000 |
| RAND-250-2 | **556604** | 557822.8 | 20000 | 558400 | 559512.1 | 20000 | 557122 | **557677.2** | 20000 |
| RAND-250-3 | 557060 | **557762.0** | 20000 | 559029 | 559454.8 | 20000 | **556714** | 557858.1 | 20000 |
| SOAK-150-1 | **206721** | 206900.7 | 4000 | **206721** | 207064.1 | 4000 | **206721** | **206784.7** | 4000 |
| SOAK-150-2 | **206761** | **206894.5** | 4000 | 207076 | 207356.2 | 4000 | 206905 | 207089.4 | 4000 |
| SOAK-150-3 | **206759** | **206839.9** | 4000 | 206914 | 207087.2 | 4000 | **206759** | 206906.0 | 4000 |
| SOAK-200-1 | **369851** | **370220.1** | 12000 | 370701 | 371004.8 | 12000 | 370060 | 370277.0 | 12000 |
| SOAK-200-2 | 369803 | **369932.6** | 12000 | 370056 | 370766.4 | 12000 | **369736** | 370086.0 | 12000 |
| SOAK-200-3 | **369794** | **370010.2** | 12000 | 370035 | 370674.5 | 12000 | 369808 | 370124.0 | 12000 |
| SOAK-250-1 | 581671 | 582364.4 | 20000 | 582829 | 583544.8 | 20000 | **581639** | **582051.7** | 20000 |
| SOAK-250-2 | **581492** | **581903.0** | 20000 | 582817 | 583228.2 | 20000 | 581556 | 582043.6 | 20000 |
| SOAK-250-3 | **581573** | 582323.8 | 20000 | 583196 | 583571.7 | 20000 | 581775 | **582158.7** | 20000 |
| 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 | **14142** | **14142.0** | 3600 | **14142** | **14142.0** | 3600 | **14142** | **14142.0** | 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 |

reveals a $p$-value of $3.41 \times 10^{-10}$ in terms of best objective values, and a $p$-value of $2.46 \times 10^{-10}$ 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 14 instances, leading to a $p$-value of 1) and average objective values (corresponding to a $p$-value of $3.75 \times 10^{-1}$).

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-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).

## Acknowledgements

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 and LigeRO, 2010-2014), Jacques Hadamard Mathematical Foundation (PGMO, 2014-2015), and a Post-doc fellowship from Angers Loire Metropole (France).

## References

[1] Assad, A., & Xu, W., 1992. The quadratic minimum spanning tree problem. Naval Research Logistics 39, 399-417.

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

[3] Benlic, U., & Hao, J.K., 2013a. Breakout local search for the quadratic assignment problem. Applied Mathematics and Computation 219(9), 4800-4815.

[4] Benlic, U., & Hao, J.K., 2013b. Breakout local search for the max-cut problem. Engineering Applications of Artificial Intelligence 26(3), 1162-1173.

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

[6] Buchheim, C., & Klein, L., 2014. Combinatorial optimization with one quadratic term: spanning trees and forests. Discret Applied Mathmatics 177, 34-52.

[7] Burkard, R.E., Karisch, S.E., & Rendl, F., 1997. QAPLIB-a quadratic assignment problem library. Journal of Global Optimization 10, 391-403.

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

[9] Christofides, N., & Benavent, E., 1989. An exact algorithm for the quadratic assignment problem. Operations Research, 37(5), 760-768.

[10] 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, p.168-171.

[11] Cordone, R., & Passeri, G., 2012. Solving the quadratic minimum spanning tree problem. Applied Mathematics and Computation 218, 11597-612.

[12] Fischer, A., & Fischer, F., 2013. Complete description for the spanning tree problem with one linearised quadratic term. Operations Research Letters 41, 701-705.

[13] Fu, Z.H., & Hao, J.K., 2014. Breakout local search for the Steiner tree problem with revenue, budget and hop constraints. European Journal of Operational Research 232(1), 209-220.

[14] Gao, J., & Lu, M., 2005. Fuzzy quadratic minimum spanning tree problem. Applied Mathematics and Computation 164(3), 773-788.

[15] Glover, F., & Laguna, M., 1997. Tabu search. Kluwer Academic Publishers.

[16] Lourenco, H.R., Martin, O., & Stützle, T., 2003. Iterated local search. Handbook of Meta-heuristics, Springer-Verlag.

[17] 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 Transactions 46(4), 414-428.

[18] Maia, S.M.D.M., Goldbarg, E.F.G., & Goldbarg, M.C., 2013. On the biobjective adjacent only quadratic spanning tree problem. Electronic Notes in Discrete Mathematics 41, 535-542.

[19] Maia, S.M.D.M., Goldbarg, E.F.G., & Goldbarg, M.C., 2014. Evolutionary algorithms for the bi-objective adjacent only quadratic spanning tree. International Journal of Innovative Computing and Applications 6(2), 63-72.

[20] Nugent, C.E., Vollman T.E., & Ruml J., 1968. An experimental comparison of techniques for the assignment of facilities to locations. Operations Research 16(1), 150-173.

[21] Öncan, T., & Punnen A.P., 2010. The quadratic minimum spanning tree problem: a lower bounding procedure and an efficient search. Computers & Operations Research 37(10), 1762-1773.

[22] Palubeckis, G., Rubliauskas, D., & Targamadzè, A., 2010. Metaheuristic approaches for the quadratic minimum spanning tree problem. Information Technology and Control 39(4), 257-268.

[23] 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 `http://www.optimization-online.org/DB_FILE/2013/12/4166.pdf`.

[24] Pereira, D.L., Gendreau, M., & Cunha, A.S.D., 2013b. Stronger lower bounds for the quadratic minimum spanning tree problem with adjacency costs. Electronic Notes in Discrete Mathematics, 41, 229-236.

[25] 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.

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

[27] Soak, S.M., Corne, D.W., & Ahn, B.H., 2005. A new evolutionary algorithm for spanning-tree based communication network design. IEICE Transactions on Communication, E88-B(10), 4090-4093.

[28] Soak, S.M., Corne, D.W., & Ahn, B.H., 2006. The edge-window-decoder representation for tree based problems. IEEE Transactions on Evolutionary Computation 10(2), 124-144.

[29] Sundar, S., & Singh, A., 2010. A swarm intelligence approach to the quadratic minimum spanning tree problem. Information Sciences 180(17), 3182-3191.

[30] Xu, W., 1984. The quadratic minimum spanning tree problem and other topics. PhD thesis, University of Maryland, College Park, MD.

[31] Xu, W., 1995. On the quadratic minimum spanning tree problem. In: Japan-China International Workshop on Information Systems, Ashikaga, p. 141-148.

[32] Zhou, G., & Gen, M., 1998. An effective genetic algorithm approach to the quadratic minimum spanning tree problem. Computers & Operations Research 25(3), 229-237.

## A Appendix: 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 A.1–A.5. 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 $\nu_i$) 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 $\chi$ 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 $\lambda_i = \frac{\sum_{j=1}^{|SI|} \lambda_{ij}}{|SI|}$, where $|SI|$ denotes the number of sample instances selected to tune a specified parameter. Intuitively, a larger value of $\nu_i$ or a lower value of $\lambda_i$ 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 $\nu_i$ and $\lambda_i$ 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 A.1): The Friedman test reveals a $p$-value of $9.45 \times 10^{-7}$, 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 A.2): 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 A.3): For parameter $l_{swap}$, no significant difference is detected by the Friedman test (with a $p$-value of $7.96 \times 10^{-2}$). 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 A.4): 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 $2.72 \times 10^{-14}$) and is quite robust on the 14 QAP-QMSTP instances (with a $p$-value of $3.69 \times 10^{-1}$). 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 A.5): 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 $4.05 \times 10^{-8}$) and is rather robust on the 14 QAP-QMSTP instances (with a $p$-value of $4.82 \times 10^{-1}$). 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.

Table A.1
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] | [0.1$n$,0.2$n$] | [0.2$n$,0.3$n$] | [0.3$n$,0.4$n$] | [0.4$n$,0.5$n$] | [0.5$n$,$n$] |
| SS-100-1 | 88808.9 | 88809.9 | 88817.3 | 88800.1 | **88786.9** | 88804.2 | 88800.6 | 88800.5 |
| SS-100-2 | 88824.1 | 88830.8 | 88814.4 | 88851.3 | 88818.5 | 88815.5 | **88810.8** | 88839.6 |
| SS-100-3 | 88625.3 | 88629.7 | **88623.4** | 88671.5 | 88631.0 | 88641.3 | 88629.8 | 88660.1 |
| SS-150-1 | 205196.5 | **205037.9** | 205141.8 | 205113.8 | 205135.9 | 205171.8 | 205095.5 | 205220.8 |
| SS-150-2 | 205306.6 | **205077.1** | 205226.1 | 205122.5 | 205154.6 | 205107.8 | 205179.3 | 205174.9 |
| SS-150-3 | 205178.0 | **204989.4** | 205090.3 | 205069.6 | 205072.1 | 205136.0 | 205021.7 | 205180.4 |
| SS-200-1 | 371036.6 | 371010.6 | 371205.3 | 370930.7 | **370748.8** | 370952.8 | 370968.6 | 371175.8 |
| SS-200-2 | 371040.2 | 371165.3 | 371216.7 | 371150.3 | 371304.0 | 371126.3 | **370972.7** | 371242.3 |
| SS-200-3 | 371209.3 | 371242.9 | 371271.8 | 371244.2 | 371340.3 | 371134.3 | **371087.2** | 371469.4 |
| SS-250-1 | 587177.0 | 587050.9 | 587227.2 | **586820.5** | 587165.4 | 587104.5 | 587356.0 | 587438.2 |
| SS-250-2 | **586690.1** | 587041.8 | 586820.2 | 586947.2 | 586717.9 | 586871.2 | 587035.5 | 587095.4 |
| SS-250-3 | 586711.6 | 586867.6 | 586995.5 | 586849.7 | **586629.6** | 586633.7 | 586873.2 | 587089.8 |
| RAND-150-1 | **192705.9** | 192770.7 | 192766.1 | 192839.6 | 192745.1 | 192801.0 | 192927.1 | 192830.2 |
| RAND-150-2 | 192842.1 | 192964.3 | 192861.0 | 192933.9 | 192946.7 | 192978.7 | 192960.4 | **192779.8** |
| RAND-150-3 | **192698.9** | 192856.8 | 192783.2 | 192757.6 | 192811.5 | 192796.7 | 192870.8 | 192783.0 |
| RAND-200-1 | **351546.4** | 351762.5 | 351594.6 | 351561.7 | 351622.4 | 351588.0 | 352004.7 | 351884.5 |
| RAND-200-2 | 351457.4 | 351918.2 | 351524.9 | 351643.4 | **351282.8** | 351689.8 | 351801.5 | 351836.7 |
| RAND-200-3 | 351658.7 | 351695.6 | 351491.9 | **351238.4** | 351555.0 | 351639.5 | 351937.2 | 351875.1 |
| RAND-250-1 | **558750.8** | 558920.7 | 559327.7 | 558778.0 | 559220.4 | 558873.9 | 559253.8 | 559313.4 |
| RAND-250-2 | 559296.6 | 558854.4 | 559148.9 | **558784.3** | 559222.4 | 559004.5 | 559149.2 | 559286.7 |
| RAND-250-3 | **558678.6** | 558714.0 | 558689.5 | 558876.5 | 559431.2 | 559366.5 | 559266.4 | 559697.3 |
| SOAK-150-1 | 206835.0 | **206830.0** | 206938.0 | 206880.2 | 206917.9 | 206888.7 | 206912.6 | 206965.9 |
| SOAK-150-2 | 207004.8 | 207003.7 | 207030.9 | **206958.8** | 207083.6 | 207076.4 | 206978.5 | 207029.1 |
| SOAK-150-3 | **206877.2** | 206888.4 | 206904.9 | 206896.5 | 206944.4 | 206901.5 | 206910.2 | 206966.8 |
| SOAK-200-1 | **370572.6** | 370679.9 | 370592.9 | 370670.0 | 370752.4 | 370835.3 | 370679.8 | 371032.7 |
| SOAK-200-2 | **370488.9** | 370680.2 | 370658.0 | 370529.8 | 370545.8 | 370667.1 | 370589.9 | 370955.6 |
| SOAK-200-3 | 370799.4 | 370474.3 | 370413.5 | **370076.2** | 370479.1 | 370519.2 | 370615.4 | 370613.2 |
| SOAK-250-1 | 583541.8 | 583073.7 | 583152.0 | 583057.5 | **582963.5** | 583201.5 | 583483.9 | 583895.6 |
| SOAK-250-2 | 583119.7 | 582798.1 | **582740.6** | 583304.2 | 583209.8 | 583049.8 | 583244.6 | 583423.2 |
| SOAK-250-3 | 583315.1 | **582916.5** | 583239.0 | 583226.0 | 583703.4 | 583708.4 | 583157.6 | 583557.7 |
| $\nu_i$ | **9** | 5 | 2 | 5 | 5 | 0 | 3 | 1 |
| $\lambda_i$ | **3.67** | 4.03 | 4.43 | **3.33** | 4.40 | 4.53 | 4.87 | 6.73 |

Table A.2
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] | [0.1$n$,0.2$n$] | [0.2$n$,0.3$n$] | [0.3$n$,0.4$n$] | [0.4$n$,0.5$n$] | [0.5$n$,$n$] |
| SS-100-1 | 88833.6 | 88828.2 | 88847.4 | 88855.8 | **88770.2** | 88818.4 | 88810.9 | 88787.8 |
| SS-100-2 | 88848.0 | 88871.8 | 88865.9 | 88831.3 | 88818.2 | 88834.0 | **88812.9** | 88840.1 |
| SS-100-3 | 88654.6 | 88695.3 | 88742.1 | 88697.3 | 88644.9 | 88654.7 | **88643.6** | 88680.1 |
| SS-150-1 | 205188.3 | 205237.5 | 205714.0 | 205123.2 | 205147.7 | **205060.0** | 205129.2 | 205224.5 |
| SS-150-2 | 205249.6 | 205289.7 | 205726.3 | 205166.4 | 205126.4 | **205124.3** | 205142.3 | 205229.4 |
| SS-150-3 | 205300.0 | 205236.5 | 205513.5 | 205097.4 | 205018.7 | 205064.2 | **204997.1** | 205093.0 |
| SS-200-1 | 371135.4 | 371181.9 | 371880.3 | 370865.4 | 371214.7 | **370778.3** | 370989.9 | 371210.7 |
| SS-200-2 | 371254.9 | 371394.1 | 372056.5 | **370894.1** | 371174.9 | 370903.7 | 371101.4 | 371276.3 |
| SS-200-3 | 371535.3 | 371310.9 | 372048.0 | 370847.7 | 371115.0 | **370823.5** | 371423.7 | 371390.7 |
| SS-250-1 | 587993.4 | 587188.1 | 588617.0 | **586477.4** | 587142.6 | 586609.2 | 587093.8 | 587274.5 |
| SS-250-2 | 587525.4 | 586587.8 | 588199.9 | 586624.3 | **586559.9** | 586877.6 | 587152.4 | 586784.7 |
| SS-250-3 | 587455.5 | 586739.5 | 588323.1 | **586357.3** | 586631.1 | 586610.2 | 587131.9 | 586802.4 |
| RAND-150-1 | 192879.7 | 192843.5 | 193364.0 | 192850.4 | 192862.1 | 192949.8 | 192883.1 | **192781.6** |
| RAND-150-2 | 193222.1 | 192948.8 | 193456.8 | 192872.9 | **192796.1** | 192803.4 | 192953.1 | 192971.6 |
| RAND-150-3 | 192895.7 | 192873.1 | 193404.8 | 192827.6 | **192681.7** | 192718.3 | 192933.4 | 192997.8 |
| RAND-200-1 | 352226.4 | 351540.1 | 353068.5 | 351475.1 | 351499.2 | **351312.0** | 351635.1 | 352184.6 |
| RAND-200-2 | 352177.3 | 351413.9 | 352832.2 | 351380.9 | 351398.4 | **351327.6** | 351699.0 | 351943.5 |
| RAND-200-3 | 352034.9 | 351433.9 | 352856.9 | 351310.6 | 351380.7 | **351277.2** | 351506.7 | 351778.1 |
| RAND-250-1 | 559519.1 | 558925.9 | 561506.1 | **558415.2** | 558756.3 | 558537.4 | 559115.9 | 559378.6 |
| RAND-250-2 | 560533.0 | 559190.7 | 561395.0 | **558398.5** | 558818.0 | 559087.1 | 559183.5 | 559246.1 |
| RAND-250-3 | 560002.2 | 558947.6 | 561363.8 | **558278.9** | 558869.4 | 558543.3 | 559034.8 | 559162.4 |
| SOAK-150-1 | 206993.6 | 206989.0 | 207179.3 | **206878.3** | 206911.5 | 206907.7 | 206918.4 | 206991.4 |
| SOAK-150-2 | 207284.8 | 207160.4 | 207404.7 | 207118.9 | 207147.2 | **206911.8** | 206943.2 | 207198.5 |
| SOAK-150-3 | 206998.3 | 206950.4 | 207168.2 | 206861.7 | 206855.8 | 206841.5 | **206841.0** | 206928.5 |
| SOAK-200-1 | 371093.3 | 370850.4 | 371872.6 | **370505.7** | 370743.7 | 370572.9 | 370589.0 | 371060.1 |
| SOAK-200-2 | 371123.4 | 370831.0 | 371700.9 | 370580.5 | 370594.2 | **370345.1** | 370608.1 | 370667.7 |
| SOAK-200-3 | 370758.5 | 370538.0 | 371414.9 | 370475.8 | 370516.5 | 370456.3 | 370529.0 | **370412.4** |
| SOAK-250-1 | 583595.1 | 583239.0 | 585526.4 | 582996.2 | 583127.4 | **582756.5** | 583228.8 | 583656.8 |
| SOAK-250-2 | 583382.6 | 583298.4 | 585262.2 | **582891.7** | 582959.2 | 582931.5 | 582917.8 | 583314.6 |
| SOAK-250-3 | 583694.4 | 583726.0 | 585592.5 | 583234.1 | 583016.2 | **582930.4** | 583189.3 | 583682.1 |
| $\nu_i$ | 0 | 0 | 0 | 9 | 4 | **11** | 4 | 2 |
| $\lambda_i$ | 6.33 | 5.03 | 7.93 | 2.60 | 2.93 | **2.20** | 3.77 | 5.20 |

Table A.3
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] | [$n$,2$n$] | [2$n$,3$n$] | [3$n$,4$n$] | [4$n$,5$n$] | [5$n$,10$n$] |
| 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 | **14142.0** | **14142.0** | **14142.0** | **14142.0** | **14142.0** | **14142.0** | **14142.0** | **14142.0** |
| 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 |
| $\nu_i$ | 10 | **14** | 11 | **14** | 11 | 11 | 10 | 11 |
| $\lambda_i$ | 2.29 | **1.00** | 2.14 | **1.00** | 1.71 | 2.00 | 2.64 | 2.29 |

Table A.4
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 | 88801.8 | 88807.5 | 88798.8 | 88796.2 | 88808.7 | 88816.9 | **88788.6** | 88826.9 |
| SS-100-2 | **88807.0** | 88833.5 | 88837.8 | 88828.6 | 88834.0 | 88829.4 | 88842.3 | 88865.3 |
| SS-100-3 | 88655.7 | 88649.3 | 88630.1 | 88650.1 | 88628.6 | 88629.2 | **88628.0** | 88635.1 |
| SS-150-1 | 205156.8 | 205140.3 | 205146.7 | **205037.8** | 205227.9 | 205154.8 | 205131.2 | 205247.9 |
| SS-150-2 | 205165.2 | 205243.5 | 205227.7 | **205071.3** | 205177.2 | 205136.9 | 205181.0 | 205157.9 |
| SS-150-3 | 205122.5 | 205079.1 | 205087.5 | **204985.1** | 205106.3 | 205097.4 | 205152.5 | 205166.2 |
| SS-200-1 | 370928.4 | 371009.5 | 370917.6 | 371003.1 | **370812.9** | 371014.2 | 371137.5 | 371145.2 |
| SS-200-2 | 370945.5 | 370838.6 | **370832.8** | 370999.6 | 370971.0 | 371191.3 | 371306.0 | 371414.9 |
| SS-200-3 | **370845.8** | 371220.0 | 371169.7 | 371044.2 | 371200.1 | 371329.1 | 371236.0 | 371737.0 |
| SS-250-1 | **586878.5** | 586889.3 | 587001.1 | 587054.4 | 587168.1 | 587315.0 | 587170.4 | 587510.2 |
| SS-250-2 | 586820.7 | 586792.1 | 586731.9 | **586705.8** | 586867.4 | 587089.2 | 587332.8 | 587554.5 |
| SS-250-3 | **586272.9** | 586714.4 | 586562.8 | 586790.0 | 586619.3 | 586904.3 | 586976.9 | 587376.6 |
| RAND-150-1 | 192713.6 | 192789.8 | 192784.0 | **192676.3** | 192679.7 | 192900.4 | 192921.7 | 193132.5 |
| RAND-150-2 | **192759.3** | 192835.7 | 192887.7 | 192869.1 | 192982.0 | 192923.7 | 193004.8 | 193033.2 |
| RAND-150-3 | 192784.4 | **192676.2** | 192683.4 | 192748.2 | 192838.9 | 192987.6 | 192999.8 | 192923.3 |
| RAND-200-1 | **351245.3** | 351651.6 | 351486.3 | 351399.6 | 351637.2 | 352031.6 | 351778.3 | 351958.4 |
| RAND-200-2 | 351332.4 | 351478.7 | 351547.3 | **351310.9** | 351848.9 | 351824.6 | 351799.2 | 351933.1 |
| RAND-200-3 | 351357.3 | 351783.5 | 351371.4 | 351516.3 | **351281.3** | 351525.1 | 351852.0 | 351914.0 |
| RAND-250-1 | 558795.9 | 559053.2 | **558602.2** | 558818.1 | 558799.9 | 559334.2 | 559450.4 | 559380.3 |
| RAND-250-2 | 558620.9 | 558854.2 | 558931.8 | **558552.1** | 558947.8 | 559194.0 | 559427.0 | 559551.8 |
| RAND-250-3 | 558757.1 | 558892.3 | 558884.8 | **558639.3** | 559085.4 | 559243.4 | 558927.8 | 559469.0 |
| SOAK-150-1 | 206876.9 | 206869.0 | **206811.0** | 206893.3 | 206873.3 | 206919.8 | 206967.4 | 206970.1 |
| SOAK-150-2 | 207060.7 | 206958.0 | 207084.7 | 206905.4 | 207070.5 | **206855.7** | 206958.3 | 206933.0 |
| SOAK-150-3 | 206866.2 | **206845.1** | 206855.9 | 206848.4 | 206922.4 | 206931.4 | 206913.5 | 206949.1 |
| SOAK-200-1 | 370560.0 | 370647.3 | 370570.1 | **370471.2** | 370697.5 | 370753.3 | 370986.7 | 370861.5 |
| SOAK-200-2 | 370481.6 | 370423.2 | **370359.9** | 370528.7 | 370538.2 | 370577.6 | 370613.5 | 370747.1 |
| SOAK-200-3 | 370293.0 | 370436.1 | 370329.0 | **370211.8** | 370322.2 | 370463.6 | 370536.1 | 370816.1 |
| SOAK-250-1 | 582844.9 | 583013.1 | 582866.8 | **582778.0** | 583009.8 | 583351.6 | 583447.5 | 583671.5 |
| SOAK-250-2 | 582884.7 | 582978.5 | **582811.8** | 582861.5 | 582901.2 | 583203.0 | 582969.3 | 583516.8 |
| SOAK-250-3 | 583115.3 | 582970.6 | 583002.5 | **582861.5** | 582887.9 | 583357.2 | 583180.3 | 583673.0 |
| 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 | **14142.0** | **14142.0** | **14142.0** | **14142.0** | **14142.0** | **14142.0** | **14142.0** | **14142.0** |
| 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** |
| $\nu_i$ | 19 | 14 | 17 | **26** | 15 | 15 | 15 | 13 |
| $\lambda_i$ | 2.45 | 3.20 | 2.77 | **2.00** | 3.32 | 4.18 | 4.55 | 5.45 |

Table A.5
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 | 88911.7 | 88873.1 | 88870.6 | 88814.9 | 88853.0 | **88787.2** | 88798.4 | 88793.4 |
| SS-100-2 | 88919.5 | 88908.7 | 88893.6 | 88872.6 | 88836.2 | 88838.7 | 88829.2 | **88810.0** |
| SS-100-3 | 88804.2 | 88717.7 | 88713.8 | 88732.1 | 88713.1 | 88635.7 | 88640.5 | **88619.0** |
| SS-150-1 | 205423.6 | 205210.8 | 205235.3 | 205159.6 | 205377.8 | 205128.8 | 205120.2 | **205080.1** |
| SS-150-2 | 205355.8 | 205291.3 | 205325.9 | 205222.2 | 205158.6 | 205168.5 | 205165.3 | **205084.4** |
| SS-150-3 | 205227.5 | 205334.9 | 205138.1 | 205099.9 | 205113.5 | 205081.0 | 205120.0 | **204957.5** |
| SS-200-1 | 371401.8 | 371087.6 | 371049.3 | 371058.6 | 371077.5 | 371007.2 | **370952.7** | 371058.6 |
| SS-200-2 | 371278.0 | 371308.1 | 371089.0 | 370999.1 | 371105.2 | **370982.6** | 371021.9 | 371179.9 |
| SS-200-3 | 371386.7 | 371233.2 | 371274.6 | **371009.0** | 371017.3 | 371165.8 | 371086.1 | 371220.7 |
| SS-250-1 | 586970.1 | 586929.7 | 586971.8 | **586835.3** | 587058.2 | 587211.0 | 587184.3 | 587128.7 |
| SS-250-2 | 587027.8 | 586721.0 | 587094.7 | **586508.3** | 587088.0 | 586913.8 | 587037.6 | 587071.2 |
| SS-250-3 | 587150.0 | 586687.8 | 586748.5 | **586530.9** | 587086.1 | 586782.6 | 586753.2 | 586764.6 |
| RAND-150-1 | 193159.3 | 193067.1 | 192884.4 | 192835.1 | 192847.1 | **192734.8** | 192812.7 | 192890.5 |
| RAND-150-2 | 193177.8 | 193183.9 | 193119.0 | 192918.8 | 192937.2 | 192914.3 | **192816.5** | 192994.7 |
| RAND-150-3 | 193021.5 | 192958.3 | 192930.4 | 192872.8 | 192733.5 | 192728.7 | **192705.4** | 192995.1 |
| RAND-200-1 | 351631.5 | 351553.9 | 351711.8 | 351695.4 | 351553.8 | 351555.1 | **351247.1** | 351724.7 |
| RAND-200-2 | 351763.7 | 351715.7 | 351631.7 | 351723.5 | 351627.7 | **351399.6** | 351552.1 | 351831.4 |
| RAND-200-3 | 351613.4 | 351706.6 | 351606.3 | 351655.4 | 351511.9 | **351434.2** | 351595.5 | 351760.9 |
| RAND-250-1 | 559418.3 | 559187.9 | 558842.1 | **558833.8** | 558856.7 | 558927.4 | 558930.4 | 559381.5 |
| RAND-250-2 | 559280.0 | 559285.0 | 559100.4 | **558877.6** | 558930.2 | 559314.8 | 559189.1 | 559377.7 |
| RAND-250-3 | 558997.6 | 558981.3 | **558721.1** | 558761.8 | 559104.2 | 559137.9 | 559096.0 | 559364.4 |
| SOAK-150-1 | 207093.9 | 207026.2 | 207060.8 | 206955.0 | 207034.5 | **206855.7** | 206889.5 | 206885.8 |
| SOAK-150-2 | 207401.5 | 207261.5 | 207234.8 | 207098.2 | 207092.3 | 207003.9 | 206966.8 | **206943.6** |
| SOAK-150-3 | 206980.5 | 206869.8 | **206845.5** | 206867.8 | 206865.9 | 206893.2 | 206847.2 | 206871.0 |
| SOAK-200-1 | 370901.0 | 370753.8 | 370875.1 | 370559.3 | 370754.7 | 370837.9 | 370657.7 | **370514.2** |
| SOAK-200-2 | 371060.1 | 370922.0 | 370775.3 | 370693.6 | **370421.7** | 370515.6 | 370761.3 | 370487.8 |
| SOAK-200-3 | 370661.1 | 370710.8 | 370525.2 | 370334.9 | 370585.5 | 370384.9 | 370518.4 | **370117.6** |
| SOAK-250-1 | 583780.1 | **582928.5** | 583481.8 | 583162.7 | 583330.8 | 583164.3 | 583416.6 | 583107.5 |
| SOAK-250-2 | 583316.3 | 583413.6 | 583167.5 | 583001.1 | 582855.4 | **582855.1** | 582988.1 | 582938.2 |
| SOAK-250-3 | 583525.5 | 583852.2 | 583409.2 | 583737.2 | 583538.0 | 583255.8 | **583222.3** | 583344.2 |
| 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 | **14142.0** | **14142.0** | **14142.0** | **14142.0** | **14142.0** | **14142.0** | **14142.0** | **14142.0** |
| 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** |
| $\nu_i$ | 14 | 14 | 16 | 20 | 15 | **21** | 19 | **22** |
| $\lambda_i$ | 5.07 | 4.34 | 3.75 | 2.73 | 3.14 | **2.57** | **2.48** | 3.16 |