Tabu Search with Strategic Oscillation for the Quadratic Minimum Spanning Tree

Manuel Lozano
Department of Computer Science and Artificial Intelligence,
University of Granada, Granada 18071, Spain, lozano@decsai.ugr.es

Fred Glover
OptTek Systems, Boulder (Co), USA, glover@opttek.com

Carlos García-Martínez
Department of Computing and Numerical Analysis,
University of Córdoba, Córdoba 14071, Spain, cgarcia@uco.es

Francisco Javier Rodríguez
Department of Computer Science and Artificial Intelligence,
University of Granada, Granada 18071, Spain, fjrodriguez@decsai.ugr.es

Rafael Martí
Department of Statistics and Operations Research
University of Valencia, Valencia, Spain, rmarti@uv.es

Abstract

The quadratic minimum spanning tree problem consists of determining a spanning tree that minimizes the sum of costs of the edges and pairs of edges in the tree. Many algorithms and methods have been proposed for this hard combinatorial problem, including several highly sophisticated metaheuristics. In this paper we present a simple tabu search (TS) for this problem that incorporates strategic oscillation (SO) by alternating between constructive and destructive phases. We show commonalties shared by this strategy and the more recently introduced methodology called iterated greedy search, and identify implications of their differences regarding the use of memory structures. Extensive computational experiments reveal that the proposed SO algorithm with embedded TS is highly effective for solving complex instances of the problem as compared to the best metaheuristics in the literature. We also introduce a hybrid method that proves similarly effective for problem instances that are both simple and complex.

Keywords. Tabu search, strategic oscillation, adaptive memory programming, Quadratic minimum spanning tree, iterated greedy.

1 Introduction

The Quadratic Minimum Spanning Tree Problem (QMSTP) has been widely studied in the literature due to its applications in a wide variety of settings, including transportation, telecommunication, irrigation, and energy distribution. The problem appears, for example, when transferring oil from one pipe to another in a situation where the cost depends on the type of interface between two pipes. The same pairwise interaction effect arises in the connection of aboveground and underground cables in a road network with turn penalties [23, 28].

We may define the QMSTP as follows. Let $G = (V, E)$ be an undirected graph where $V = \{v_1, \cdots, v_n\}$ is the vertex set and $E = \{e_1, \cdots, e_m\}$ is the edge set. Consider that each edge and each pair of edges has an associated cost. In mathematical terms, we have two cost functions: $w : E \rightarrow \Re^+$ and $c : (E \times E) - \{(e, e), \forall e \in E\} \rightarrow \Re^+$ where as in previous approaches [2, 19], we assume that $c(e_i, e_j) = c(e_j, e_i)$ for $i, j = 1, ..., m$. The QMSTP consists of finding a spanning tree T of G with edge set $\xi(T) \subseteq E$ that minimizes:

$$\sum_{e_i \in \xi(T)} \sum_{\substack{e_j \in \xi(T) \\ e_i \neq e_j}} c(e_i, e_j) + \sum_{e \in \xi(T)} w(e).$$

The QMSTP is an extension of the well-known minimum spanning tree problem, where in addition to edge costs, we have costs associated with pairs of edges. The problem was first introduced by Assad and Xu [1, 25], showing that it is NP-hard.

This paper has two objectives: the primary objective is to investigate the strategic oscillation proposal to alternate between constructive and destructive phases as a basis for creating a competitive method for the QMSTP, and the secondary objective is to compare memory-less with memory based designs. The remainder of this paper is organized as follows. In Section 2, we give a background of Tabu Search (TS) and Strategic Oscillation (SO) that sets the stage for the later specific algorithmic design we employ. We also refer to the more recent Iterated Greedy (IG) approach which has some resemblances to strategic oscillation, and which provides a basis for subsequent comparative testing. Section 3 gives an overview of particular metaheuristics that have previously been applied to the QMSTP. Section 4 describes our proposed methods based on TS, SO and IG. In Section 5, we present empirical studies, which are designed to: 1) analyze the influence of the parameters and settings of our methods, 2) compare the different designs, paying special attention to the influence of memory structures, 3) obtain an algorithm with a robust performance across test problems with different characteristics, and 4) compare its results with those of the best approaches from the literature. We finish with the associated conclusions in Section 6 and our proposals for further extensions.

2 Background of Tabu Search and Strategic Oscillation

Tabu Search is a meta-heuristic that guides a local heuristic search procedure to explore the solution space beyond local optimality. One of the main components of Tabu Search is its use of adaptive memory, which creates a highly flexible search behavior. Memory-based strategies which are the hallmark of tabu search approaches, are founded on a quest for "integrating principles," by which alternative forms of memory are appropriately combined with effective strategies for exploiting them. The adaptive memory feature of TS allows the implementation of procedures that are capable of searching the solution space economically and effectively. Since local choices are guided by information collected during the search, TS contrasts with memoryless designs that heavily rely on semi-random processes that implement a form of sampling.

The structure of a neighborhood in tabu search goes beyond that used in local search by embracing the types of moves used in constructive and destructive processes (where the foundations for such moves are accordingly called constructive neighborhoods and destructive neighborhoods). Following basic TS principles, memory structures can be implemented within a constructive process to favor (or avoid) the inclusion of certain elements in the solution previously identified as attractive (or unattractive). Such expanded uses of the neighborhood concept reinforce a fundamental perspective of TS, which is to define neighborhoods in dynamic ways that can include serial or simultaneous consideration of multiple types of moves.

This dynamic neighborhood approach applies not only to the types of neighborhoods used in "solution improvement methods" (sometimes called "local search methods") but also applies to constructive neighborhoods used in building solutions from scratch - as opposed to transitioning from one solution to another. Although it is commonplace in the metaheuristic literature to restrict the word "neighborhood" to refer solely to transitions between solutions as embodied in improvement methods, constructive neighborhoods have been proposed as an important ingredient of search processes from the very beginning of the TS methodology, as documented by Glover and Laguna [7]. Nevertheless, tabu search methods for exploiting constructive neighborhoods have rarely been applied in computational studies.

Our tabu search approach for the QMSTP is additionally based on the strategic oscillation methodology [7, 8]. Strategic oscillation (SO) is closely linked to the origins of tabu search, and operates by orienting moves in relation to a critical level, as identified by a stage of construction. In particular, we consider a constructive/destructive type of strategic oscillation, where constructive steps "add" elements and destructive steps "drop" elements. As described in Chapter 4 of [8], the alternation of constructive with destructive processes, which strategically dismantle and then rebuild successive trial solutions, affords an enhancement of such traditional constructive procedures.

More recently, constructive and destructive neighborhoods have been applied within a simplified and effective method known as Iterated Greedy (IG) [10], which generates a sequence of solutions by iterating over a greedy constructive heuristic using two main phases: *destruction* and *construction*. IG is a memory-less metaheuristic easy to implement that has exhibited state-of-the-art performance in some settings (see for example [4, 12, 21]). We identify links between this more recent constructive/destructive approach and strategic oscillation by first proposing an adaptation of the IG methodology to the QMSTP and then extending it to include short term memory structures to create a tabu search approach based on strategic oscillation.

3 Previous Metaheuristics Applied to the QMSTP

Although exact procedures, including branch and bound [1, 25] and Lagrangian relaxation [14], have been applied to the QMSTP, their success has been chiefly limited to problems that may be classified as small, containing from 6 to 15 vertices. By contrast, metaheuristics have been shown capable of finding high quality solutions to QMSTP problems over a wide range of sizes, from small to significantly larger than those handled by exact methods. Early genetic algorithms (GAs) implementations were made by Zhou and Gen [28], which proposed a GA based on the Prüfer number to encode a spanning tree. The algorithm was tested on QMSTP instances with up to 50 vertices and proved superior to two greedy algorithms proposed in [1, 25]. More recently, Soak, Corne, and Ahn [22] developed another GA approach that employed an edge-window-decoder encoding, (in which a tree is represented as a linear string of node identifiers, and these in turn are interpreted as a set of edges). This GA implementation was demonstrated to perform much better than GAs based on other encodings (including the Prüfer number representation) on Euclidean instances containing between 50 and 100 nodes. In related work, Gao and Lu [5] proposed another GA to address a QMSTP variant, the fuzzy QMSTP, which was formulated as an expected value model with chance-constrained programming and dependent-chance constrained programming. The authors suggested the use of Prüfer number representation for the GA.

The Artificial bee colony method is another bio-inspired metaheuristic that has been a tool of choice for dealing with the QMSTP [23], involving a swarm intelligence technique based on the analogy to the intelligent foraging behavior of honey bees. To investigate the effectiveness of this approach, the authors carried out experiments with problem instances of up to $n = 250$, finding that artificial bee colony obtained better solutions than those achieved by the GA-based methods of Zhou and Gen [28] and Soak et al [22].

Recently, Cordone and Passeri [2] proposed a tabu search implementation for the QMSTP based on exchanges. At each iteration, the method adds a new edge to the current spanning tree and removes one of the edges from the resulting loop. The tabu mechanism forbids recently removed edges to be added into the solution and recently added edges to be removed for a certain number of iterations. The algorithm was tested over a benchmark set of randomly generated instances with up to 30 vertices. However, no comparison with previous metaheuristics is presented.

In the context of adaptive memory programming (the use of memory structures within metaheuristics), Öncan and Punnen [14] presented a local search algorithm with tabu thresholding. The algorithm was tested on different benchmark problems with sizes ranging from 6 to 100 vertices. Empirical results evidenced that the method improves upon the greedy algorithms presented in [1, 25]. Finally, the most recent TS proposal is the highly effective Iterated Tabu Search approach, proposed by Palubeckis [19] using two phases: solution perturbation and TS. The method to perturb a solution (current spanning tree) relies on randomization: an edge for removal and a non-tree edge for replacement are randomly selected from a candidate list. At each iteration, the TS method examines all feasible exchange moves (replacement of an edge in the tree) and the best admissible non-tabu move is performed. The removed and added edges are labeled tabu for $\frac{m}{4}$ iterations. The TS method is run for a number of iterations taken from the interval $[I_{min}, I_{max}]$. Computational results for problem instances with up to

50 vertices indicated that its performance (in terms of both solution quality and computational time) surpasses those of the other metaheuristics included in their comparison, which consisted of multistart simulated annealing, GA, and GA with local search.

4 New Metaheuristic Adaptations

In this section, we explore the adaptation of the TS and IG methodologies to obtain high quality solutions to the QMSTP. We first describe in Section 4.1 the greedy constructive and destructive algorithms that will be applied in both methods. Then, in Section 4.2, we provide details of the improvement methods proposed in the literature for the QMSTP, which will be employed as the local search phase of our IG for the QMSTP. Moreover, we describe the short term memory structure that added to the local search creates the tabu search improvement method of our TS for the QMSTP. Section 4.3 gives the entire IG method, showing how its different elements interact; and finally, Section 4.4 completes the development by providing the general overview of the TS algorithm with the Strategic Oscillation strategy.

4.1 Constructive and Destructive Methods

We begin by introducing two greedy constructive algorithms, C1 and C2, and a destructive one, D, which may be used to build feasible solutions for the QMSTP. The two constructive methods accomplish their task by adding, at each construction step, exactly one edge to a current partial solution, which is a forest of G (disjoint union of trees). In order to describe C1 and C2, we firstly define the contribution of an edge $e_i \in E$ to the total cost of a forest F with edge set $\xi(F) \subseteq E$ as:

$$Q(e_i, F) = w(e_i) + \sum_{e_j \in \xi(F)} c(e_i, e_j),\ i = 1, ..., m.$$

C1 is an iterative process that is similar to Kruskal's algorithm [11]. At each iteration, the algorithm considers the set $S \subseteq E \setminus \xi(F)$ of all edges that can be used to feasibly augment the current forest F (i.e., those edges whose inclusion would not result in a cycle) and adds the feasible edge with the minimum contribution value across all elements in S, e_{min}. We can easily update $Q(e_i, F)$ for each e_i, $i = 1, ..., m$ by adding the value $c(e_i, e_{min})$ to it. We should point out that given a spanning tree T, the objective function value z can be obtained from the Q values with the expression:

$$z = \sum_{e_i \in \xi(T)} Q(e_i, T).$$

The greedy constructive procedure C2 is based on the sequential fixing method proposed by Assad and Xu [1, 25]. It proceeds like C1 but employing the following greedy function to guide the iterative selection of edges (again, the lowest values are preferable):

$$Q_2(e_i, F) = Q(e_i, F) + \frac{n_1}{m_1} \cdot \sum_{\substack{e_j \in S \\ i \neq j}} c(e_i, e_j),\ i = 1, ..., m.$$

where $m_1 = |S| - 1$ and $n_1 = n - 1 - |\xi(F)|$. Note that Q_2 extends Q by further considering interaction costs with the candidates edges for the tree in S (in this way, C2 is more complicated than C1).

The greedy destructive algorithm is based on the reverse-delete algorithm (reverse version of Kruskal's algorithm). The algorithm starts with the original graph G and then it removes one edge at a time until there are only $n - 1$ selected edges remaining. Specifically, it deletes the edge e with the maximum contribution to the current graph that does not disconnect the graph, e_{max}. In this case, the Q values for each e_i ($i = 1, ..., m$) are updated by subtracting $c(e_i, e_{max})$ from them.

4.2 Local Search Methods

In this section, we describe three improvement methods, based on exchanges, previously proposed for the QMSTP. Given a solution T, we exchange an edge $e_i \in \xi(T)$ with an edge $e_j \in E \backslash \xi(T)$ resulting on a tree T' with $\xi(T') = \xi(T) \cup \{e_j\} \backslash \{e_i\}$. Note that when we delete e_i from T, the tree is partitioned into two components and therefore, the edge e_j is selected from the set of edges connecting these two components of T in order to obtain a new feasible solution (tree T'). We observe that we can efficiently evaluate an exchange move without recomputing the objective function from scratch. Let z be the value of the objective function of solution T. The value z' of the new solution T' may be directly computed as:

$$z' = z - 2 \cdot Q(e_i, T) + w(e_i) + 2 \cdot Q(e_j, T) - w(e_j) - 2 \cdot c(e_i, e_j).$$

In this way, the move value of the solutions in the neighborhood of a given solution can be quickly evaluated, and once a move is chosen, the values $Q(e_i, T')$, $k = 1, ..., n$, may be calculated as follows:

$$Q(e_k, T') = Q(e_k, T) - c(e_k, e_i) + c(e_k, e_j).$$

Based on the neighborhood defined by the exchanges above, we have studied three local search methods:

- *Best-improvement local search* (BL). This is the classical local search method in which at each iteration we explore the entire neighborhood and perform the best move. In other words, the *best choice* strategy selects the move with the largest move value among all the moves in the neighborhood. If it improves the current solution, we apply it and examine in the next iteration the neighborhood of the new solution. Otherwise, the method stops.

- *First-improvement local search* (FL) [19]. This method implements the so-called *first choice* strategy that scans the moves in the neighborhood in search for the first exchange yielding an improvement in the objective function. At each iteration, an edge is randomly selected from the current solution ($e_i \in \xi(T)$) to be removed from it. Then, we examine the edges out of the solution ($e_j \in E \backslash \xi(T)$) that can be added to $\xi(T) \backslash \{e_i\}$, producing a new solution (spanning tree). We randomly select an edge e_j and evaluate the associated move. If it is an improving move, we perform it; otherwise, we randomly select a new edge e_j until we find an improving move or all the e_j edges have been explored. In the latter case we simply proceed to the next iteration in which a new edge e_i is randomly selected for removal. The procedure terminates when no further improvement is possible.

- *ABC local search* (AL) [23]. This is a hybrid method between the BL and the FL described above. At each iteration, this local search randomly selects an edge from the current solution ($e_i \in \xi(T)$) to be removed from it. Then, it evaluates all the possible edges that can be added to $\xi(T) \backslash \{e_i\}$ and performs the best one if it improves the current solution. The procedure terminates when no further improvement is possible.

These three methods represent typical implementations of a local search process. They basically resort to randomization or exhaustive exploration to select a move in the neighborhood of a solution. Based on AL, we now propose a *short term memory tabu local search* (TLS) in which we add a memory structure and apply candidate list strategies for move selection.

For each edge in the solution, $e_i \in \xi(T)$, we can compute a measure, $m(e_i)$ of its contribution to the solution value:

$$m(e_i) = w(e_i) + \sum_{\substack{e_j \in \xi(T) \\ e_j \neq e_i}} c(e_i, e_j).$$

These measures are mapped onto probabilities for move selection. Specifically, at each iteration of TLS, we probabilistically select an edge in the solution to be removed. The probabilities are computed from the measure above, where the larger the contribution the larger the probability to be selected (the probability of selecting edge e_i is proportional to its contribution $m(e_i)$). Then, as in AL, we evaluate all the possible edges that can be added to $\xi(T)\backslash\{e_i\}$, but here we select the best one that is not tabu, regardless of whether it improves the current solution (i.e., the move is executed even when the move value is not positive, resulting in a deterioration of the current objective function value). Then, we update the measure values and perform a new iteration. The moved edge become tabu for $TabuTenure$ iterations, and therefore they cannot be selected for addition or removal during this time.

To implement our memory structure, we employ a one-dimensional array $tabu(e)$, initially set to $-TabuTenure$ to permit initial selections, to store the iteration number when edge e is moved. That is, if edge e_i is removed from the solution and e_j is added to the solution at iteration $iter$, then $tabu(e_i) = tabu(e_j) = iter$. Then, in a subsequent iteration $iter_2$, we say that an edge e is tabu (and cannot be added to or removed from the solution) if

$$iter_2 - tabu(e) < TabuTenure.$$

Note that although we use the same tenure value for both edges, an interesting variant is to use a different tenure value for e_i than for e_j. In our experiments, however, we determined that the effect of using different tenure values does not justify the increase in complexity related to calibrating an additional search parameter.

4.3 Iterated Greedy

From an initial solution the IG method alternates between destructive and constructive phases just as strategic oscillation does. During the destructive phase, some solution components are removed from a previously constructed solution. The construction procedure then applies a greedy constructive heuristic to reconstruct a solution. Once a newly reconstructed solution has been obtained, an acceptance criterion is applied to decide whether it will replace the incumbent solution. Additionally, an optional local search phase for improving the reconstructed solution can be applied for improved outcomes.

An outline of the proposed IG is depicted in Figure 1. It starts from a complete initial solution T (Initialise(); Step 1) and then iterates through a main loop which first generates a partial candidate solution F by removing a fixed number of edges from the complete candidate solution T (Destruction-phase(T, n_d); Step 4) and next reconstructs a complete solution T_c starting with F (Construction-phase(F); Step 5). In the local search phase (Local-Search-phase(T_c); Step 6), an improvement procedure is performed in order to find better solutions near the reconstructed solution. Before continuing with the next loop, an acceptance criterion (AcceptCriterion(T, T_i); Step 10) decides whether the solution returned by the local search procedure, T_i, becomes the new incumbent solution. The process iterates through these phases until a computation limit t_{max} is reached. The best solution, T_b, generated during the iterative process is kept to provide the final result.

In the algorithm in Figure 1 we apply the greedy destructive algorithm D (see Section 4.1) to obtain the initial solution Initialise(). Then we can apply either C1 or C2 constructive algorithms (Section 4.1) to implement the Construction-phase(F). These methods insert n_d edges into the forest resulting from the destruction phase, obtaining a feasible spanning tree. The Destruction-phase(T, n_d) removes n_d (a parameter of the algorithm) edges from the current solution. These edges are selected at random as in many previous IG algorithms [20, 21, 26]. Finally, we can apply any of the three local search algorithms described in Section 4.2 as the Local-search-phase(T_c).

We have considered two different following acceptance criteria in the scheme shown in Figure 1:

```
   Input: G, t_max, n_d
   Output: T_b
1  T ← Initialise();
2  T_b ← T;
3  while t_max is not reached do
4  |   F ← Destruction-phase(T, n_d);
5  |   T_c ← Construction-phase(F);
6  |   T_i ← Local-Search-phase(T_c);
7  |   if T_i is better than T_b then
8  |   |   T_b ← T_i;
9  |   end
10 |   if AcceptCriterion(T, T_i) then
11 |   |   T ← T_i;
12 |   end
13 end
```

Figure 1: Iterated greedy pseudocode

- *'Replace if better' acceptance criterion* (RB). The new solution is accepted only if it provides a better objective function value [27].
- *'Random walk' acceptance criterion* (RW). An IG algorithm using the RB acceptance criterion may lead to stagnation situations of the search due to insufficient diversification [20]. At the opposite extreme is the random walk acceptance criterion, which always applies the destruction phase to the most recently visited solution, irrespective of its objective function value. This criterion clearly favors diversification over intensification, because it promotes a stochastic search in the space of local optima.

4.4 Tabu Search with Strategic Oscillation

Our SO approach for the QMSTP implements a 1-sided oscillation that does not cross into infeasible space by adding more edges than necessary (which could be an interesting design to test). It thus differs from instances of SO that are organized relative to feasibility and infeasibility, involving a 2-sided oscillation that crosses that boundary.

An outline of the proposed SO is depicted in Figure 2. It starts from a complete initial solution T and then iterates through a main loop which first generates a solution T_i by a combination of a constructive procedure and a memory structure. Once a fully constructed solution is obtained (and a limiting computational time has not yet been reached), we pass to the Destructive Phase in which edges are removed from the complete candidate solution T_i. Once a Turn Around point is reached, the method goes to the next Constructive Phase.

Note that in the SO algorithm the meaning of a best move depends not only on problem context but on whether a Constructive or Destructive phase is being employed. The Turn Around point itself may oscillate by creating partial solutions containing different numbers of elements. The Constructive and Destructive processes can be interrupted at any point to apply an Improvement Process that may or may not include memory structures. Often Improvement Processes are applied only at the conclusion of a Constructive Phase, when a complete solution is obtained, but they may be applied at other points as well. Sometimes applying them earlier in a constructive process, for example, can remove deficient features that would otherwise be inherited by later construction stages and that would create undesirable solutions.

In our SO method for the QMSTP we consider a constructive phase with two parts. In the first one, we simply apply a greedy algorithm (C1 or C2, described in Section 4.1) but some elements (those labeled as tabu) are not allowed to become part of the constructed solutions. In the second part, we apply a short term tabu search (TLS) to improve the solution as described

```
   Input: G, t_max, n_tabu
   Output: T_b
1  T ← Initialise();
2  T_b ← T;
3  while t_max is not reached do
       // Destructive Phase
4      T_i ← T;
5      while Current solution T_i has not reached a Turn
       Around point do
6          Select a best element to drop from T_i subject to (po-
           tential) tabu restrictions.
7          Drop it from the (partial) solution T_i
8      end
       // Constructive Phase
9      while Current solution T_i is not complete do
10         Select a best element subject to tabu restrictions.
11         Add it to the partial solution
12     end
13     T'_i ← TLS Improvement Process(T_i, n_tabu);
14     if T'_i is better than T_b then
15         T_b ← T_i;
16     end
17     if AcceptCriterion(T, T'_i) then
18         T ← T'_i;
19     end
20 end
```

Figure 2: Strategic Oscillation pseudocode

in Section 4.2. In line with this, our destructive phase for the QMSTP also incorporates memory structures. As in the IG algorithm we randomly remove n_d elements from the current solution, but now, a number n_{tabu} of these elements, $n_{tabu} \leq n_d$, removed by the destruction phase, are labeled tabu and thus not allowed to be added to the solution in the following constructive phase. This strategy attempts to avoid looping over already visited solutions, allowing the algorithm to advance towards diversified promising solutions. Note that the *tabu tenure* in our memory structure is one entire iteration (destructive + constructive phases). This memory structure may become decisive to ensure that SO performs an effective search when dealing with complex problems (diversification is a key factor in this process). The level of diversity induced by this technique is controlled by the n_{tabu} parameter.

As customary in tabu search, we have implemented an *aspiration criterion* to overrule the tabu status under certain circumstances. Note that specific situations may arise where the construction of a feasible tree is not possible given the constraints imposed by this strategy. When such an outcome is detected, the reference to tabu status is directly disabled ($n_{tabu} = 0$) during the current cycle to allow the creation of the new solution. (A common alternative is to use a so-called "aspiration by default" which removes the tabu status from a certain number of the least tabu elements, measured by reference to their unexpired tabu tenures.)

5 Computational Experiments

This section describes the computational experiments that we performed to assess the performance of the IG and SO models presented in the previous sections. Firstly, we detail the experimental setup and the statistical methods applied (Section 5.1), then, we analyze the

results obtained from different experimental studies carried out with these algorithms. Our aim is: 1) to analyze the influence of the parameters and settings associated with IG, which is a memory-less based algorithm (Section 5.2), 2) to show the benefits of the use of memory structures in SO and TLS (Section 5.3), 3) to combine SO, IG, and ITS with the aim of obtaining a hybrid metaheuristic being able to show a robust operation for test problems with different characteristics (Section 5.4), and 4) to compare the results of the hybrid algorithm with those of other metaheuristic approaches for the QMSTP from the literature (Section 5.5).

5.1 Experimental Setup

The codes of all the studied algorithms have been implemented in C and the source code has been compiled with gcc 4.6. The experiments were conducted on a computer with a 3.2 GHz Intel® Core™ i7 processor with 12 GB of RAM running Fedora™ Linux V15. We considered three groups of benchmark instances for our experiments (in total constituting 83 instances), which are described below.

- The first set of benchmarks, hence forth denoted by CP, is composed of 36 instances, ranging in size from 40 to 50 vertices, introduced by Cordone and Passeri (they are publicly available[1]). For experimentation with these instances, the cutoff time for each run was 10 seconds.
- Öncan and Punnen [14] present a transformation scheme to obtain QMSTP instances from *quadratic assignment* problem (QAP) instances. They demonstrated that the optimum value of a QMSTP instance obtained with this transformation is equal to the optimum value of the corresponding QAP instance. Particularly, the authors offered two instance sets by transforming 15 QAP instances (from $n = 24$ to 60) by Nugent et al. [13] and 14 QAP instances (from $n = 24$ to 50) by Christofides and Benavent [3] (whose optimal solutions are known). They will be denoted by NUG and CHR, respectively. The experiments reported in [14] showed that these instances are particularly challenging for QMSTP algorithms. A time limit of 1000 seconds was assigned for these problems.
- The last group consists of two sets of large instances: RAND and SOAK. They were generated exactly in the same manner as in [28] and [22], respectively. All the instances in the RAND set represent complete graphs with integer edge costs uniformly distributed in $[1, 100]$. The costs between edges are also integers and uniformly distributed in $[1, 20]$. In the SOAK instances, nodes are distributed uniformly at random on a 500×500 grid. The edge costs are the integer Euclidean distance between these points. The cost between edges are uniformly distributed between $[1, 20]$. For each value of $n \in \{150, 200, 250\}$, there are 3 instances, leading to a total of 9 instances for each set. All methods were stopped using a time limit that varied according to problem size (400, 1200, and 2000 seconds for problems from $n = 150$ to 250, respectively).

Non-parametric tests [6] have been used to compare the results of the different optimization algorithms under consideration. The only condition to be fulfilled for the use of non-parametric tests is that the algorithms to be compared should have been tested under the same conditions (that is, the same set of problem instances, the same stopping conditions, the same number of runs, etc). In the first place, *average ranking* for each algorithm is firstly computed according to *Friedman's* test. This measure is obtained by computing, for each problem instance, the ranking r_a of the observed results for algorithm a assigning to the best of them the ranking 1, and to the worst the ranking $|A|$ (A is the set of algorithms). Then, an average measure is obtained from the rankings of this algorithm for all test problems. For example, if a certain algorithm achieves rankings 1, 3, 1, 4, and 2, on five problem instances, the average ranking is $\frac{1+3+1+4+2}{5} = \frac{11}{5}$. Note that the lower an average ranking is, the better its associated algorithm

[1] http://homes.dsi.unimi.it/∼cordone/research/qmst.html

is. We have considered two alternative methods based on non-parametric tests to analyze the experimental results:

- The first method is the application of the *Iman and Davenport* test and the *Holm* method as a post hoc procedure. The first test may be used to see whether there are significant statistical differences among the compared algorithms. If differences are detected, then Holm's test is employed to compare the best algorithm (*control algorithm*) against the remaining ones.
- The second method is the utilization of the *Wilcoxon matched-pairs signed-ranks* test. With this test, the results of two algorithms may be directly compared. In statistical terms, this test answers the question: Do the two samples represent two different populations? In the context of algorithms' comparison the Wilcoxon test determines if the results of two methods are significantly different.

5.2 Study of the Memory-less Based Method: IG

In this section, we investigate the effect of the different parameters and strategies applied in IG and their interactions. In particular, we consider:

- The acceptance criteria: RB and RW.
- The memory-less based improvement methods: FL, BL, and AL.
- The number of removed solution components n_d=$0.2n$, $0.4n$, $0.6n$, and $0.8n$.

With the purpose of fine-tuning this method, we employed two types of training problem instances: 1) the 14 CHR instances and 2) 14 *large instances* from the SOAK and RAND sets with $n = 150$ to $n = 200$. All the IG variants in this experiment apply C1 in the construction phase (we will study the application of C2 in the next experiment). The IG methods were stopped using a time limit of 360 seconds and one run was performed for each problem instance.

In each experiment, we compute for each instance the overall best solution value, *BestValue*, obtained by the execution of all methods under consideration. Then, for each method, we compute the relative deviation between the best solution value found by the method and the *BestValue*. In Table 1, we report the average of this relative deviation (*Dev*) across all the instances considered in each particular experiment and the number of instances (*#Best*) for which the value of the best solution obtained by a given method matches *BestValue*. We also show the average *rankings* (computed by the Friedman test) obtained by these IG variants (Section 5.1).

In order to analyze the results, we have applied Iman-Davenport's test (the level of significance considered was 0.05). We have observed the existence of significant differences among the rankings (the statistical values, 79.41 and 160.16, are greater than the critical ones, 1.56 and 1.55, respectively). Then, we have compared the best ranked algorithm for each training set (control algorithm), [RW, AL, $0.4n$] and [RW, AL, $0.6n$], with the other IG versions, by means of Holm's test (a post hoc statistical analysis) with $p = 0.05$. The last column in Table 1 indicates whether Holm's test finds statistical differences between the control algorithm and the corresponding algorithm.

We can draw the following conclusions from Table 1:

- The acceptance criterion has the largest impact on the IG performance. Unexpectedly, the best outcomes (rankings) are obtained when the RW acceptance criterion is used (Holm's test confirms that its superiority is statistically significant). As a matter of fact, it is unusual to find IG models in the literature employing this strategy (most of them show a bias towards maintaining high quality solutions). Thus, we may remark that the combination of the diversification provided by RW and the intensification of the improvement procedure benefits the IG performance.

IG	CHR Instances				IG	LARGE Instances			
(AC, LS, n_d)	*Av. Ran.*	*Dev*	*#Best*	Holm	(AC, LS, n_d)	*Av. Ran.*	*Dev*	*#Best*	Holm
RB FL $0.2n$	21.5	6.1971	0	yes	RB BL $0.8n$	21.9	0.0530	0	yes
RB AL $0.4n$	21.3929	6.2129	0	yes	RB FL $0.8n$	21.6667	0.0586	0	yes
RB BL $0.2n$	21.2857	6.1791	0	yes	RB AL $0.8n$	21.6667	0.0526	0	yes
RB FL $0.4n$	20.7143	6.1676	0	yes	RB BL $0.6n$	21.1667	0.0442	0	yes
RB BL $0.4n$	20.2143	6.1697	0	yes	RB AL $0.6n$	20.6	0.0413	0	yes
RB AL $0.2n$	20.0714	6.0514	0	yes	RB FL $0.6n$	19.8	0.0468	0	yes
RB AL $0.6n$	18.6071	5.6991	0	yes	RB BL $0.4n$	17.3333	0.0247	0	yes
RB FL $0.6n$	17.8571	5.5717	0	yes	RB BL $0.2n$	16.5333	0.0224	0	yes
RB BL $0.6n$	17.7143	5.6132	0	yes	RB FL $0.4n$	15.9333	0.0222	0	yes
RB AL $0.8n$	14.5714	4.7705	0	yes	RB AL $0.4n$	15.7	0.0214	0	yes
RB BL $0.8n$	14.1429	4.7515	0	yes	RB AL $0.2n$	15.0333	0.0198	0	yes
RB FL $0.8n$	13.9286	4.6530	0	yes	RB FL $0.2n$	14.6667	0.0205	0	yes
RW FL $0.8n$	8.7857	0.1185	1	no	RW BL $0.2n$	10.5333	0.0088	0	yes
RW FL $0.6n$	8.6429	0.0996	0	no	RW BL $0.8n$	10.1333	0.0081	0	yes
RW BL $0.6n$	8.2857	0.0996	0	no	RW AL $0.2n$	9.4	0.0078	0	yes
RW BL $0.8n$	7.9286	0.1136	1	no	RW FL $0.2n$	9	0.0073	0	yes
RW AL $0.8n$	7.3214	0.0959	1	no	RW FL $0.8n$	7.5333	0.0065	0	no
RW BL $0.2n$	6.2857	0.0775	2	no	RW AL $0.8n$	7.3333	0.0062	0	no
RW AL $0.6n$	6.1429	0.0764	2	no	RW BL $0.4n$	7.2	0.0055	0	no
RW FL $0.2n$	5.8571	0.0626	4	no	RW AL $0.4n$	4.7333	0.0038	1	no
RW AL $0.2n$	5.5357	0.0633	3	no	RW FL $0.4n$	4.2333	0.0031	0	no
RW BL $0.4n$	5.0714	0.0762	2	no	RW BL $0.6n$	4	0.0027	1	no
RW FL $0.4n$	4.8929	0.0565	3	no	RW FL $0.6n$	2.1333	0.0010	4	no
RW AL $0.4n$	3.25	0.0412	4	no	RW AL $0.6n$	1.7667	0.0005	9	no

Table 1: Results of the IG instances

- The choice of the value for n_d has an important influence, as well, on the IG behavior. For both instance sets, the three best ranked algorithms employed the same setting for this parameter ($0.4n$ for the CHR set and $0.6n$ for the large instances).
- Although the results of Holm's test indicate that there not exist significant differences in the performance of the improvement procedures (and then, there exists little sensitivity to changes in this IG component), we choose the ABC local search for all remaining experiments involving IG since the best ranked configurations for the two instance sets are based on this method.

We now undertake to analyze the effects of the greedy constructive algorithms (C1 and C2 described in Section 4.1) on the IG performance. We have implemented IGs with C1 and C2, and the two best values of the n_d parameter identified in the previous experiment ($n_d = 0.4n$ and $0.6n$). They apply the RW acceptance criterion and the ABC local search. The Iman and Davenpor's test indicates the existence of significant differences among the rankings (the statistical values, 58.17 and 164.97 are larger than the critical ones, 2.84 and 2.82, respectively).

IG	CHR Instances				IG	LARGE Instances			
(Greedy, n_d)	*Av. Ran.*	*Dev*	*#Best*	Holm	(Greedy, n_d)	*Av. Ran.*	*Dev*	*#Best*	Holm
C2 $0.6n$	3.6071	0.7424	0	yes	C2 $0.4n$	3.9333	0.0102	0	yes
C2 $0.4n$	3.3929	0.6595	0	yes	C2 $0.6n$	3	0.0084	0	yes
C1 $0.6n$	1.6786	0.0440	5	no	C1 $0.4n$	2	0.0034	1	yes
C1 $0.4n$	1.3214	0.0098	10		C1 $0.6n$	1.0667	4.66E-05	14	

Table 2: Results of IG versions with different greedy constructive algorithms

With regard to this result, we compare the best ranked IG for each instance set, [C1, $0.4n$] and [C1, $0.6n$], with the other IG variants, by means of Holm's test. Table 2 reports its results. Our main observation about the data in this table is that Holm's test revealed the clear advantage of C1 on C2. It is interesting to remark that, quite surprisingly, the simplest constructive algorithm, when iterated inside IG, becomes the most effective one.

5.3 Study of the Memory Based Methods: SO and TLS

In this section, firstly, we present a series of preliminary experiments that were conducted to set the values of the key search parameters of TLS (Section 4.2), *TabuTenure* and *MaxIter*. In par-

ticular, we have built different SO instances (RW, C1, and $n_{tabu} = 0.05$) with $n_d = \{0.4n, 0.6n\}$, $TabuTenure = \{2, 10, 50\}$, and $MaxIter = \{100, 200, 500\}$. Table 3 summarizes the results of these algorithms and the conclusions of Holm's test (the Iman-Davenport test finds significant performance differences between the considered algorithms because its statistical values, 14.03 and 27.56, are greater than the critical ones, 1.669 and 1.665, respectively).

SO	CHR Instances				SO	LARGE Instances			
(n_d, TT, MI)	*Av. Ran.*	*Dev*	*#Best*	Holm	(n_d, TT, MI)	*Av. Ran.*	*Dev*	*#Best*	Holm
0.6n 50 500	16.75	0.1947	0	yes	0.6n 50 500	15.2	0.0072	0	yes
0.4n 50 200	13.3929	0.1096	0	yes	0.6n 2 500	14.9333	0.0064	0	yes
0.6n 10 500	13	0.1204	1	yes	0.6n 10 500	14.6667	0.0067	0	yes
0.4n 50 500	12.6786	0.1259	1	yes	0.6n 2 200	13.3333	0.0062	0	yes
0.6n 50 100	12.6071	0.1156	2	yes	0.6n 10 200	13.0333	0.0059	0	yes
0.6n 50 200	12.1786	0.1298	3	yes	0.6n 2 100	12.7333	0.0060	0	yes
0.6n 10 200	12.1429	0.1065	1	yes	0.6n 50 100	12.5	0.0055	0	yes
0.4n 10 500	11.3571	0.1016	2	yes	0.6n 50 200	12.1667	0.0055	0	yes
0.6n 10 100	10.8571	0.0961	2	yes	0.6n 10 100	11.8333	0.0053	0	yes
0.4n 10 200	9.8214	0.0869	3	yes	0.4n 10 500	9.0333	0.0034	1	yes
0.4n 50 100	9.7857	0.0790	2	yes	0.4n 50 500	8.3	0.0034	1	yes
0.6n 2 500	6.1786	0.0324	5	no	0.4n 2 500	7.6	0.0029	0	no
0.4n 10 100	5.7143	0.0350	6	no	0.4n 2 200	5.2667	0.0020	0	no
0.6n 2 200	5.6429	0.0425	3	no	0.4n 50 200	5.1667	0.0018	3	no
0.6n 2 100	5.1429	0.0364	3	no	0.4n 10 200	4.8	0.0019	2	no
0.4n 2 200	4.8929	0.0327	4	no	0.4n 2 100	4.0667	0.0014	4	no
0.4n 2 100	4.8214	0.0342	4	no	0.4n 10 100	3.3333	0.0012	3	no
0.4n 2 500	4.0357	0.0164	5		0.4n 50 100	3.0333	0.0011	6	

Table 3: Results of SO with TLS when applying different n_d, *TabuTenure*, and *MaxIter* values

The results in Table 3 show that the best outcomes for the CHR instances are obtained with the lowest *TabuTenure* value, and the ones for the large instances with the lowest *MaxIter* values. Interestingly, the *TabuTenure* parameter had less impact on the performance of TLS when dealing with large instances, and the same happens with the *MaxIter* parameter in the case of CHR. Nevertheless, given the relatively robust performance achieved by using $n_d = 0.4n$, *TabuTenure*=10, and $MaxIter = 100$ (second best ranked algorithm for the large instance set and Holm's test did not detect significant differences between this configuration and best one for CHR), we choose this setting for all remaining experiments involving this local search operator.

Next, we investigate the effects of varying the n_{tabu} parameter associated with SO (number of removed elements that cannot be reinserted into the solution). In particular, we have generated 5 SO configurations with $n_{tabu} = \{0.05n_d, 0.1n_d, 0.25n_d, 0.5n_d, n_d\}$. In order to study the results, we have applied Iman-Davenport's test (the level of significance considered was 0.05). We have observed the existence of significant differences among the rankings (the statistical values, 7.85 and 277.66, are greater than the critical ones, 2.54 and 2.53, respectively). Then, we analyze the performance of these SO instances by means of Holm's test (Table 4).

SO	CHR Instances				SO	LARGE Instances			
(n_{tabu})	*Av. Ran.*	*Dev*	*#Best*	Holm	(n_{tabu})	*Av. Ran.*	*Dev*	*#Best*	Holm
n_d	4.2857	0.10776	yes	2	n_d	5	0.02139	0	yes
0.5 n_d	3.4286	0.06932	yes	4	0.5 n_d	4	0.01429	0	yes
0.1 n_d	2.9286	0.05291	no	3	0.25 n_d	3	0.00697	0	yes
0.25 n_d	2.7143	0.03983	no	3	0.1 n_d	1.6	0.00129	6	no
0.05 n_d	1.6429	0.00514		12	0.05 n_d	1.4	0.00069	9	

Table 4: Results of SO with different n_{tabu} values

The results in Table 4 strongly reveals a clear superiority of the SO version with $n_{tabu} = 0.05n_d$ for both instance sets. Clearly, too much diversity (high n_{tabu} values) is not suitable to allow SO to reach fitter search areas. To sum up, the effects derived from the combination of the exploration power of TLS and a moderate diversification by SO (low n_{tabu} values) are sufficient to attain a high level of robustness for both instance sets.

5.4 Comparison Between IG, SO, and ITS

The first objective of this section is to compare the results for IG, SO, and Iterated Tabu Search ITS [19]. ITS is another memory-based approach proposed to deal with the QMSTP that has proved as one of the most appealing contemporary metaheuristic approaches for this problem and other combinatorial optimization problems with quadratic objective functions, including the unconstrained binary quadratic optimization problems [15], the maximum diversity problem [16], and the Max-2-SAT problem [17, 18]. We should point out that IG, SO and ITS were run under the same computational conditions (machine, programming language, compiler, and time limits; see Section 5.1) in order to enable a fair comparison between them. We have used the source code of ITS provided by the author[2]. Henceforth, all studied algorithms were run 10 times on each problem instance.

In Table 5, we have summarized the results of the IG and SO versions that achieved the best outcomes in Sections 5.2 and 5.3, respectively, and the ones of ITS (best performance measure values are outlined in boldface). For each algorithm, in column *Avg-Dev* , we include the average of the relative deviations from *BestValue* (Section 5.2) of the solution values found by the algorithm in the 10 runs on a set of problem instances and, in the case of the column *%best*, we outline the percentage of runs in which the algorithm reached *BestValue* for the instances in the corresponding group.

Inst. Set	IG		SO		ITS		HSII	
	Avg-Dev	*%best*	*Avg-Dev*	*%best*	*Avg-Dev*	*%best*	*Avg-Dev*	*%best*
NUG	0.0745	0	0.0429	7.33	**0.0104**	**15.33**	0.0176	8.67
CHR	0.2434	10	**0.0945**	**23.57**	0.9918	10	0.0665	20
RAND	**0.0016**	**8.89**	0.0048	0	0.0046	0	0.0021	1.11
SOAK	**0.001**	**6.67**	0.0072	0	0.0016	3.33	0.0012	2.22
CP	0.0005	71.94	0.004	25.56	**0.0004**	**77.50**	0.0006	72.22
Avg.	0.0642	19.5	0.03068	11.292	0.20176	21.232	0.0176	20.844

Table 5: Comparison between IG, SO, ITS, and HSII

The results of IG, SO, and ITS (Table 5) allow us to make the following observations.

- The high *Avg-Dev* values for IG, SO, and, specially, for ITS on the CHR set suggest that it includes the hardest QMSTP instances. Note that, in this case, the results of SO are much better than those of its competitors. The high levels of diversity promoted by the memory-based techniques in SO became decisive to guarantee an effective search when dealing with these complex problem instances.

- For the large instance sets, RAND and SOAK, the *Avg-Dev* and *%best* measures for IG show better quality than the ones for SO and ITS. We should note that, in contrast with the previous case, SO obtained the lowest quality results. For problems with many vertices, the action of the memory-based techniques in SO may produce detrimental disruptions in the solutions, causing misbehavior in the SO search. However, the diversification originated from the random deletion of elements during the destruction phase in IG may be enough to favor the progress towards promising search zones.

- ITS outperforms IG and SO on the CP and NUG instance sets.

Hence, it seems that, somewhat unsurprisingly, there is no single best algorithm; SO is superior for the most complex instance set, IG for the large instances, and ITS becomes the winner for the case of the easiest instances. Thus, regarding these results and with the aim of producing a robust operation, we have built a hybrid algorithm, dubbed HSII, that combines SO, IG, and ITS. Specifically, HSII is a *relay collaborative* hybrid metaheuristic [24] that executes SO, IG, and ITS in a pipeline fashion. First, SO is performed during the $P_{SO}\%$ of the

[2]ITS is publicly available at http://www.soften.ktu.lt/∼gintaras/qmstp.html

imposed time limit and the best found solution becomes the initial solution for IG, which is run during P_{IG}% of this time. Finally, the output of IG is supplied as input to ITS. We have set P_{SO} and P_{IG} to 25% and 50%, respectively. The idea of this hybridization scheme is: (1) to use SO as *diversification* agent to reach good initial solutions for the most difficult problem instances, then (2) to allow IG enough time to offer a suitable behavior on large instances, and finally (3) to employ ITS as effective *improvement* procedure for refining the best solution found in the previous stages.

Table 5 has the *Avg-Dev* and *%best* values for HSII. In general, for most problem sets, this algorithm might obtain *Avg-Dev* values very similar to the ones returned by the fittest algorithms (or even better, as was the case for CHR). Thus, we may conclude that the combination of the proposed IG and SO approaches along with ITS (by following a simple hybridization scheme) resulted really advisable to achieve an acceptable level of robustness across a wide range of different QMSTP instances.

5.5 HSII vs. State-of-the-art Metaheuristics for the QMSTP

In this section, we undertake a comparative analysis among HSII and the current best algorithms for the QMSTP, ITS [19], ABC [23], and local search algorithm with tabu thresholding (LS-TT) [14]. We should point out that HSII, ITS, and ABC were run under the same computational conditions (Section 5.1). We have implemented ABC in C. Since we could not obtain the source code of LS-TT, we have used the results reported in [14] for the NUG and CHR instances to compare with our algorithm (they were obtained with a single run). The parameter values used for each considered algorithm are the ones recommended in the original works. Their results are outlined in Tables 9-13 in Appendix A. With the aim of determining the position of SO and IG with regards to the state-of-the-art, we have included them in this comparative study as well.

In order to detect the differences among HSII and the other algorithms, we have applied Wilcoxon's test. Tables 6-8 have the *%best* and *Avg-Dev* measures and summarize the results of this procedure for $p = 0.05$, where the values of $R+$ (associated to HSII) and $R-$ of the test are specified. The last column indicates whether Wilcoxon's test found statistical differences between these algorithms. If $min\{R^+, R^-\}$ is less than or equal to the critical value, this test detects significant differences between the algorithms, which means that an algorithm outperforms its opponent. Particularly, if this occurs and $R^- = min\{R^+, R^-\}$, then HSII is statistically better than the other algorithm. All large instances (RAND and SOAK) and QAP based instances (NUG and CHR) have been grouped in order to apply the statistical test to a significant number of instances.

Algorithm	*Avg-Dev*	*%best*	$R+$	$R-$	Diff?
IG	0.0005	71.94	300.0	366.0	no
SO	0.0040	25.56	628.5	1.5	yes
ITS	0.0004	77.50	174.5	455.5	yes
ABC	0.0061	20.28	661.0	5.0	yes
HSII	0.0006	72.22			

Table 6: Comparison on the CP set (Wicoxon's test; critical value = 208)

	NUG set		CHR set				
Algorithm	*Avg-Dev*	*%best*	*Avg-Dev*	*%best*	$R+$	$R-$	Diff?
IG	0.0745	0	0.2459	10.00	434.0	1.0	yes
SO	0.0429	7.33	0.0965	22.86	294.5	140.5	no
ITS	0.0104	15.33	0.9965	10.00	196.0	210.0	no
ABC	0.2597	0.00	1.9261	0.00	435.0	0.0	yes
LS-TT	0.0671	0.00	0.2918	7.14	411.0	24.0	yes
HSII	0.0176	8.67	0.0686	20.00			

Table 7: Comparison on the NUG and CHR sets (Wicoxon's test; critical value = 126)

	RAND set		SOAK set				
Algorithm	*Avg-Dev*	*%best*	*Avg-Dev*	*%best*	*R+*	*R−*	Diff?
IG	0.0016	8.89	0.0010	6.67	23.0	148.0	yes
SO	0.0048	0	0.0072	0	171.0	0.0	yes
ITS	0.0046	0	0.0016	3.33	162.0	9.0	yes
ABC	0.0111	0.00	0.0077	0.00	171.0	0.0	yes
HSII	0.0021	1.11	0.0012	2.22			

Table 8: Comparison on the RAND and SOAK sets (Wicoxon's test; critical value = 40)

The results presented in Tables 6-8 reveal the following.

- For large instances (Table 8), HSII has the upper hand in the statistical comparison over its competitors (ABC and ITS).
- For complex instances (Table 7), our hybrid metaheuristic clearly obtained statistically significant improvements relative to ABC and LS-TT. In addition, analyzing the computational time reported in [14] (where a Pentium IV PC at 3 GHz was used to carry out experiments) for LS-TT on these instances (Tables 10 and 11), we may conclude that, in general, LS-TT required much time than our hybrid algorithm (with a time limit of 1000 seconds). Therefore, our proposal outperforms this algorithm in all aspects, when considering the results on the NUG and CHR instance sets.

  On the other hand, Wilcoxon's test did not detect significant differences between HSII and ITS on these instances. According to the *Avg-Dev* and *%best* measures, our algorithm clearly beats ITS for the CHR instances. For the case of the NUG instances, they show similar values for the *Avg-Dev* measure, which explains that Wilcoxon's test did not find statistical differences between them when the two sets of instances were considered.
- HSII was found to be superior to ABC for the CP instances (Table 6), however, only in this case, ITS could statistically outperform our proposal (even so, note that HSII attained a similar *%best* value). It is worth mentioning that, in [19], iterated tabu search proved superior to other advanced methods precisely on this instance set.

In summary, this experimental analysis confirms that our hybrid HSII approach is a very attractive alternative to the existing approaches for the QMSTP.

Finally, we should note that SO achieved better *Avg-Dev* and *%best* values than ABC and LS-TT in the case of the NUG and CHR sets (Table 7), and IG was able to outperform all other algorithms on the RAND and SOAK sets (Table 8). These facts evidence the great potential of the proposed SO and IG metaheuristics to effectively handle large and highly complex problem instances, which posed real challenges for previous metaheuristic approaches in the literature.

6 Conclusions

Our research study demonstrates the effectiveness of a strategy for solving the QMSTP that alternates between constructive and destructive phases, as originally proposed in strategic oscillation (SO) and more recently in the iterated greedy (IG) method. We limit consideration of SO in this study to a one-sided oscillation that does not cross the feasibility boundary in order to make it more comparable to the IG approach, and differentiate it from IG primarily by introducing tabu search (TS) memory as in previous SO algorithms.

Our tests disclose that the memory-based SO method is able to solve complex QMSTP instances better than other previous algorithms. On the other hand, our implementation of the IG algorithm succeeds in performing most effectively for large problems that are not highly complex, while the iterated tabu search (ITS) algorithm performs best in application to easier instances. Based on these findings we additionally developed a hybrid method HSII

that combines these three algorithms, which proved very effective across the board with the exception of one class of problem instances, where iterated tabu search remained the winner.

Overall, the ability of the alternating constructive/destructive strategies to give superior outcomes for larger problems, with memory proving valuable for complex problems and a disregard for memory proving valuable for easier problems, invites further consideration of other forms of strategic oscillation, particularly by crossing feasibility boundaries (whose value is underscored in [9]) and by going beyond the reliance on randomization in carrying out destructive moves by introducing more advanced strategies for selecting these moves (as indicated in [7]).

Acknowledgments

We are thankful to Dr. Temel Öncan for providing the NUG and CHR instances used in [14]. This work was supported by the Research Projects TIN2011-24124, TIN2009-07516, and P08-TIC-4173.

A Results of the Algorithms

Tables 9-13 outline the results of the algorithms concerning the experiment in Section 5.5. The columns with heading 'Ave' (respectively, 'Min') present the difference between the average of the objective function value of the solutions reached by an algorithm after 10 runs (respectively, the objective function value of the best solution out of these 10 solutions) and the value in the second column. Additionally, Tables 10 and 11 have the computational times required by LS-TT to achieve their results (both of them were directly extracted from [14]).

Instance	Best Known V.	HSII		ITS		ABC	
		Ave	Min	Ave	Min	Ave	Min
n40_m257_1	5945	0	0	0	0	13.8	0
n40_m257_2	56237	0	0	0	0	0	0
n40_m257_3	6925	0	0	0	0	0	0
n40_m257_4	57874	0	0	0	0	0	0
n40_m522_1	5567	3.2	0	0.6	0	24.8	18
n40_m522_2	51851	0	0	24.6	0	392.2	220
n40_m522_3	6456	0	0	0	0	18.9	18
n40_m522_4	53592	9.9	0	13	0	104.5	43
n40_m780_1	5368	0.5	0	0.9	0	68.1	13
n40_m780_2	49817	18.2	0	6.2	0	591.4	129
n40_m780_3	6208	0	0	1	0	8	0
n40_m780_4	51229	77.1	0	0	0	647.6	0
n45_m326_1	7521	0	0	0	0	1.4	0
n45_m326_2	70603	0	0	0	0	42.9	0
n45_m326_3	8720	0	0	0	0	4.7	0
n45_m326_4	72676	0	0	0	0	82	82
n45_m663_1	7161	16.8	0	15.7	0	51.9	44
n45_m663_2	66889	23.8	0	0	0	287.6	0
n45_m663_3	8225	0.6	0	0	0	36.3	25
n45_m663_4	68737	46	0	0	0	440.2	0
n45_m990_1	6944	7.3	1	4.7	0	76.9	47
n45_m990_2	64840	45.6	0	50.9	0	1048.3	583
n45_m990_3	7827	0	0	0	0	4	0
n45_m990_4	66508	148	0	141.9	0	1042.2	331
n50_m404_1	9393	0	0	0	0	34.1	24
n50_m404_2	88942	0	0	0	0	370.4	349
n50_m404_3	10717	0	0	0	0	0	0
n50_m404_4	91009	0	0	0	0	165	165
n50_m820_1	8958	10.5	0	1.3	0	76.8	33
n50_m820_2	84020	100.4	0	4.4	0	1256.6	601
n50_m820_3	10100	0	0	0	0	46.7	37
n50_m820_4	86231	147.6	0	18.7	0	1169.9	599
n50_m1225_1	8713	25.1	0	21	0	128.7	53
n50_m1225_2	81858	205.5	27	192.5	27	1128.1	450
n50_m1225_3	9836	8.4	0	1.9	0	47.8	20
n50_m1225_4	83838	31.6	0	102.1	0	883.6	43

Table 9: Results for CP instance set

Instance	Opt. Value	HSII		ITS		ABC		LS-TT	
		Ave	Min	Ave	Min	Ave	Min	Ave/Min	Time
nug12	578	0	0	0	0	165.8	78	27	639
nug14	1014	20.8	12	9.2	0	273.8	126	70	724
nug15	1150	16.8	2	10.2	0	334.6	254	115	1348
nug16a	1610	39.4	24	36.6	28	432	334	132	2311
nug16b	1240	33.2	6	28.4	8	369.8	240	110	2936
nug17	1732	58.6	42	49.2	36	475.8	334	142	3422
nug18	1930	74.8	54	61	34	488.6	294	126	3482
nug20	2570	127	92	100	74	645.4	330	290	5151
nug21	2438	137.2	102	106.8	64	870.6	604	260	5184
nug22	3596	190.2	154	161	116	1397.2	984	272	5482
nug24	3488	237.6	200	187	160	1133	852	386	5914
nug25	3744	252.6	196	224.4	210	1116.4	778	339	5983
nug27	5234	342.6	300	303.4	222	1481	1050	732	6025
nug28	5166	353.2	318	321.4	240	1300.2	1072	653	6087
nug30	6124	482	404	429.6	382	1775.8	1564	799	6227

Table 10: Results for NUG instance set

Instance	Opt. Value	HSII		ITS		ABC		LS-TT	
		Ave	Min	Ave	Min	Ave	Min	Ave/Min	Time
chr12a	9552	0	0	14625.6	7142	20331.4	4738	1618	783
chr12b	9742	92	0	16221.2	6614	22457	11810	1011	790
chr12c	11156	3	0	12243.8	6278	14143.2	4654	1556	783
chr15a	9896	446.6	56	16624.8	6822	27719.4	14328	1742	1239
chr15b	7990	1628	394	21137.8	9218	33899.8	20350	2155	1136
chr15c	9504	1035.6	0	19956.2	9798	25663.2	16062	3265	1254
chr18a	11098	3439.2	2736	22572.4	11398	31298.6	13856	1659	3325
chr18b	1534	3.4	0	0	0	1123	626	142	3354
chr20a	2192	225.8	84	162.4	40	4142.4	2550	253	4968
chr20b	2298	223.8	164	184.4	142	3435	1406	432	4652
chr20c	14142	7976	6064	31867.6	22416	50674.6	35700	15982	4763
chr22a	6156	312.6	178	269	234	3759.8	2532	2604	5089
chr22b	6194	336.6	202	249.4	120	4067.6	2714	2208	4741
chr25a	3796	929.6	514	787	504	7728.2	4744	5862	5223

Table 11: Results for CHR instance set

Instance	Best Known V.	HSII		ITS		ABC	
		Ave	Min	Ave	Min	Ave	Min
RAND-150-1	192606	304.1	0	638.5	340	2223.1	1688
RAND-150-2	192607	315.8	0	762.9	427	2187	1611
RAND-150-3	192577	215.6	0	726.1	388	2112.8	1305
RAND-200-1	350517	506.6	0	1270.2	699	3874.3	2646
RAND-200-2	350389	513.4	0	1434.7	923	3756.3	3395
RAND-200-3	351057	228.4	0	883.8	409	2856.4	2112
RAND-250-1	556929	505.6	0	2306.5	1522	5585.8	4935
RAND-250-2	557474	376.1	0	2004.2	1346	4898	3230
RAND-250-3	556813	650.4	0	2676.8	2491	5423.3	4684

Table 12: Results for RAND instance set

Instance	Best Known V.	HSII		ITS		ABC	
		Ave	Min	Ave	Min	Ave	Min
SOAK-150-1	206721	368.4	204	283.9	0	1557.2	931
SOAK-150-2	206761	519.5	341	392.6	0	2024.1	1445
SOAK-150-3	206781	173.2	0	178.6	21	1336.4	752
SOAK-200-1	370137	393.5	128	396.3	0	3196.8	2282
SOAK-200-2	369982	201.6	0	369	46	2311	1659
SOAK-200-3	370045	300.8	0	345.7	1	2860	2072
SOAK-250-1	581819	464.7	0	1250.4	463	4260.3	2980
SOAK-250-2	581691	322.3	0	1181.9	454	4132.7	2718
SOAK-250-3	581854	736.8	0	1671.8	854	4491.1	3863

Table 13: Results for SOAK instance set

References

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

[2] Cordone R, Passeri G. Heuristic and exact approaches to the quadratic minimum spanning tree problem. In: Seventh Cologne-Twente Workshop on Graphs and Combinatorial Optimization (CTW08), Gargnano, Italy, Universit degli Studi di Milano, 2008, pp. 52–55.

[3] Christofides N, Benavent E. An exact algorithm for the quadratic assignment problem. Operations Research 1989; 37(5):760–8.

[4] Fanjul-Peyroa L, Ruiz R. Iterated greedy local search methods for unrelated parallel machine scheduling. European Journal of Operational Research 2010; 207: 55–69.

[5] Gao J, Lu M. Fuzzy quadratic minimum spanning tree problem. Applied Mathematics and Computation 2005; 164: 773–788.

[6] García S, Molina D, Lozano M, Herrera F. A study on the use of nonparametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 special session on real parameter optimization. Journal of Heuristics 2009; 15: 617–644.

[7] Glover F, Laguna M. Tabu Search. Dordrecht: Kluwer, 1997.

[8] Glover F. Heuristics for Integer Programming Using Surrogate Constraints. Decision Sciences 1977; 8: 156–166.

[9] Glover, F., Hao, J.K. The Case for Strategic Oscillation.Annals of Operations Research 2011; 183 (1): 163–173.

[10] Jacobs LW, Brusco MJ. A local-search heuristic for large set-covering problems. Naval Research Logistics 1995; 42: 1129–1140.

[11] Kruskal JB. On the shortest spanning subtree of a graph and the traveling salesman problem. Proceedings of the American Mathematical Society 1956; 7: 48–50.

[12] Lozano M, Molina D, García-Martínez C. Iterated greedy for the maximum diversity problem. European Journal of Operational Research 2011; 214: 31–38.

[13] Nugent CE, Vollman TE, Ruml J. An experimental comparison of techniques for the assignment of facilities to locations. Operations Research 1968; 16: 150–73.

[14] Öncan T, Punnen AP. The quadratic minimum spanning tree problem: a lower bounding procedure and an efficient search algorithm. Computers and Operations Research 2010; 37: 1762–1773.

[15] Palubeckis G. Iterated tabu search for the unconstrained binary quadratic optimization problem. Informatica 2006; 17: 279–296.

[16] Palubeckis G. Iterated tabu search for the maximum diversity problem. Applied Mathematics and Computation 2007; 189: 371–383.

[17] Palubeckis G. Solving the weighted Max-2-SAT problem with iterated tabu search. Information Technology and Control 2008; 37: 275–284.

[18] Palubeckis G. A new bounding procedure and an improved exact algorithm for the Max-2-SAT problem. Applied Mathematics and Computation 2009; 215: 1106–1117.

[19] Palubeckis G, Rubliauskas D, Targamadz A. Metaheuristic approaches for the quadratic minimum spanning tree problem. Information Technology and Control 2010; 39(4): 257–268.

[20] Ruiz R, Stützle T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research 2007; 177: 2033–2049.

[21] Ruiz R, Stützle T. An Iterated Greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. European Journal of Operational Research 2008; 187: 1143–1159.

[22] Soak SM, Corne DW, Ahn BH. The edge-window-decoder representation for tree-based problems. IEEE Transactions on Evolutionary Computation 2006; 10: 124–144.

[23] Sundar S, Singh A. A swarm intelligence approach to the quadratic minimum spanning tree problem. Information Sciences 2010; 180; 3182–3191.

[24] Talbi E-G. A taxonomy of hybrid metaheuristics. J Heuristics 2002; 8(5): 541–65.

[25] Xu W. On the quadratic minimum spanning tree problem. In: Gen M, Xu W, editors. Proceedings of 1995 Japan–China International Workshops on Information Systems, Ashikaga, Japan, 1995, pp. 141–148.

[26] Ying KC. Solving non-permutation flowshop scheduling problems by an effective iterated greedy heuristic. International Journal of Advanced Manufacturing Technology 2008; 38(3–4): 348–354.

[27] Ying KC, Cheng HM. Dynamic parallel machine scheduling with sequence-dependent setup times using an iterated greedy heuristic. Expert Systems with Applications 2010; 37(4): 2848–2852.

[28] Zhou G, Gen M. An effective genetic algorithm approach to the quadratic minimum spanning tree problem. Computers and Operations Research 1998; 25: 229–237.