ISSN 1392–124X INFORMATION TECHNOLOGY AND CONTROL, 2010 Vol. 39, No. 4

METAHEURISTIC APPROACHES FOR THE QUADRATIC MINIMUM SPANNING TREE PROBLEM

Gintaras Palubeckis*, Dalius Rubliauskas*, Aleksandras Targamadzė**

**Multimedia Engineering Department, Kaunas University of Technology*
Studentu St. 50, LT-51368 Kaunas, Lithuania
***Software Engineering Department, Kaunas University of Technology*
Studentu St. 50, LT-51368 Kaunas, Lithuania
e-mail: gintaras@soften.ktu.lt, dalius@soften.ktu.lt, a.targamadze@internet.ktu.lt

Abstract. Given an undirected graph with costs associated both with its edges and unordered pairs of edges, the quadratic minimum spanning tree problem asks to find a spanning tree that minimizes the sum of costs of all edges and pairs of edges in the tree. We present multistart simulated annealing, hybrid genetic and iterated tabu search algorithms for solving this problem. We report on computational experiments that compare these algorithms on random graphs of size up to 50 vertices. The results indicate that the iterated tabu search algorithm is superior to the other two approaches in terms of both solution quality and computation time.

Keywords: combinatorial optimization, quadratic minimum spanning tree, metaheuristics, simulated annealing, genetic algorithm, tabu search.

1. Introduction

Let $G = (V, E)$ be an undirected graph with vertex set V and edge set E. The quadratic minimum spanning tree problem (QMSTP) is a generalization of the classical minimum spanning tree problem where we are given not only edge costs $c_e > 0$, $e \in E$, as in the latter, but in addition also interaction costs c_{eg} between pairs of edges e, g, $e \neq g$, in E. The quality of a spanning tree $T = (V, E(T))$ in the case of the QMSTP is evaluated by the following objective function:

$$F(T) = \sum_{e \in E(T)} c_e + \sum_{(e,g) \in \Psi(T)} c_{eg}, \qquad (1)$$

where $\Psi(T)$ is the set of all unordered pairs of edges in $E(T)$ (from now on we assume that the order of the subscripts does not matter, and both c_{eg} and c_{ge} refer to the same value). The QMSTP asks to find a spanning tree T that minimizes the objective function given by (1). The problem was first introduced by Assad and Xu [1]. It arises in several contemporary application areas such as telecommunications, transportation and energy distribution.

The QMSTP can be regarded as a special case of the binary quadratic optimization problem (BQOP for short). Indeed, let S be a set of objects of some kind and P be the power set of S, i.e., the set of all subsets of S including the empty set and S itself. Let us denote by c'_i, $i \in S$, the cost of selecting object i and by c'_{ij}, $i, j \in S$, $i < j$, the cost incurred by selecting objects i and j. Assuming that P' is a fixed subset of P, the BQOP can be stated as follows:

$$\min \text{ (or } \max) \sum_{i \in S'} c'_i + \sum_{i,j \in S', i<j} c'_{ij} \qquad (2)$$

$$\text{s.t.} \quad S' \in P'. \qquad (3)$$

Perhaps the most widely studied case of (2), (3) is the one where $P' = P$. This case is called an unconstrained binary quadratic optimization problem (UBQOP) (see [6], [11], [15]). As another example, for a fixed positive integer k, consider P' consisting of all subsets of S of size k. Suppose that the objective function in (2) is maximized. Then, for such P', (2), (3) is the formulation of the maximum diversity problem (see [4], [9]). The QMSTP can be recast into the form (2), (3) as well. This fact comes from the following observations: $S = E$; c_e and c_{eg} in (1) correspond to c'_i and c'_{ij} in (2), respectively; P' is the collection of edge sets of all spanning trees of G.

The QMSTP is known to be NP-hard [1], [20] and thus cannot be solved to optimality in polynomial time, unless P=NP. Therefore, exact algorithms are viable only for solving QMSTP instances of small size. Assad and Xu [1] proposed such an algorithm based on the general branch-and-bound scheme. They reported results for QMSTP instances of size up to 15 vertices. However, the exact algorithms rapidly degrade as the size of the graph grows. Thus, when

dealing with medium to large size graphs, efficient algorithms, which provide good, but not necessarily optimal solutions, are required. The first two heuristics for the QMSTP were given in [1]. These algorithms are constructive by nature and, therefore, are not able to provide good enough solutions for larger graphs. Zhou and Gen [21] presented a genetic algorithm (GA) in which the Prüfer number [16] to encode a spanning tree was adopted. They reported computational results for 17 test instances. Soak, Corne and Ahn [18] developed another genetic algorithm which employed a decoder-based redundant encoding strategy. They have shown that their GA implementation outperforms genetic algorithm using the Prüfer number representation. More recently, Cordone and Passeri [2] have applied a tabu search technique to solve the QMSTP. At each iteration of their algorithm, the search is performed in the 1-exchange neighborhood, which consists of all spanning trees that can be obtained from the current spanning tree by replacing one of its edges with a non-tree edge. In [10], Öncan and Punnen have presented a local search algorithm with tabu thresholding. In this algorithm, the same neighborhood structure as in [2] is used. An artificial bee colony algorithm for solving the QMSTP was proposed by Sundar and Singh [19]. In the last phase, the algorithm makes a call to a local search procedure. This algorithm compares favourably with earlier evolutionary approaches. Gao and Lu [5] introduced the fuzzy quadratic minimum spanning tree problem. It is formulated as expected value model, chance-constrained programming and dependent-chance programming according to different criteria. In [5], a genetic algorithm using Prüfer number representation for solving this problem was developed.

The purpose of the current paper is to investigate computationally the applicability of three quite different metaheuristics to the QMSTP, namely, simulated annealing, genetic algorithm and tabu search. We compare two versions of the genetic algorithm. The first one is a pure GA, while the second is a GA hybridized with an effective local search procedure. The tabu search metaheuristic is represented by an iterated tabu search algorithm. Such an approach appeared to be successful when applied to other combinatorial optimization problems with quadratic objective function, including the UBQOP [11], the maximum diversity problem [12] and the Max-2-SAT problem [13], [14]. The empirical results summarized in this paper were obtained by testing the above listed algorithms on the QMSTP instances used by Cordone and Passeri [3].

The remainder of this paper is organized into five sections. In Sections 2 to 4, we describe our implementations of simulated annealing, genetic and tabu search algorithms, respectively. In Section 5, we present computational results. Finally, in Section 6, some concluding remarks are made.

To end this introduction, let us give a few notations used in the sequel of the paper. We denote by n and m the number of vertices and edges of a graph $G = (V, E)$, respectively. Given a spanning tree $T = (V, E(T))$ of G and edges $e \in E(T)$, $g \in E \setminus E(T)$, we will write $T(e, g)$ for the subgraph of G obtained from T by replacing the edge e with the edge g. Trivially, the subgraph $T(e, g)$ is connected if and only if it is a spanning tree. The set $N(T) = \{T(e, g) \mid e \in E(T), g \in E \setminus E(T)$ and $T(e, g)$ is connected$\}$ is called a neighborhood of T.

2. Simulated annealing

In this section, we describe an implementation of the simulated annealing algorithm for solving the QMSTP. Simulated annealing (SA) is a general-purpose optimization method that attempts to exploit an analogy between the physical process of annealing and the process of obtaining a global extremum of a function. In physical annealing, a material, like metal or glass, is first heated up to a very high temperature and then slowly cooled down to reach the lowest energy state. During the optimization process, each solution corresponds to a state of some physical system and the value of the objective function corresponds to the energy level. In our SA algorithm, the initial (high) temperature, denoted $\bar{t}$, is obtained by evaluating a certain number of trees selected at random from the neighborhood $N(T)$ of a randomly generated spanning tree $T = (V, E(T))$. More precisely, suppose that the tree $T' \in N(T)$ is obtained from T by replacing an edge $e \in E(T)$ with an edge $g \in E \setminus E(T)$. We denote by $\delta(T, e, g) = F(T') - F(T)$ the change in the objective function value when moving from T to T'. Clearly, $\delta(T, e, g) = c_g - c_e + \sum_{h \in E(T) \setminus \{e\}} (c_{gh} - c_{eh})$. The algorithm assigns to $\bar{t}$ the largest absolute value of $\delta(T, e, g)$ over a sample of 10000 spanning trees randomly drawn from $N(T)$. Other parameters of the algorithm are the cooling rate α, minimum temperature t_0, which should be very close to 0, and repetition factor R_0.

In order to make a fair comparison between the different algorithms, we use the same stopping rule for each of them. Our choice was to stop an algorithm after a prescribed time period had elapsed. The running time of the SA algorithms typically depends on the cooling rate as well as on the repetition factor. Therefore, to be able to apply our stop-

ping rule, we execute the simulated annealing procedure repetitively using randomly constructed spanning trees as starting points. The steps of the main algorithm, named MSA (Multistart Simulated Annealing), are as follows.

MSA

1. Randomly generate a spanning tree T. Initialize T^* with T and F^* with $F(T)$.
2. Compute $\bar{t} = \max\{|\delta(T, e, g)| \mid (e, g) \in H\}$, where H is a set of randomly selected edge pairs (e, g) for which $T(e, g) \in N(T)$. Set $K := \lfloor(\log(t_0) - \log(\bar{t}))/\log\alpha\rfloor$, $R := R_0 n$ and *start* $:= 1$.
3. Apply SA(T, T^*, F^*, $\bar{t}$, K, R, α, *start*). Increment *start* by 1.
4. Check if the termination condition is satisfied. If so, then stop with the spanning tree T^* of value F^*. Otherwise return to 3.

In the above description, T^* is used to denote the best spanning tree found so far. The corresponding value of the objective function is denoted by F^*. The algorithm starts with the tree $T = T^*$ generated using the augmentation technique. Initially, the tree consists of a single vertex. At each of $n - 1$ steps of the generation routine, the tree $T = (V(T), E(T))$ is augmented with a randomly selected edge having exactly one endpoint in $V \setminus V(T)$. At the end of this process, a tree spanning all the vertices in V is obtained. Step 2 of MSA prepares the parameters to be passed to the simulated annealing algorithm. The parameter K is the number of temperature reductions, R is the number of trees evaluated at a temperature level, and *start* is the counter for the number of calls to the procedure SA given below.

SA($T, T^*, F^*, \bar{t}, K, R, \alpha$,*start*)

1. If *start* $= 1$, then initialize f with F^* (which equals $F(T)$). Otherwise, randomly generate a spanning tree T and set $f := F(T)$.
2. Initialize t with $\bar{t}$ and i with 1.
3. Set $j := 1$.
4. Randomly select edges e and g such that $T(e, g) \in N(T)$. Compute $\delta' = \delta(T, e, g)$. If $\delta' \leqslant 0$, then go to 5. Otherwise, randomly draw a number ξ from the uniform distribution on $[0, 1]$. If $\xi \leqslant \exp(-\delta'/t)$, then proceed to 5; else go to 6.
5. Update the tree T by substituting the edge e by the edge g. Set $f := f + \delta'$. If $f < F^*$, then set $F^* := f$ and store T as the new best solution: $T^* \leftarrow T$.
6. Increment j by 1. If $j \leqslant R$, then go to 4.
7. Increment i by 1. If $i \leqslant K$, then set $t := \alpha t$ and go to 3. Otherwise return with T^* and F^*.

As can be seen from the above description, simulated annealing process is started from a random spanning tree. When SA is invoked for the first time, the same random tree as in MSA is used. The main body of SA consists of two nested loops. The outer one successively modifies the temperature t by multiplying it by the cooling factor α. The initial temperature is set equal to $\bar{t}$. Each iteration of the inner loop evaluates a tree selected at random from the neighborhood $N(T)$ of the current spanning tree T. If either the new tree $T(e, g)$ is at least as good as T or the condition stated in Step 4 is satisfied, then $T(e, g)$ is accepted to replace T. In the case where the current objective function value f is smaller than F^*, the updated tree T is saved as the best solution found so far. The time complexity of an iteration of the inner loop is $O(m)$.

3. Genetic algorithm

A widely used approach for solving optimization problems is based on the idea of applying evolutionary principles to problem solutions. The most famous technique in the area of evolutionary computation is the genetic algorithm. We have developed a GA to solve the QMSTP. In our algorithm, each individual of the population is a spanning tree represented by the list of its edges. Therefore, our approach differs from the genetic algorithm of Zhou and Gen [21], in which spanning trees are encoded using Prüfer numbers. Our decision in choosing the tree representation was influenced by the fact that, according to several authors, including Raidl and Julstrom [17] and Gottlieb et al. [7], the Prüfer encoding is not suitable for GA due to its low locality and heritability. Moreover, Prüfer numbers can be used to represent spanning trees only in the case where the QMSTP instance graph is complete. In this study, our aim is to develop algorithms for the QMSTP, including the genetic algorithm, which could deal with graphs of arbitrary density.

Actually, in this section we will present two variations of the genetic algorithm — the pure GA and the hybrid GA which is obtained from the former by equipping it with a local search procedure. The first step of the genetic algorithm is to create a random population of individuals. For this purpose, a randomized version of Kruskal's algorithm [8] is adopted. At each iteration, the algorithm considers the set of all edges that can be used to augment the current for-

est $(V, E_{\rm sel})$ and chooses a subset of "best" candidates, that is, those feasible edges e for which the sum $Q_e = c_e + \sum_{g \in E_{\rm sel}} c_{eg}$ is the smallest. One of the edges is randomly selected from this subset and added to the tree being built. If the resulting tree appears to be very similar to some of the individuals in the population, then it is rejected. Otherwise it is accepted as a new member of the population. Once the initial population has been created, the algorithm enters into the evolution phase, which includes the following main steps: reproducing offspring, applying local search to offspring, and updating the current population. Let the latter be denoted by Π. The crossover operation is performed on the two spanning trees randomly chosen from Π. It consists of two steps. In the first step, the algorithm identifies all edges that are common to both parents (for illustration, see Figure 1 where (c) displays the result of this step for parents shown in (a) and (b)). In the second step, the offspring is completed by adding edges which belong to only one of the parents (Figure 1, (d)). This is done in the same way as in the above outlined randomized version of Kruskal's algorithm. Afterwards, the offspring is submitted to a local search procedure. Basically, this procedure can be regarded as a kind of mutation operator. To update Π, the algorithm employs a frequently used strategy of replacing the worst individual in the population with the generated offspring. Before doing this, the algorithm checks whether the offspring is sufficiently different from each individual in Π. If the answer is negative, then the offspring is simply discarded. The algorithm, named HGA (Hybrid Genetic Algorithm), can be described as follows.

HGA

1. (Initialization) Set $\Pi := \emptyset$, $l := 0$, $F^* := \infty$ and $\rho := 0$. While $l <$ *pop_size* do the following:
   1.1. Set $E_{\rm sel} := \emptyset$, $E_{\rm cand} := E$ and $Q_e := c_e$ for each edge $e \in E$.
   1.2. Form a set E' of $z = \min(\bar{z}_1, |E_{\rm cand}|)$ edges $e \in E_{\rm cand}$ such that $Q_e \leqslant Q_g$ for each $e \in E'$ and each $g \in E_{\rm cand} \setminus E'$ (in other words, pick the z smallest values Q_e among those with $e \in E_{\rm cand}$). Select an edge $h \in E'$ at random.
   1.3. Move the edge h from $E_{\rm cand}$ to $E_{\rm sel}$. If $|E_{\rm sel}| = n - 1$, then proceed to 1.4. Otherwise, eliminate from $E_{\rm cand}$ all the edges that create a cycle when added to $E_{\rm sel}$. For the remaining edges $e \in E_{\rm cand}$, increase Q_e by c_{eh}. Return to 1.2.
   1.4. If $l = 0$, then go to 1.5. Otherwise, compute $D = \min_{T \in \Pi} |E_{\rm sel} \triangle E(T)|$, where $\triangle$ denotes the symmetric difference of two sets. If $D \geqslant D^1_{\rm min}$, then go to 1.5. Otherwise increment ρ by 1. Check whether $\rho < \bar{\rho}$. If so, then return to 1.1. If not, then set *pop_size* $:= l$ and escape from the loop 1.1–1.5.
   1.5. Set $\rho := 0$. Add to Π the tree $T = (V, E_{\rm sel})$. Increment l by 1.If $F(T) < F^*$, then set $T^* := T$ and $F^* := F(T)$. If $l <$ *pop_size*, then repeat the loop.
2. (Parents selection) Randomly choose two trees, say $T_1 = (V, E(T_1))$ and $T_2 = (V, E(T_2))$, from the population Π.
3. (Mating) Perform the following steps:
   3.1. Initialize the set $E_{\rm sel}$ with all edges that are common to both trees T_1 and T_2.
   3.2. Set $E_{\rm cand} := E(T_1) \triangle E(T_2)$ and $Q_e := c_e + \sum_{g \in E_{\rm sel}} c_{eg}$ for each $e \in E_{\rm cand}$.
   3.3. Form a set E' of $z = \min(\bar{z}_2, |E_{\rm cand}|)$ edges $e \in E_{\rm cand}$ such that $Q_e \leqslant Q_g$ for each $e \in E'$ and each $g \in E_{\rm cand} \setminus E'$. Select an edge $h \in E'$ at random.
   3.4. Move the edge h from $E_{\rm cand}$ to $E_{\rm sel}$. If $|E_{\rm sel}| = n - 1$, then go to 4. Otherwise, eliminate from $E_{\rm cand}$ all the edges that create a cycle when added to $E_{\rm sel}$. For the remaining edges $e \in E_{\rm cand}$, increase Q_e by c_{eh}. Return to 3.3.
4. (Local search) Apply the local search procedure LS(T) to the tree $T = (V, E_{\rm sel})$. Let T also denote the tree returned by it.
5. (Offspring evaluation) Perform the following steps:
   5.1. Check whether $F(T) < F^*$. If so, then set $T^* := T$, $F^* := F(T)$ and go to 5.3. Otherwise proceed to 5.2.
   5.2. If $T = (V, E(T))$ is worse than the worst tree in the population, then go to 6. Otherwise, compute $D = \min_{T' \in \Pi} |E(T) \triangle E(T')|$. If $D < D^2_{\rm min}$, then go to 6; else proceed to 5.3.
   5.3. Replace the worst tree in Π by T.
6. Check if the termination condition is satisfied. If so, then stop with the spanning tree T^* of value F^*. Otherwise return to 2.

In the algorithm, *pop_size* stands for the cardinality of Π. The other parameters are $\bar{z}_1$, $\bar{z}_2$, $D^1_{\rm min}$, $D^2_{\rm min}$ and $\bar{\rho}$. The first two of them are used to randomize the selection of edges during the construction of spanning trees in the initialization and, respectively, offspring generation steps. The role of $D^1_{\rm min}$ and $D^2_{\rm min}$

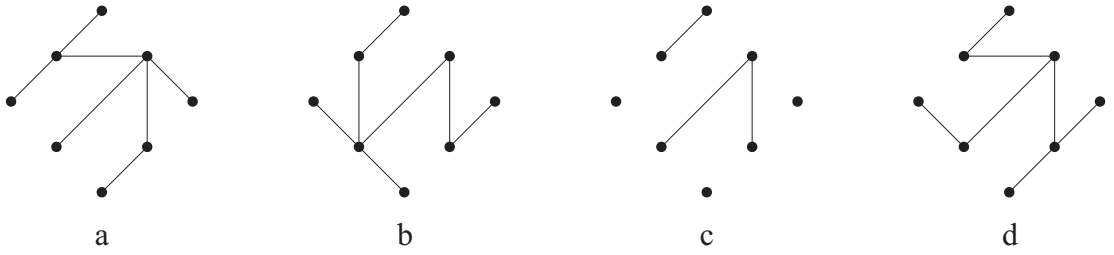

Figure 1. Illustration of crossover operation: (a)–(b) parents; (c) common edges; (d) offspring

is to identify those spanning trees which are too similar to at least one member of the population. Such spanning trees are not included in Π. The parameter $\bar{\rho}$ is used to stop the initialization process when it becomes difficult to generate a diverse population of the required size. This may happen only if the graph G is very small and sparse. In the description of the algorithm, the best solution is denoted by T^*. Initially, T^* is the best spanning tree in the population constructed in Step 1 of HGA. In the evolution phase, T^* is updated each time a better spanning tree is found by the local search procedure LS. This procedure is an implementation of the 1-opt local improvement method. It goes as follows.

LS(T)

1. Initialize γ with 0. For each $e \in E$, compute $Q_e = c_e + \sum_{g \in E(T) \setminus \{e\}} c_{eg}$, where $E(T)$ denotes the edge set of the tree T as before.
2. Running through all pairs of edges $e \in E(T)$ and $g \in E \setminus E(T)$ such that $T(e,g) \in N(T)$, compute $\delta' = Q_g - Q_e - c_{eg}$. If $\delta' < 0$, then perform the following actions. Exchange the edge e in $E(T)$ with the edge g. Set $\gamma := 1$, $Q_e := Q_e + c_{eg}$, $Q_g := Q_g - c_{eg}$ and $Q_h := Q_h - c_{eh} + c_{gh}$ for each $h \in E \setminus \{e, g\}$. Go to 3. If however $\delta' \geqslant 0$, then repeat 2 for the next pair (e, g).
3. If $\gamma = 0$, then return with T. Otherwise, set $\gamma := 0$ and go to 2 (thus, start the search for an improving exchange from the beginning).

At each iteration of LS, the neighborhood $N(T)$ of the current tree T is explored. The algorithm tries to exchange each tree edge with each non-tree edge provided such an exchange does not create a cycle. Different ways to accomplish this are possible. Our strategy is to make the tree T rooted at any vertex and, for any edge (u, v) outside T, using the obtained information to effectively enumerate the edges on the path from u to v in T. In order to describe this process more formally, we need a couple of notations. For a non-root vertex $i \in V$, let Φ_i be the father of i in the rooted tree T. Let L_i stand for the level of vertex i in T. The procedure, dubbed NS (Neighborhood Search), consists of the following two steps.

NS

1. Make the tree T rooted at any fixed vertex.
2. For each edge $g = (u, v) \in E \setminus E(T)$ do the following:
   2.1. Initialize i with u and j with v.
   2.2. Check whether $L_i \geqslant L_j$. If so, then set $w := i$, $i := \Phi_i$ and $e := (i, w)$. If not, then set $w := j$, $j := \Phi_j$ and $e := (j, w)$.
   2.3. Compute δ' for the edges e and g. If $i \neq j$, then return to 2.2.

The described procedure is embedded in Step 2 of LS where, for each pair of edges (e, g) identified by NS, the value of $\delta(T, e, g)$ is calculated and, depending on the result, appropriate actions are taken. The behavior of NS is illustrated in Figure 2 where (u, v) is the edge to be added to T. Notice that the order of assignment statements in Step 2.2 of NS is fixed. It is not hard to see that Step 1 of NS (i.e., getting Φ_i and L_i for non-root vertices $i \in V$) requires $O(n)$ time. The complexity of Step 2 of NS is $O(nm)$. In the case of dense graphs, it amounts to $O(n^3)$. Thus the time of making the tree rooted is negligible compared with the overall time taken by NS. As it can be seen from the description of LS, exploration of the neighborhood is restarted after each update of the current spanning tree. This means that a new rooted tree needs to be built (or the existing one reconfigured). However, as just remarked, this operation is computationally very cheap.

In closing this section, we note that the pure genetic algorithm can be obtained simply by removing Step 4 from HGA. Moreover, to keep a higher level of intensification in the search process it is useful to make the selection of an edge in Step 3.3 deterministic. The modification is to choose an edge with the

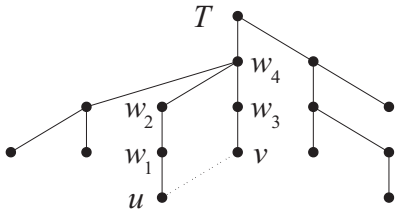

Figure 2. Illustration for Step 2 of LS: the edges for removal from T are considered in the order (u, w_1), (w_1, w_2), (v, w_3), (w_2, w_4), (w_3, w_4)

smallest value of Q_e. This can be done by setting $\bar{z}_2$ to 1 in Step 3.3 of HGA.

4. Iterated tabu search

At the core of our third algorithm is an adaptation of the tabu search technique to the quadratic minimum spanning tree problem. In order to obtain better solutions, we apply the tabu search procedure repeatedly. To get a starting spanning tree for the next iteration, a solution perturbation mechanism is used. The main procedure of the iterated tabu search algorithm can be stated as follows.

ITS

1. Randomly generate a spanning tree $T = (V, E(T))$. Initialize T^* with T and F^* with $F(T)$. For each $e \in E$, compute $Q_e = c_e + \sum_{g \in E(T) \setminus \{e\}} c_{eg}$.
2. Randomly draw an integer I between $I_{\min}$ and $I_{\max}$. Execute the tabu search procedure TS(T, T^*, F^*, I).
3. Check if the termination condition is satisfied. If so, then go to 5. Otherwise proceed to 4.
4. Randomly draw integers $\bar{p}$ and $\bar{q}$ in $\{a_1, a_1 + 1, \ldots, a_2\}$ and $\{b_1, b_1+1, \ldots, b_2\}$, respectively; here a_1, b_1 and b_2 are constants, whereas a_2 is an integer chosen at random from the interval $[n\lambda_1, n\lambda_2]$. Apply GST(T, $\bar{p}$, $\bar{q}$). Return to 2.
5. Stop with the spanning tree T^* of value F^*.

In Step 1 of ITS, an initial spanning tree is generated. For this purpose, the same algorithm as in MSA is used. This step also initializes the variables Q_e, $e \in E$. They are needed to efficiently compute the decrease (or increase) in the objective function value that will result from exchanging an edge in $E(T)$ with an edge in $E \setminus E(T)$. Step 2 of ITS invokes the tabu search procedure TS. The input to TS includes the number of iterations for a tabu search run, denoted by I. This number is taken from the interval $[I_{\min}, I_{\max}]$, where $I_{\min}$ and $I_{\max}$ are the parameters of ITS. Other parameters are used in Step 4 to get the values of $\bar{p}$ and $\bar{q}$. These values are submitted to the solution perturbation procedure, named GST (Get Start Tree). We shall discuss the meaning of $\bar{p}$ and $\bar{q}$ later in this section. Making I, $\bar{p}$ and $\bar{q}$ variable strengthens the search diversification capabilities of the algorithm.

Next, we will present the main ingredients of ITS, namely, TS and GST. In the description of TS given below, τ_e, $e \in E$, are the tabu values and $\bar{\tau}$ is the tabu tenure, which in our experiments was set to $\min(10, m/4)$.

TS(T, T^*, F^*, I)

1. Initialize i and τ_e, $e \in E$, with 0. Set $f := F(T)$.
2. Increment i by 1. Set $\delta^* := \infty$ and $\mu := 0$.
3. Running through all pairs of edges $e \in E(T)$ and $g \in E \setminus E(T)$ such that $T(e, g) \in N(T)$, perform the following steps.
   3.1. Compute $\delta' = Q_g - Q_e - c_{eg}$. If $\delta' > \delta^*$, then go to 3.3. Otherwise check whether $f + \delta' < F^*$. If so, then set $\delta^* := \delta'$, $h := e$, $d := g$, $\mu := 1$ and go to 4. If not, then proceed to 3.2.
   3.2. If at least one of τ_e, τ_g is positive, then go to 3.3. Otherwise check whether $\delta' < \delta^*$. If so, then set $\delta^* := \delta'$, $h := e$, $d := g$ and $k := 1$. If not, then increment k by 1 and set $h := e$, $d := g$ with probability $1/k$.
   3.3. Repeat 3.1 and 3.2 for the next pair (e, g), if any.
4. Exchange the edge h in $E(T)$ with the edge d. Set $f := f + \delta^*$. Update Q_e, $e \in E$ (in the same way as it is done in Step 2 of the LS procedure). If $\mu = 1$, then go to 5. Otherwise go to 6.
5. Apply the local search procedure LS to the tree T. Let T also denote the tree returned by it. Set $T^* := T$, $F^* := F(T)$ and $f := F^*$.
6. If $i = I$, then return. Otherwise, decrement each positive τ_e, $e \in E$, by 1, set $\tau_h := \bar{\tau}$, $\tau_d := \bar{\tau}$ and go to 2.

As can be seen, TS consists of the initialization part and the loop comprising Steps 2–6, which is generally executed I times. It is, however, possible to terminate TS prematurely in Step 6 after the time allotted for the ITS run has expired. The most costly part of TS is Step 3 where the neighborhood of the current spanning tree T is explored. Like in the LS case, this is done using the procedure NS. For a tree $T(e, g)$ obtained while running NS, the condition $F(T(e, g)) = f + \delta(T, e, g) < F^*$ is checked. If it is satisfied, then the indicator μ is set to 1. Obviously,

$\mu = 1$ means that a new best solution in the ITS run has been found. Only in this case the local search procedure is applied. It is essentially the same as LS used in HGA. The only difference is that now the initialization of Q_e, $e \in E$, in Step 1 of LS is not required. Indeed, the values of Q_e are computed in Step 1 of ITS and are maintained by both TS and GST.

The tabu search is restarted from a spanning tree produced by the procedure GST implementing a strategy for perturbation of the current spanning tree T. Besides T, the input to GST includes parameters $\bar{p}$ and $\bar{q}$. The parameter $\bar{p}$ is the number of edges to be removed from T. The procedure is randomized. An edge for removal and a non-tree edge replacing it are picked at random from the candidate list of length at most $\bar{q}$. This list is constructed by including edge pairs (e, g) of type (tree edge, non-tree edge) for which the values $\delta(T, e, g)$ are smallest. An additional requirement is that each edge can be chosen (to remove from T or to add to T) at most once. The perturbation algorithm proceeds as follows.

GST($T, \bar{p}, \bar{q}$)

1. Set $p := 0$ and $\tilde{E} := \emptyset$.
2. If $B := \{(e_1, e_2) \mid e_1 \in E(T) \setminus \tilde{E}, e_2 \in E \setminus (E(T) \cup \tilde{E}), T(e_1, e_2) \in N(T)\}$ is empty, then return. Otherwise, form a set B' of $q = \min(\bar{q}, |B|)$ edge pairs $(e_1, e_2) \in B$ such that $\delta(T, e_1, e_2) \leqslant \delta(T, g_1, g_2)$ for each $(e_1, e_2) \in B'$ and each $(g_1, g_2) \in B \setminus B'$. Randomly select $(h_1, h_2) \in B'$.
3. Exchange the edge h_1 in $E(T)$ with the edge h_2. Add h_1 and h_2 to $\tilde{E}$. Update Q_e, $e \in E$.
4. Increment p by 1. If $p < \bar{p}$, then go to 2. Otherwise return.

In Step 2 of GST, the set B is searched in order to retrieve edge pairs with the smallest δ values. For this purpose, we again use the procedure NS stated in the previous section.

5. Experimental results

In this section, we present some computational results in order to evaluate the performance of the algorithms we have described. All the algorithms have been coded in the C programming language and all the tests have been carried out on a PC with an Intel Core 2 Duo CPU running at 3.0GHz. The sources are publicly available at http: //www.soften.ktu.lt/~gintaras/qmstp.html. As a testbed for the algorithms, the problem instances introduced by Cordone and Passeri [3] were considered. We will provide results only for the largest instances in their test set. More specifically, in the main experiments, we used graphs ranging in size from 40 to 50 vertices.

To compare HGA with the pure genetic algorithm, we performed tests also on a few of the graphs of order 30 and 35. The density of each graph in the testbed is one of the following: 33%, 67% and 100%.

Based on the results of preliminary computational experiments with algorithms, we have fixed the values of their parameters: for MSA, $\alpha = 0.95$, $t_0 = 0.0001$, $R_0 = 100$; for HGA, *pop_size*=100, $\bar{z}_1 = 20$, $\bar{z}_2 = 10$, $D^1_{\min} = D^2_{\min} = 6$, $\bar{\rho} = 1000$; for ITS, $a_1 = 10$, $\lambda_1 = 0.1$, $\lambda_2 = 1$, $b_1 = 5$, $b_2 = 300$, $I_{\min} = 100$, $I_{\max} = 200$. For the pure genetic algorithm, the same values as in the case of HGA were used, except that the parameter $\bar{z}_2$ was fixed at 1. The results presented in this section were obtained by performing 10 runs of each algorithm on each problem instance in the test set. In the first experiment, the cutoff time for each run was 10 seconds. The results of MSA, HGA and ITS are summarized in Tables 1 and 2. The first column of these and subsequent tables represents the problem instances. In the name of an instance, the integer following "n" indicates the graph order (the number of its vertices), whereas the integer following "m" shows the number of its edges. For example, n40_m257_1 denotes the first instance whose graph has 40 vertices and 257 edges. The second column of Table 1 gives the best objective function values, which were reported by Cordone and Passeri [3]. The third (respectively, fourth) column shows the difference between the value of the best solution out of 10 runs (respectively, the average value of 10 solutions) found by MSA and the value displayed in the second column. The rest of Table 1 gives these differences for HGA and ITS. Table 2 includes the results for the 24 largest instances only. The second and third columns of this table, for each instance, provide the shortest (out of 10 runs) and, respectively, the average CPU time taken by MSA to first find a solution that is best in the run. The remaining columns report the CPU time taken by HGA and ITS. The results, averaged over all tested instances, are presented in the last row of each table.

Inspection of Table 1 reveals that ITS performs considerably better than the other tested algorithms. In particular, ITS is the only algorithm that was able to find the best known solutions for all benchmark instances used in our study. For comparison, MSA and HGA failed to achieve the best objective function values in 18 and, respectively, 5 cases (out of 36). From the average results, we can see that there is no instance for which ITS is inferior to HGA. A similar observation can be made when comparing HGA and

Table 1. Results of running MSA, HGA and ITS on the chosen dataset (time limit=10s)

Instance	Best known value	Solution difference					
		MSA		HGA		ITS	
		Min	Ave	Min	Ave	Min	Ave
n40_m257_1	5945	0	3.6	0	0	0	0
n40_m257_2	56237	0	27.0	0	0	0	0
n40_m257_3	6925	0	0	0	0	0	0
n40_m257_4	57874	0	24.1	0	0	0	0
n40_m522_1	5567	0	22.0	0	2.4	0	0
n40_m522_2	51851	266	538.3	0	32.8	0	6.0
n40_m522_3	6456	0	21.8	0	0	0	0
n40_m522_4	53592	11	190.1	0	7.7	0	5.4
n40_m780_1	5368	5	47.2	0	10.9	0	0
n40_m780_2	49817	452	1184.7	0	51.6	0	3.3
n40_m780_3	6208	0	17.9	0	0	0	0
n40_m780_4	51229	0	778.5	0	176.1	0	0
n45_m326_1	7521	0	6.5	0	0	0	0
n45_m326_2	70603	0	66.7	0	0	0	0
n45_m326_3	8720	0	0	0	0	0	0
n45_m326_4	72676	0	109.7	0	0	0	0
n45_m663_1	7161	37	78.5	0	24.2	0	8.4
n45_m663_2	66889	0	826.6	0	301.4	0	0
n45_m663_3	8225	0	14.9	0	1.8	0	0
n45_m663_4	68737	0	865.0	0	248.3	0	0
n45_m990_1	6944	11	95.6	6	14.1	0	1.9
n45_m990_2	64840	631	1289.3	25	242.3	0	33.2
n45_m990_3	7827	5	44.6	0	0	0	0
n45_m990_4	66508	530	1262.3	50	186.0	0	41.1
n50_m404_1	9393	0	22.9	0	2.4	0	0
n50_m404_2	88942	349	458.8	0	34.9	0	0
n50_m404_3	10717	0	8.2	0	0	0	0
n50_m404_4	91009	0	285.8	0	82.5	0	0
n50_m820_1	8958	63	112.1	0	32.8	0	0
n50_m820_2	84020	759	1437.0	0	361.5	0	0
n50_m820_3	10100	2	34.8	0	0	0	0
n50_m820_4	86231	685	1257.8	0	398.0	0	0
n50_m1225_1	8713	87	209.1	8	41.2	0	6.0
n50_m1225_2	81858	1459	2361.5	27	267.2	0	112.1
n50_m1225_3	9836	8	85.3	0	0.6	0	0.6
n50_m1225_4	83838	548	1604.4	0	67.5	0	4.3
Average		164.1	427.6	3.2	71.9	0	6.2

MSA. In all cases, the average values of F obtained by HGA are smaller than or equal to those obtained using MSA. Thus, HGA is the second best algorithm in our tests.

From Table 2, we see that, considering the CPU time required by the algorithms to find the best solution in the run, HGA is comparable to ITS. These algorithms are significantly faster than MSA.

In Table 3, we compare the performance of the pure genetic algorithm, denoted as GA, with that of HGA. The results are presented for a subset of the full set of the problem instances. This table is organized similarly to Table 1. We conclude from the table that the pure genetic algorithm is beaten by HGA. It was able to deliver solutions of best quality only for 3 instances out of 15. By analyzing the results in Tables 1 and 3, we find that also MSA is better than GA. Actually, MSA dominates over GA almost completely. The only exceptions are n45_m990_3 ($Min = 5$, $Ave = 22.9$ for GA) and n50_m404_2

Table 2. Time to the best solution in the run for graphs of order 45 and 50 (in seconds)

Instance	MSA		HGA		ITS	
	Max	Ave	Max	Ave	Max	Ave
n45_m326_1	9.0	4.3	0.3	0.2	0.2	0.1
n45_m326_2	7.2	1.7	0.3	0.2	0.5	0.2
n45_m326_3	3.6	1.0	0.4	0.2	<0.1	<0.1
n45_m326_4	7.5	3.0	1.1	0.5	1.0	0.4
n45_m663_1	6.4	3.3	8.1	2.8	9.1	4.0
n45_m663_2	7.3	3.7	9.2	2.8	1.9	0.6
n45_m663_3	6.2	3.3	5.5	2.7	0.6	0.2
n45_m663_4	7.1	6.4	5.0	2.8	1.8	0.7
n45_m990_1	9.6	4.9	8.2	2.5	9.3	3.9
n45_m990_2	10.0	6.7	8.2	3.4	7.1	2.3
n45_m990_3	9.4	7.0	0.6	0.4	0.9	0.4
n45_m990_4	9.8	5.1	9.2	3.2	9.7	5.2
n50_m404_1	4.7	2.0	4.5	0.8	0.7	0.2
n50_m404_2	9.6	4.5	5.3	1.9	0.5	0.2
n50_m404_3	8.6	4.3	0.1	0.1	0.2	<0.1
n50_m404_4	9.7	4.7	0.9	0.3	0.9	0.2
n50_m820_1	8.6	3.8	7.0	2.7	8.4	2.2
n50_m820_2	9.5	6.3	9.9	4.2	7.9	3.7
n50_m820_3	8.7	4.6	0.5	0.3	0.3	0.2
n50_m820_4	9.5	6.4	7.6	2.9	7.2	2.0
n50_m1225_1	3.1	2.8	8.0	3.6	9.4	5.0
n50_m1225_2	3.3	2.8	8.7	4.5	9.8	5.1
n50_m1225_3	3.5	3.0	9.0	4.3	8.2	3.9
n50_m1225_4	3.0	2.7	3.7	1.9	8.5	4.4
Average	7.3	4.1	5.1	2.0	4.3	1.9

Table 3. Comparison of GA and HGA (time limit=10s)

Instance	Best known value	Solution difference			
		GA		HGA	
		Min	Ave	Min	Ave
n30_m143_1	3205	0	2.8	0	0
n30_m291_1	2998	28	43.1	0	0
n30_m435_1	2874	31	60.0	0	6.6
n35_m196_1	4474	0	15.4	0	0
n35_m398_1	4147	20	61.5	0	0
n35_m595_1	4000	102	132.2	0	0
n40_m257_1	5945	16	36.1	0	0
n40_m522_1	5567	60	91.5	0	2.4
n40_m780_1	5368	130	175.3	0	10.9
n45_m326_1	7521	0	42.3	0	0
n45_m663_1	7161	154	180.1	0	24.2
n45_m990_1	6944	174	235.3	6	14.1
n50_m404_1	9393	37	126.5	0	2.4
n50_m820_1	8958	175	259.2	0	32.8
n50_m1225_1	8713	240	324.5	8	41.2
Average		77.8	119.1	0.9	9.0

Table 4. Results of longer runs of MSA, HGA and ITS (time limit=180s)

Instance	Solution difference					
	MSA		HGA		ITS	
	Min	Ave	Min	Ave	Min	Ave
n40_m522_2	0	101.2	0	16.4	0	0
n40_m522_4	11	50.7	0	6.6	0	0
n40_m780_2	0	127.7	0	46.9	0	0
n45_m663_1	17	28.3	0	20.9	0	0
n45_m663_2	0	146.9	0	237.2	0	0
n45_m663_3	0	0	0	0	0	0
n45_m663_4	0	181.4	0	229.9	0	0
n45_m990_1	1	33.8	6	11.7	0	0
n45_m990_2	274	525.8	25	178.5	0	0
n45_m990_3	0	0	0	0	0	0
n45_m990_4	50	421.1	0	154.1	0	0
n50_m404_1	0	0	0	0	0	0
n50_m404_2	0	34.9	0	0	0	0
n50_m404_3	0	0	0	0	0	0
n50_m404_4	0	0	0	0	0	0
n50_m820_1	0	31.3	0	27.0	0	0
n50_m820_2	44	565.7	0	235.0	0	0
n50_m820_3	0	0.2	0	0	0	0
n50_m820_4	104	677.7	0	367.8	0	0
n50_m1225_1	59	87.7	0	36.3	0	0
n50_m1225_2	379	738.2	27	201.0	0	0
n50_m1225_3	0	18.5	0	0.6	0	0
n50_m1225_4	238	712.1	0	32.1	0	0
Average	51.2	194.9	2.5	78.3	0	0

($Min = 0$,$Ave = 994.6$ for GA) instances.

In our second experiment, we allowed the algorithms to run longer. We have limited the computation time to 3 minutes (180 seconds) per run. We tested MSA, HGA and ITS on all instances of size 50 as well as on 8 densest instances of size 45 and 3 instances of size 40 for which ITS failed to find the best solutions in at least one of the 10-second runs. The results are summarized in Tables 4 and 5. As Table 4 shows, ITS again outperforms HGA and again HGA is ranked ahead of MSA. We can notice that ITS succeeded in finding the best solutions in all the runs. Meanwhile, the HGA implementation was still unable to reach the best known minima for 3 instances. In addition, the average performance of HGA in a number of test cases was not good enough. It can be observed that the difference between the quality of solutions produced by HGA and those produced by MSA becomes smaller as the amount of time allotted for each run of the algorithm increases. As seen in Table 4, MSA performed better than HGA for n45_m663_2 and n45_m663_4 and was not dominated by HGA in the case of the n45_m990_1 instance.

The results in Table 5 indicate that the CPU time taken to find the best solution is smaller for ITS than for the other two approaches. In fact, a time limit of 60 seconds was sufficient for ITS to deliver the best spanning trees in all the runs for each benchmark instance except n50_m1225_2. Again, as in the first experiment, MSA is the slowest of the examined algorithms.

6. Conclusions

In this paper we have presented simulated annealing (MSA), genetic (HGA) and tabu search (ITS) algorithms for the quadratic minimum spanning tree problem. We conducted computational experiments to evaluate the performance of these algorithms. Their results demonstrate that ITS is superior to HGA as well as to MSA in terms of both solution quality and computation time. In particular, ITS was the only one of the three approaches which was able to find the best known solutions for all QMSTP instances we have tried. HGA was the second best algorithm in the ex-

Table 5. Time to the best solution in the run (in seconds)

Instance	MSA		HGA		ITS	
	Max	Ave	Max	Ave	Max	Ave
n40_m522_2	152.7	92.9	83.3	9.8	10.2	3.6
n40_m522_4	160.6	54.5	62.7	8.7	24.9	7.8
n40_m780_2	155.2	82.1	71.7	8.8	21.3	5.7
n45_m663_1	151.1	109.0	98.0	22.2	24.1	11.2
n45_m663_2	163.3	63.8	155.5	35.1	1.9	0.6
n45_m663_3	98.3	35.9	14.1	4.0	0.6	0.2
n45_m663_4	159.7	63.3	37.9	6.4	1.8	0.7
n45_m990_1	161.9	92.2	108.2	14.1	50.5	21.0
n45_m990_2	136.0	95.4	64.9	16.6	30.3	8.9
n45_m990_3	159.3	59.1	0.6	0.4	0.9	0.4
n45_m990_4	155.4	77.8	20.7	5.5	48.3	20.9
n50_m404_1	67.5	36.5	27.2	3.5	0.7	0.2
n50_m404_2	170.4	44.8	11.0	2.8	0.5	0.2
n50_m404_3	30.5	11.3	0.1	0.1	0.2	<0.1
n50_m404_4	172.8	65.7	58.1	16.0	0.9	0.2
n50_m820_1	160.2	70.4	42.0	8.1	8.4	2.2
n50_m820_2	174.9	119.2	81.2	21.0	7.9	3.7
n50_m820_3	167.7	45.1	0.5	0.3	0.3	0.2
n50_m820_4	155.4	82.7	49.9	8.2	7.2	2.0
n50_m1225_1	179.9	100.2	78.9	12.9	46.5	13.3
n50_m1225_2	175.0	103.3	173.2	36.8	151.7	50.1
n50_m1225_3	172.4	99.7	9.0	4.3	30.0	7.6
n50_m1225_4	164.8	78.1	25.2	4.1	41.8	8.3
Average	149.8	73.2	55.4	10.9	22.2	7.3

periments. However, the success of HGA strongly depends on the use of a local search procedure. The genetic algorithm without local search performed rather poorly. This version of GA is not competitive with the other algorithms (MSA, HGA and ITS) involved in the comparison. On the basis of our results, it may be speculated that also other approaches incorporating local search could be promising techniques for the QMSTP. Such approaches include VNS (Variable Neighborhood Search) and GRASP (Greedy Randomized Adaptive Search Procedure) with path-relinking.

References

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

[2] **R. Cordone, G. Passeri**. Heuristic and exact approaches to the quadratic minimum spanning tree problem. *In: Seventh Cologne–Twente Workshop on Graphs and Combinatorial Optimization (CTW08), May 13–15, 2008, Gargnano, Italy, Università degli Studi di Milano*, 2008, pp. 52–55.

[3] **R. Cordone, G. Passeri**. The quadratic minimum spanning tree problem (QMSTP). http://homes.dsi.unimi.it/~cordone/research/qmst.html. Accessed 27 October 2010

[4] **A. Duarte, R. Martí**. Tabu search and GRASP for the maximum diversity problem. *European Journal of Operational Research*, 2007, *Vol.*178, 71–84.

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

[6] **F. Glover, Z. Lü, J.-K. Hao**. Diversification-driven tabu search for unconstrained binary quadratic problems. *4OR–A Quarterly Journal of Operations Research*, 2010, *Vol.*8, 239–253.

[7] **J. Gottlieb, B.A. Julstrom, G.R. Raidl, F. Rothlauf**. Prüfer numbers: a poor representation of spanning trees for evolutionary search. *In: Proceedings of the Genetic and Evolutionary Computation Conference, San Francisco, CA, USA, Morgan Kaufmann Publishers*, 2001, pp. 343–350.

[8] **J.B. Kruskal**. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 1956, *Vol.*7, 48–50.

[9] **R. Martí, M. Gallego, A. Duarte**. A branch and

bound algorithm for the maximum diversity problem. *European Journal of Operational Research*, 2010, *Vol*.200, 36–44.

[10] **T. Öncan, A.P. Punnen**. The quadratic minimum spanning tree problem: a lower bounding procedure and an efficient search algorithm. *Computers and Operations Research*, 2010, *Vol*.37, 1762–1773.

[11] **G. Palubeckis**. Iterated tabu search for the unconstrained binary quadratic optimization problem. *Informatica*, 2006, *Vol*.17, 279–296.

[12] **G. Palubeckis**. Iterated tabu search for the maximum diversity problem. *Applied Mathematics and Computation*, 2007, *Vol*.189, 371–383.

[13] **G. Palubeckis**. Solving the weighted Max-2-SAT problem with iterated tabu search. *Information Technology and Control*, 2008, *Vol*.37, 275–284.

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

[15] **P.M. Pardalos O.A. Prokopyev, O.V. Shylo, V.P. Shylo**. Global equilibrium search applied to the unconstrained binary quadratic optimization problem. *Optimization Methods and Software*, 2008, *Vol*.23, 129–140.

[16] **H. Prüfer**. Neuer beweis eines satzes über permutationen. *Archiv für Mathematik und Physik*, 1918, *Vol*.27, 742–744.

[17] **G.R. Raidl, B.A. Julstrom**. Edge sets: an effective evolutionary coding of spanning trees. *IEEE Transactions on Evolutionary Computation*, 2003, *Vol*.7, 225–239.

[18] **S.-M. Soak, D.W. Corne, B.-H. Ahn**. The edge-window-decoder representation for tree-based problems. *IEEE Transactions on Evolutionary Computation*, 2006, *Vol*.10, 124–144.

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

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

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

Received August 2010.

DOI: 10.5755/j01.itc.39.4.12389