Available online at www.sciencedirect.com
ScienceDirect
Applied Mathematics and Computation 188 (2007) 1071–1086
APPLIED MATHEMATICS AND COMPUTATION
www.elsevier.com/locate/amc

Remarks on the application of genetic algorithm and tabu search method to nonlinear spanning tree problems

El Bekkaye Mermri *, Hideki Katagiri, Masatoshi Sakawa, Kosuke Kato

Graduate School of Engineering, Hiroshima University, Higashi-Hiroshima 739-8527, Japan

Abstract

In this paper we describe and discuss the application of some genetic algorithm approaches, using Prüfer numbers for their encoding methods, for solving nonlinear minimum spanning tree problems. Next, we develop an algorithm based on tabu search method to solve a class of these problems. To evaluate and compare the performances of the proposed TS algorithm with GAs described in this paper, some computational experiments are provided.

Keywords: Spanning tree; Genetic algorithm; Prüfer number; Tabu search

1. Introduction

Let $G = (V, E)$ be a connected undirected graph, where V and E denote sets of n vertices and m edges, respectively. A spanning tree is a connected sub-graph of G which contains all the vertices, but no cycle. Spanning tree problems arise in many applications of network and telecommunication designs. The linear Minimum Spanning Tree (MST) problem is to find, in edge-weighted graph, a spanning tree of minimal total weight. The efficient polynomial-time algorithms to solve MST problems have been developed by Kruskal [19] and Prim [21]. Because of its importance in the applications, this problem has been the interest of many researchers, for instance, see the book [2] on algorithms and applications of MST problem, and see the surveys [3,17] on the existing algorithms for solving MST problem. All these algorithms, even for large problems, can produce the optimal solution in a timely enough manner. For other MST problems, this is not always the case. For some problems there are no practical algorithms known. For these kinds of problems, it is generally accepted to be solved by an appropriate heuristic algorithm, such as genetic algorithms, tabu search method, ant colony, GRASP, variable neighborhood search, among others.

In this paper we describe and discuss the application of some genetic algorithm (GA) approaches, using Prüfer numbers for encoding trees, to solve nonlinear MST problems. Next, we propose an algorithm based

* Corresponding author.
E-mail address: mermri@msl.sys.hiroshima-u.ac.jp (E.B. Mermri).

0096-3003/$ - see front matter
doi:10.1016/j.amc.2006.06.111

on tabu search (TS) method to solve a class of nonlinear MST problems; MST problems with a quadratic objective function (q-MST) and MST problems with objective functions depending on linear and quadratic terms. As an example, the later problem arises in a solution method for a fuzzy random minimum spanning tree problem formulated by Mermri et al. [20]. The q-MST problem has been studied by Zhou et al. [26] by using GA with Prüfer number encoding method. The authors compared their approach with two heuristics methods and concluded that it is effective in solving q-MST problem.

One of the classical theorems in enumeration is Cayley's theorem [5], which says that in a complete undirected graph G with n vertices there are n^{n-2} distinct labeled trees. Prüfer [23] provided a constructive proof of Cayley's theorem by establishing a one to one correspondence between spanning trees of the graph G and the set of all permutations of $n-2$ digits. Prüfer numbers are an $n-2$ digit sequences, where the digits are n different numbers.

Thus, when applying GAs to spanning tree problems, several researchers used Prüfer numbers for encoding trees (see [1,8,18,22,26,27], for instance). Some of them report high, and others report low, performance. Some other works on the investigation of the use of Prüfer numbers in the context of evolutionary search, reported that in general this encoding method has low locality problem (see [24,14], for instance). As it is discussed in Section 3 of this paper; the GA method using Prüfer numbers for its solution encoding to solve nonlinear MST problems, has low locality problem and the schema principal is not applicable.

In Section 2 we present some of tree representation methods for GA, these include characteristic vector, predecessor and Prüfer representations. Then, we construct a new algorithm to decode a Prüfer number into a spanning tree in a time of $O(n)$. The known decoding algorithm requires a time of $O(n \log n)$ (see [14,22,24], for instance). In Section 3 we give a brief description of GA features and discuss GA approaches using Prüfer number for their encoding methods. In Section 4 we develop a TS algorithm for some nonlinear MST problems. We also provide a motivation to explain how this TS approach can work effectively, in particular to solve q-MST. Section 5 is devoted to give some computational experiments to test and compare the performances of GAs and TS method developed in this paper. Finally, some conclusions are given in Section 6.

2. Tree representations

Let G be a complete undirected graph with n vertices and m edges, $m = n(n-1)/2$. One of the main issues that needs to be addressed when trying to implement a GA for a problem is the definition of an appropriate encoding (or representation) of solutions. For problems that have spanning trees in their solutions, in order for a GA to function effectively, the encoding should satisfy the following:

(i) It should be capable to represent all possible spanning trees of the given graph.
(ii) The encoding should be unbiased in the sense that all trees are equally represented.
(iii) When crossover and mutation operations are performed, the resulting chromosome represents exactly a spanning tree.
(iv) Schemata principals should be applicable to the encoding structure.
(v) The encoding should posses a locality in the sense that small changes in the chromosome result small changes in the tree.

In this section we provide a description of some possible tree encoding methods.

2.1. Characteristic vector representation

A tree T can be represented by a 0–1 m-vector, V. If an edge of index k lies in the tree T then the kth component of V, V_k, takes the value 1, otherwise V_k takes the value 0. Now, consider an initial parent V which represents a spanning tree. The crossover or mutation operations may produce changes on some elements of the vector V, unfortunately the resulting vector is unlikely a spanning tree. In a complete graph there are 2^m possible 0–1 m-vectors, and n^{n-2} possible spanning trees, as we will see in Section 2.3. Most of the representing solutions are not trees, and the probability that a random representation corresponds to a spanning tree is very small as n increases, which is equal to $\frac{n^{n-2}}{2^{n(n-1)/2}}$.

2.2. Predecessor representation

A rooted tree is a tree with a specified vertex as a root vertex of the tree. The root will often be at the top of the tree, and the tree will appear to grow downwards. Then a rooted tree, with a rooted vertex r, can be encoded by maintaining for each vertex labeled i its predecessor $p_r(i) = j$, where j denotes the label of the first vertex in the path from i to the root vertex r. As a convention, set $p_r(r) = r$. Hence every rooted tree can be represented by a unique n-vector p_r where $p_r(i)$, $i = 0, \ldots, n-1$, are numbers between 0 and $n-1$. Conversely, let $p_r = [p_0, p_1, \ldots, p_{n-1}]$ be a predecessor representation, then its corresponding tree is represented by the set of edges of the form $(i, p_r(i))$, $i = 0, \ldots, n-1$. For one tree T with n vertices we can derive n rooted trees. Then each tree is represented by n of such encoding numbers. This encoding is unbiased, and covers the space of all possible trees. However, there are n^n such representations, each tree is represented n times, and there are only n^{n-2} possible trees. The probability that a random encoding number p_r results a spanning tree is equal to $1/n$. This is an improvement over the characteristic vector representation, but still allows nonspanning trees being generated. Fig. 1 shows an example of an encoding number and its decoded tree and vice versa, where the root is the vertex labeled 4.

Note that the graph in Fig. 1 is undirected. The arrows indicates the directions to the root vertex 4. As an example of an encoding number that results a nonspanning tree we may consider the following representation: $p_r = [1, 4, 4, 0, 3, 2, 2]$.

2.3. Prüfer number

One of the classical theorems in enumeration is Cayley's theorem [5], which says that in a complete undirected graph with n vertices there are n^{n-2} distinct labeled trees. Prüfer provided a constructive proof of Cayley's theorem by establishing a one to one correspondence between such spanning trees and the set of all permutations of $n-2$ digits [23]. Prüfer numbers are $n-2$ digit sequences, where the digits are numbers between 0 and $n-1$.

Let G be a complete undirected graph with n vertices, and let d be a positive integer. In what follows we need the following definitions.

- A vertex v in G is called of degree d if it is a common vertex of d edges.
- A vertex of degree 1 is called a *leaf vertex*.
- A Prüfer number, P, is an $n-2$ digit sequence: $P = [p_0, p_1, \ldots, p_{n-3}]$, where the digits p_i, $0 \leqslant i \leqslant n-3$, are numbers in $\{0, 1, \ldots, n-1\}$.
- Let P be a Prüfer number. The set of numbers in $\{0, 1, \ldots, n-1\}$ which are not digits part of P is called child of P and denoted by R.

The relationship between Prüfer numbers and spanning trees are given by the following algorithms.

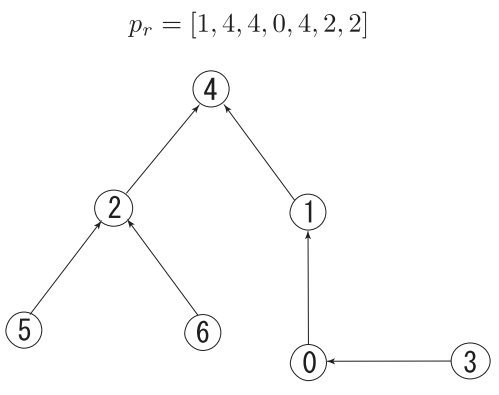

Fig. 1. Predecessor representation and its tree.

2.3.1. Encoding algorithm

This algorithm shows how to construct a Prüfer number from a given spanning tree, T. For the algorithm to be nontrivial, the tree T should have at least two edges, $n \geqslant 3$.

Algorithm 1. Construct P by appending digits to the right; thus, P is constructed from left to right.

(1) Let i be the smallest labeled vertex of degree 1 in T, and let p be the vertex antecedent of i. Then set p to the end of P.
(2) Remove the edge (i,p) from the tree T, and update T to $T\backslash\{(i,p)\}$.
(3) While the tree T has two or more edges return to step (1).

As an example on how this algorithm works, consider the tree shown in Fig. 1. The smallest labeled vertex of degree 1, is the vertex numbered 3. We therefore select 1 as the first digit of P, $P = [1]$. We then remove the edge $(1,3)$ from T, and vertex 1 becomes the smallest labeled vertex of degree 1. So the next digit of P is 6, $P = [1,6]$. We continue the process until there are two vertices left, $\{2,6\}$. Then we stop with $P = [1,6,0,0,2]$ is the Prüfer number corresponding to the tree in Fig. 2.

Conversely, it is also possible to construct a unique spanning tree corresponding to a Prüfer number P, by using the following algorithm.

2.3.2. Decoding algorithm

Algorithm 2

(1) Let P be a given Prüfer number with $n-2$ digits, and let R be the child of P. The tree T is initialized to an empty set, $T = \emptyset$.
(2) Let i be the left-most digit in P, and let k be the smallest element of R. Add the edge (i,k) to the tree T. Then remove the left-most digit from P, and also remove k from R. If i does not occur anymore in what remains in P, put it into the set R.
(3) Repeat step (2) until no digits remain in P.
(4) Add the last edge, with the two remaining nodes in the set R, to the tree T.

To illustrate this algorithm, let us consider Prüfer number of the previous example, $P = [1,6,0,0,2]$. Then numbers in the set $\{0,1,2,3,4,5,6\}$ which are not digits part of P consist the following set $R = \{3,4,5\}$. Let T be an empty set, $T = \emptyset$, then we add the edge $(1,3)$ to T and remove 3 from R and 1 from P. The digit 1 is no longer in P, then 1 is added to the set R. We have $P = [6,0,0,2]$ and $R = \{1,4,5\}$. Vertex 1 now is the smallest labeled in R. Thus we add $(6,1)$ to the tree T. Next, we remove 6 from P and 1 from R. The digit 6 is no longer in P, then it is added to the set R. We get $P = [0,0,2]$ and $R = \{6,4,5\}$. Vertex 4 now is the smallest labeled in R, then we add $(0,4)$ to the tree T. Next, we remove 0 from P and 4 from R, the digit 0 is still in P. Hence we

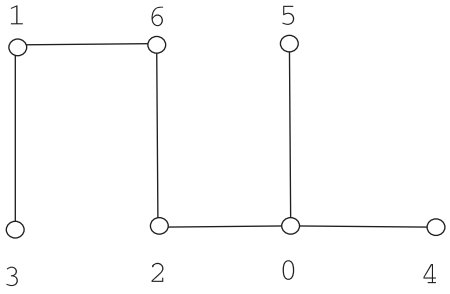

Fig. 2. Graph of T.

obtain $P = [0,2]$ and $R = \{5,6\}$. We repeat the process until no digits remain in P. Then we add the last edge with the two remaining nodes in the set R.

As it is noted in [14,22,24], this decoding algorithm can be carried out in $O(n\log n)$ with the aid of a heap. In order to decrease the time complexity to be of $O(n)$, we introduce the following decoding algorithm.

Algorithm 3

(1) Let P be a given Prüfer number with $n-2$ digits, and let R be the child of P. Elements in R are put in decreasing order from left to right. The tree T is initialized to an empty set, $T = \emptyset$.
(2) Let i be the left-most digit in P, and let k be the left-most element in R. Add the edge (i,k) to the tree T, and then remove the left-most digit from P. If i does not occur anymore in what remains in P, then replace in R, k by i; otherwise delete k from R.
(3) Repeat step (2) until no digits remain in P.
(4) Add the last edge, with the two remaining nodes in the set R, to the tree T.

Remark 1

(i) The main difference between Algorithms 2 and 3 is that, at each iteration of Algorithm 2 we seek the smallest element in R, but in Algorithm 3 we take the left-most digit in R.
(ii) A tree T encoded by Algorithm 1 gives a Prüfer number P_T. If P_T is decoded by Algorithm 2 it will produce the same tree T, and conversely.
(iii) A tree T encoded by Algorithm 1 gives a Prüfer number P_T. If P_T is decoded by Algorithm 3 it may not give the same tree T. However, in genetic algorithm we only need to decode a representation (Prüfer number) to a tree. Then we may use Algorithm 3 in the genetic algorithm, since it has a better time complexity than Algorithm 2.

Theorem 2.1. *Let P be a Prüfer number. The second decoding algorithm, Algorithm 3, constructs one and only one spanning tree corresponding to P. Moreover, for two different Prüfer numbers the algorithm produces two different spanning trees. This algorithm requires a time complexity of $O(n)$.*

Proof. It is easy to see that in performing Algorithm 3 we construct a unique tree with $n-1$ different edges, which spans the n vertices. Consequently T is a spanning tree.

Now, we show that for two different Prüfer numbers the algorithm constructs two different spanning trees. Let $P_1 = [p_0^1, p_1^1, \ldots, p_{n-3}^1]$ and $P_2 = [p_0^2, p_1^2, \ldots, p_{n-3}^2]$ be two different Prüfer numbers, and let R_1 and R_2 be their corresponding children, respectively. The corresponding trees to P_1 and P_2 are denoted by T_1 and T_2, respectively. We distinct the two following cases:

First case. Assume that R_1 and R_2 are different. Then there exists an element, vertex, r in $(R_1 \cup R_2)\backslash(R_1 \cap R_2)$. Assume that $r \in R_1\backslash R_2$, hence the vertex r is of degree 1 in T_1 and it is of degree at least 2 in T_2. Consequently, trees T_1 and T_2 are different. The same argument can be applied if $r \in R_2\backslash R_1$.

Second case. Assume that R_1 and R_2 are the same. If p_0^1 and p_0^2 are different digits, let r be the left-most element of the original child $R_1 = R_2$. Then the antecedent vertex to the leaf vertex r, in the tree T_1 (resp. T_2), is the vertex p_0^1 (resp. p_0^2). Hence trees T_1 and T_2 are different. We now assume that p_0^1 and p_0^2 are equal. Let i be the smallest index such that $p_i^1 \neq p_i^2$. Then at the $(i-1)$th iteration of Algorithm 3, trees T_1 and T_2 have the same edges. In the ith iteration of Algorithm 3, we add to the tree T_1 the edge (p_i^1, r), and to the tree T_2 the edge (p_i^2, r). The element r is either equal to $p_{i-1}^1 = p_{i-1}^2$, if p_{i-1}^1 is no longer in the remains digits of P_1, or r is a vertex of degree 1. In both cases the edge (p_i^1, r) is an element of T_1 and not an edge of T_2, also the edge (p_i^2, r) is an element of T_2 and not an element of T_1, which means that trees T_1 and T_2 are not the same.

Now, we show that Algorithm 3 can be performed in time of $O(n)$. Indeed, let Q be an n vector such that $Q(i)$ stores the degree of the vertex labeled i. We note that if a vertex is of degree 1 it belongs to R, otherwise it belongs to P. The pseudo code for step (1) of Algorithm 3 is outlined as follows: At first, each component of the vector Q is initialized to 1, then

```
for i = 1 to n − 2 do
begin
   p := P(i);
   Q(p) := Q(p) + 1;
end
for i = 1 to n do
begin
   if (Q(i) = 1) then
   begin
      R(r) := i; (r is initialized to 0)
      r := r + 1;
   end
end
```

Now it is clear that step (1) of Algorithm 3 is of O(n). Steps (2) and (3) are also of O(n). Hence Algorithm 3 is of O(n). The proof is complete. □

3. Genetic algorithm search for MST

The choice of an encoding solution is one of the most important step in the application of genetic algorithm to a problem. Prüfer numbers consist one of the most efficient encoding method for spanning trees in genetic algorithm search, since they cover the hall space of spanning trees, represent only spanning trees and unbiased, each tree is represented once. However, this representation method has little locality and does not fit with the application of concept of schemata.

One of the disadvantages in using this representation method, is that while performing a crossover operation, even a small change in a parent P may result an offspring with a tree not in the neighborhood of the parent tree. As an example, consider in a graph with seven nodes the two following Prüfer numbers with a single change, $P_1 = [5,0,6,3,1]$ and $P_2 = [5,0,6,3,0]$. The resulting trees by the decoding Algorithm 3 (resp. Algorithm 2) have only two edges in common.

In minimization problem, the goal of genetic algorithm is for each successive generation to have chance of containing more lower fit chromosomes. Genes in a chromosome which appear rarely are likely eliminated, while one which appear very often are committed.

The theoretical foundation of genetic algorithms by Holland [16] rely on binary string representations and on the notion of schemata. In a binary representations with m bits, a schema of order l is a vector with l fixed components and the remaining $m - l$ components take any values in $\{0,1\}$. These free components are represented by an asterisk ($*$). The two following 6 bits strings:

1 1 0 0 1 0 0 1 1 1 0 0

are examples of the schema

$* \; 1 \; * \; * \; * \; 0$

Consider a genetic algorithm where the coding is binary string and the fitness function is defined. In this paper the fitness is defined as the objective function value. Suppose that strings with some particular schema have, on average, a lower fitness than strings without this schema. Since this schema tends to confer above average fitness, it will tend to appear in increasing numbers of strings from generation to generation. Conversely, schemata that tend to confer below average fitness tend to die out from generation to generation. This effect is a direct result of the selection procedure of string parents and is presented in mathematical form as a concept of schema theorem. The schema theorem suggests that short, low-order, above-average fitness schemata receive exponentially increasing trials in subsequent generations of a genetic algorithm. The schema theorem (or the fundamental theorem) of genetic algorithms states that the expected number of copies of a schema H in the next generation is

$$M(H,t+1) \geqslant M(H,t)\frac{f(H)}{F}\underbrace{\left(1-p_c\frac{\delta(H)}{l-1}-o(H)p_m\right)}_{a},$$

where $M(H,t)$ is the number of strings in population t with the schema H, $f(H)$ is the average fitness of the strings with the schema H, F denotes the average fitness of the entire population, p_c (resp. p_m) is the probability of the schema being destroyed by crossover (resp. mutation), $\delta(H)$ is the distance between the first and last specific string position, l is the length of the string and $o(H)$ is the number of fixed positions present in the template. The term a in the above estimation denotes the probability that the schema will survive mutation and crossover, see [13].

The other disadvantage in the application of genetic algorithms to solve NMST problems is that the concept of schemata does not take place while using Prüfer representations. Let $P=[p_0,p_1,\ldots,p_{n-3}]$ be a Prüfer number. A gene in a position i, p_i, may take any value in the set $\{0,1,2,\ldots,n-1\}$ and each value of p_i represents the node labeled p_i, which is a vertex of at least tow edges of the graph G. Then a gene provides information related to vertices and not to edges. In general it does not have a significant fitness weight to affect some chromosomes having in their components the gene p_i, since the fitness is a function of edges representations. Hence for a particular schema to have, on average, a lower fitness than strings without this schema, cannot be applied.

To test the performances of the application of GA to some MST problems, we describe some GA approaches using Prüfer numbers. In the paper [26], the authors Zhou et al. proposed a GA to solve quadratic minimum spanning tree (q-MST) problems. Their method uses; Prüfer numbers for encoding trees, uniform crossover, mutation and mixed strategy with $(\mu+\lambda)$ selection and roulette wheel selection. After comparing results of their GA to two heuristic algorithms, they concluded that GA using Prüfer numbers is an effective approach to q-MST problems. In this paper we will use this approach to solve q-MST problem and a problem arises in fuzzy random minimum spanning tree problem [20]. In order to vary the selection method in the same algorithm, we will also employ scaling and sharing selection methods. The computational results of GA approaches will be compared to a tabu search method proposed in the next section.

3.1. Crossover and mutation

The main operators in genetic algorithms are known as crossover and mutation. In a crossover operation two parents are combined to produce two new offsprings. While a mutation stands for a random modification of a chromosome. Many crossover-like and mutation-like are defined. In this paper we will use the uniform crossover. In the uniform crossover operation individual bits in the strings of two parents are swapped with a fixed probability, typically 0.5. For each crossover operation we generate a random binary string with the same size as of chromosome, with respect to the probability p_c of a string being 1. Then genes in two parents which their positions in the string mask take the digit 1 are swapped. In the mutation operation each gene can be selected with a probability p_m, to be replaced with a random digit in the set of all possible digits. Figs. 3 and 4 illustrate these two operations [26].

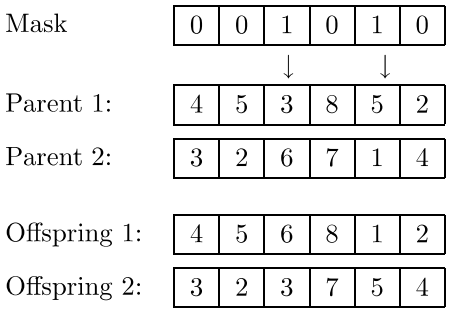

Fig. 3. Crossover operation.

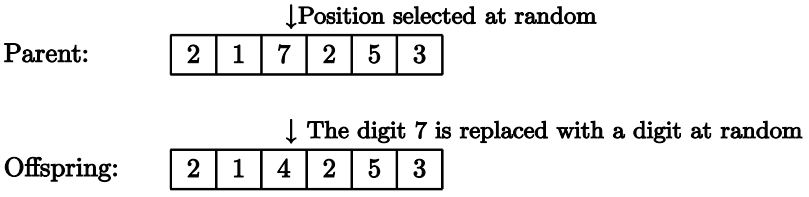

Fig. 4. Mutation operation.

3.2. Selection methods

To examine the performance of the genetic algorithm in question we will use three different selection methods: sharing method, scaling method and a mixed strategy with $(\mu + \lambda)$-selection method and roulette wheel selection.

3.2.1. Sharing selection method

A sharing function is defined to determine the neighborhood and degree of sharing for each string in the population. The most widely used sharing function is given by:

$$sh(d_{ij}) = \begin{cases} 1 - (d_{ij}/\sigma_s)^{\alpha}, & \text{if } d_{ij} < \sigma_s, \\ 0, & \text{otherwise,} \end{cases}$$

where d_{ij} denotes a distance between chromosomes i and j, σ_s denotes the threshold of dissimilarity between tow elements and α is a constant parameter which regulates the shape of the sharing function. Let P_i and P_j be two chromosomes (Prüfer numbers) in a population. We define the distance d_{ij} as the number of different components of P_i and P_j:

$$d_{ij} = 0 + \sum_{\substack{k=0, \\ P_i[k] \neq P_j[k]}}^{n-3} 1.$$

The sharing function sh measures the similarity level between two strings in the population. Strings close to an individual require a high degree of sharing (close to one), and strings far from the individual require a very small degree of sharing (close to zero). The function sh returns one if the elements are identical, zero if their distance is higher than a threshold of dissimilarity σ_s, and an intermediate value otherwise. The shared fitness function f_s, in a population of size N, is calculated by taking the fitness function and dividing through by the accumulated number of shares:

$$(f_s)_i = \frac{f_i}{\sum_{j=1}^{N} sh(d_{ij})}.$$

Thus, when many individuals are in the same neighborhood, namely close to each other with respect to the distance d, they contribute to one another share fitness value.

3.2.2. Scaling selection method

Without scaling the fitness function f (objective function) there is a tendency for few extraordinary individuals to dominate the selection process. In the normal selection rule the probability that an individual will be selected is $p_i = f_i / \sum f_j$, then extraordinary individuals would take over a significant proportion of the population in a single generation which is not desirable. In this situation the average members and best members get nearly the same number of copies in future generations, and competition among population members becomes less strong. There are several scaling method such as linear scaling, sigma scaling and power law scaling (see [13], for instance). Linear scaling is one of the useful scaling method. We calculate the scaled fitness f' from the fitness f using a linear equation of the form:

$$f' = af + b,$$

where a and b are chosen so that in a population the average scaled fitness f'_{avg} remains the same as the average raw fitness f_{avg}, and the maximum scaled fitness $f'_{\max}$ is equal to a multiple of the average raw fitness $f_{\max}$ by a real constant c, usually c is set between 1.2 and 2. It is possible that the low scaled fitness values become negative. In order that scaled fitness of each individual to be nonnegative the following algorithm is proposed, see [13,25]:

Step 1: Calculate the mean fitness f_{mean}, the maximal fitness $f_{\max}$, and the minimal fitness $f_{\min}$ of the population.
Step 2: If $f_{\min} > \frac{cf_{\text{mean}} - f_{\max}}{c-1}$, then go to step 3. Otherwise go to step 4.
Step 3: Set $a = \frac{(c-1)f_{\text{mean}}}{f_{\max} - f_{\text{mean}}}$, $b = \frac{f_{\text{mean}}(f_{\max} - cf_{\text{mean}})}{f_{\max} - f_{\text{mean}}}$, and go to step 5.
Step 4: Set $a = \frac{f_{\text{mean}}}{f_{\text{mean}} - f_{\min}}$, $b = \frac{f_{\min} f_{\text{mean}}}{f_{\text{mean}} - f_{\min}}$, and go to step 5.
Step 5: Calculate $f'_i = af_i + b$ for $i = 1, 2, \ldots, N$.

3.2.3. Mixed strategy selection method

The mixed strategy with $(\mu + \lambda)$ selection and roulette wheel selection proposed in [26], selects μ best chromosomes from μ parents and λ offsprings. If there are no μ different chromosomes available, then the vacant pool of population are filled up with roulette wheel selection.

4. Tabu search method

Tabu search (TS) method is a neighborhood metaheuristic method. The basic concept of TS is to explore the space of solutions by moving at each iteration from a solution $\boldsymbol{x}$ to the best solution in a subset of its neighborhood $\mathcal{N}(\boldsymbol{x})$. The modern form of tabu search method derives from Glover [9–11]. Many of the applications in the literature involve integer programming problems, scheduling, vehicle routing, traveling salesman and other combinatorial problems (see [6,7,12,15], for instance). Recently Cunha et al. [6] applied a tabu search algorithm to water network optimization. Blum et al. [4] investigated some metaheuristic approaches for edge-weighted k-cardinality tree problems and compared the performances of genetic algorithms, simulated annealing, ant colony optimization and TS. They demonstrated that TS has advantages for high cardinality. Since a MST problem is a special type of edge-weighted k-cardinality tree problems and corresponds to the highest cardinality case, we construct a solution method through a TS algorithm.

Let G be an undirected graph with n vertices and m edges, and let $E = \{e_1, e_2, \ldots, e_m\}$ be the set of edges in the graph G. Let T be a spanning tree of G and let $\boldsymbol{x}$ be its characteristic vector defined by

$$x_i = \begin{cases} 1 & \text{if } e_i \text{ is an edge of } T, \\ 0 & \text{otherwise.} \end{cases}$$

The objective functions to minimize are in term of $\boldsymbol{x}$, $z(\boldsymbol{x})$:

$$\left.\begin{array}{ll} \text{minimize} & z(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{x} \in X \end{array}\right\}, \tag{1}$$

where X denotes the family of characteristic vectors representing all possible spanning trees of the graph G. In this section we will construct a solution algorithm for some nonlinear minimum spanning tree MST problems. In particular, we consider q-MST problem and MST problems with objective functions depending on linear and quadratic terms. This algorithm is based on TS method incorporating strategic oscillation and intensification by elite solutions.

The algorithm starts from an initial spanning tree solution constructed by an adequate algorithm. Then the improvement strategy, which consists of exchanging a pair of edges, generates the neighborhood of the current solution. In TS method a pair of exchanged edges is called move. Let T be the current solution tree and e_r be an edge of T selected randomly. After removing the edge e_r, the tree T is cut into two sub-trees T_1 and T_2. Then the added edge consists of an edge with extremities selected randomly one from the vertices of T_1 and the other form the vertices of T_2. Hence the resulting solution is a spanning tree. In order to prevent cycling between the same solutions, certain exchanges can be forbidden by earning them the status of ''tabu move'', and the set of tabu moves defines the tabu list. Tabu moves are not permanent; a short-term memory

function enables them to leave the tabu list. The use of an aspiration criterion permits certain moves on the tabu list to overcome any tabu status. In the proposed algorithm, we use strategic oscillation to intensively explore the region around the current neighborhood. Strategic oscillation was originally introduced to provide an effective interplay between intensification and diversification over the intermediate to long term memory. In addition to a short-term memory, we use a frequency-based memory as a long-term memory. On the other hand, the intensification undertakes to create solutions aggressively encouraging the incorporating of solutions from an elite solution set. The process goes on until the termination criterion is satisfied.

Let $\boldsymbol{x}^c$ be the current solution of problem (1) and T^c be its corresponding spanning tree, and let $\boldsymbol{x}^b$ and T^b be the best solution found so far, and its corresponding spanning tree, respectively. We denote by $z(\boldsymbol{x})$ the objective function value of the problem at an element $\boldsymbol{x}$. We define the following parameters:

NIO: number of iterations in the oscillation procedure.
MAX_NIO: threshold of the counter *NIO*.
k: counts the iterations where the best solution is unrenewed during the improvement strategy process.
MAX_k: threshold of the counter k.
UNRIter: counts the iterations where the best solution is unrenewed.
MAX_Iter: threshold of the counter *UNRIter*.

The proposed algorithm is described in the following steps, which are followed by a motivation and the description of each feature implemented in this algorithm.

Step 0 (**Initial solution**). Set $NIO = UNRIter = k = 0$. Generate an initial solution $\boldsymbol{x}^0$ corresponding to an initial spanning tree T^0. Set $\boldsymbol{x}^c := \boldsymbol{x}^0$ and $\boldsymbol{x}^b := \boldsymbol{x}^0$.

Step 1 (**Improvement**). If $k > MAX\_k$, then go to step 3. Otherwise, improve the obtained solution by the *improvement strategy*.

Step 2. If $z(\boldsymbol{x}^c) < z(\boldsymbol{x}^b)$, then set $k = 0$ and $\boldsymbol{x}^b := \boldsymbol{x}^c$, and return to step 1; otherwise, set $k := k + 1$ and return to step 1.

Step 3 (**Strategic oscillation**). If $NIO > MAX\_NIO$, then go to step 5; otherwise, add a_1 edges among $E(G)\backslash E(T^c)$ by using the *edge addition rule* and continue to remove one of the edges in a cycle by using the *edge remove rule* until a spanning tree is formed. Where $E(G)$ is the edge set of G, and $E(T^c)$ is the edge set of edges of T^c.

Step 4. If $z(\boldsymbol{x}^c) < z(\boldsymbol{x}^b)$, then set $NIO = k = 0$, $\boldsymbol{x}^b := \boldsymbol{x}^c$, and return to step 1; otherwise, set $NIO := NIO + 1$ and return to step 3.

Step 5 (**Intensification by elite solutions**). Select a set of edges (*SE*) that are in most of the last M elite solutions. Then, starting from an edge chosen uniformly at random from T^c, construct a new solution by adding edges from the set of edges $T^c \cup SE$, using edge addition rule, until a spanning tree is formed. Set $UNRIter := UNRIter + 1$. If $UNRIter > Max\_Iter$, then go to step 7. Otherwise go the next step.

Step 6. If $z(\boldsymbol{x}^b) < z(\boldsymbol{x}^c)$, then set $\boldsymbol{x}^b := \boldsymbol{x}^c$, $UNRIter = k = 0$ and return to step 1; otherwise, set $k = 0$ and go to step 1.

Step 7. (**Local minimum**). Explore exhaustively the neighborhood of each spanning tree of the last M elite solutions by the procedure *Local minimum*. Then terminate.

The essential features that have been considered in building this TS algorithm for solving a minimum spanning tree problem are: generating an initial solution, the neighborhood structure, the improvement strategies, short-term and long-term memories, diversification procedure and oscillation strategy, intensification by elite solutions, termination criterion.

4.1. Motivation

Let G be a complete undirected graph with n vertices and m edges, and let $E = \{e_1, e_2, \ldots, e_m\}$ be the set of edges of the graph G. The family of all spanning trees of G is denoted by $\mathscr{T}$. We consider the following quadratic minimum spanning tree problem:

$$\text{minimize} \quad z(x) = x^t V x, \quad \text{s.t. } x \in X,$$

where $V = (v_{ij})$ is an $m \times m$ matrix and X stands for the set of characteristic vectors representing all spanning trees of the given graph. Then we have

$$x^t V x = \sum_{i=1}^{m} \sum_{j=1}^{m} v_{ij} x_i x_j. \tag{2}$$

Let fix $\boldsymbol{x}$, and denote the components of $\boldsymbol{x}$ which take the value 1 by $x_{i_1}, x_{i_2}, \ldots, x_{i_{n-1}}$. Then Eq. (1) becomes

$$x^t V x = \sum_{k=1}^{n-1} \sum_{l=1}^{n-1} v_{i_k i_l}. \tag{3}$$

Now, in order to improve the current solution $\boldsymbol{x}$ to a new solution $\boldsymbol{x}_{\text{new}}$, we want to perform an exchange of two edges. We remove an edge e_{i_r}, $i_r \in \{i_1, i_2, \ldots, i_{n-1}\}$, from T and replace it by the best edge, e', of the graph G such that the resulting solution remains a representation for a spanning tree. For the seek of simplicity, we assume that $r = 1$. Then Eq. (2) for both solutions $\boldsymbol{x}$ and $\boldsymbol{x}_{\text{new}}$ can be expressed as

$$x^t V x = \sum_{l=1}^{n-1} v_{i_1 i_l} + \sum_{k=1}^{n-1} v_{i_k i_1} + \sum_{k=2}^{n-1} \sum_{l=2}^{n-1} v_{i_k i_l}, \tag{4}$$

$$x_{\text{new}}^t V x_{\text{new}} = \sum_{l=1}^{n-1} v_{t i_l} + \sum_{k=1}^{n-1} v_{i_k t} + \sum_{k=2}^{n-1} \sum_{l=2}^{n-1} v_{i_k i_l}, \tag{5}$$

where t denotes the index, in the graph G, of the added edge $e' = e_t$. Form Eqs. (3) and (4) we deduce that the edge e' is a solution of the following equation:

$$e' = e_t; \quad t = \underset{1 \leqslant j \leqslant m, j \neq i_1}{\operatorname{argmin}} \left\{ \sum_{l=1}^{n-1} v_{j i_l} + \sum_{k=1}^{n-1} v_{i_k j} \,\middle|\, T - e + e' \in \mathscr{T} \right\}, \tag{6}$$

where $\mathscr{T}$ stands for the family of spanning trees of G. Let SE be a set of edges of G, and $\boldsymbol{y}$ be its characteristic vector i.e., $y_i = 1$ if $e_i \in SE$ and $y_i = 0$ otherwise, for $i = 1, \ldots, m$. We denote by $z(SE)$ the value of the function $z(\cdot)$ at the element y, $z(y)$. Then, Eq. (5) is equivalent to find an edge e' in the graph G such that

$$e' = e_t; \quad t = \underset{1 \leqslant k \leqslant m, k \neq i_1}{\operatorname{argmin}} \{ z(\{e_k, e_{i_2}, e_{i_3}, \ldots, e_{i_{n-1}}\}) - z(\{e_{i_2}, e_{i_3}, \ldots, e_{i_{n-1}}\}) | T - e + e\prime \in \mathscr{T} \}.$$

Based on the above arguments, we constructed the following procedures: initial solution, adding edge rule and remove edge rule; which consist of the important features in the proposed algorithm for solving nonlinear MST problems involving quadratic and linear terms, in particular q-MST problem.

4.2. Initial solution

Let $SCC(i)$ denote a Set of Connected Component that consists of i edges. To construct a spanning tree T, first, an edge $e \in E(G)$ is chosen uniformly at random. With this edge, a subtree $SCC(1)$ which consists of only one edge is created. Then, a set of connected component $SCC(k+1)$ is constructed by adding an edge $e \leftarrow \operatorname{argmin}\{z(SCC(k) + e') - z(SCC(k)) | e' \in E_{NC}(SCC(k))\}$ to the current $SCC(k)$ under construction, where $E_{NC}(SCC(k))$ is defined as follows:

$$E_{NC}(SCC(k)) = \{e \in E(G) | SCC(k) + e \text{ has no cycle}\}.$$

4.3. Neighborhood structure

Let T be a set of edges which forms a spanning tree, and let $\mathscr{T}$ be the family of all possible spanning trees in the given graph. A neighborhood $N(T)$ is defined as a set of all spanning trees which can be generated by

removing an edge $e \in T$ and by adding an edge e' from the set $E_{NH}(T-e)\backslash\{e\}$, where $E_{NH}(T-e)$ is defined as follows:

$$E_{NH}(T-e) = \{e' \in E(G) | T - e + e' \in \mathscr{T}\}.$$

4.4. Improvement strategy

Let T^c be the current spanning tree. Randomly select a subset of moves from the neighborhood $N(T^c)$. Then choose a move with the lowest objective function and nontabu status, to form a new solution. The tabu status can be overridden if an aspiration criterion is satisfied.

4.5. Short-term memory

TS uses a short-term memory to escape from local minima and to avoid cycling. The short-term memory is implemented as a set of tabu lists that store solution attributes. Attributes usually refer to components of solutions, moves, or differences between two solutions. The use of tabu lists prevents the algorithm from returning to recently visited solutions.

Our TS approach, to tackle the MST problem, uses only one tabu list denoted by *TabuList*. The attribute it stores is the index of the edges that were recently added or removed. Every move involves removing one edge e from the current spanning tree T^c, and adding a different edge to $T^c - e$. The status of the forbidden moves are explained as: If an edge e_j is in *TabuList* and $x_j = 0$, then adding the edge e_j is forbidden. In addition, if an edge e_i is in *TabuList* and $x_i = 1$, then removing the edge e_i is forbidden.

4.6. Aspiration criterion

An aspiration criterion is activated to overcome the tabu status of a move whenever the solution then produced is better than the best historical solution achieved. This criterion will be effective only after a local optimum is reached.

4.7. Strategic oscillation procedure

The strategic oscillation approaches to a boundary, which is represented by the set of spanning trees, by adding or removing edges. In this paper, we use one type of strategic oscillation approach for the problem, which recedes the boundary by continuing to add edges to a spanning tree and then approaches to the boundary by continuing to remove edges until a spanning tree is formed. Adding edges proceeds for a specified depth beyond the boundary, and turns around. At this point the boundary is again approached and is reached by removing edges.

Edge addition rule. Again, we denote by $SCC(i)$ a set of connected component that consists of i edges, defined in the previous section. Then the $SCC(k+1)$ is constructed by adding an edge $e \leftarrow \arg\min\{z(SCC(k)+e') - z(SCC(k)) | e' \notin SCC(k)\}$ to the current set $SCC(k)$ under construction.

Edge remove rule. Let $SCC(k)$ be a connected set of k edges, $k \geqslant n$. Then we construct a spanning tree from the set $SCC(k)$ by performing the procedure used to construct the initial solution, where we use the set of edges $SCC(k)$ instead of $E(G)$.

4.8. Long-term memory

The roles of intensification and diversification in TS are especially relevant in longer term search processes. Frequency-based memory is one of the long-term memories and consists of gathering pertinent information about the search process so far. In our algorithm, we use residence frequency memory, which keeps track of the number of iterations where no improvement has been done and keeps in memory a number of elite solutions. By using the residence frequency memory, we provide the following diversification and intensification processes.

1. *Diversification procedure*. The diversification derives the search into a new region. It begins at the situation that some spanning tree is formed. In order to explore a new search region a number of edges are removed and replaced by other edges. This procedure is illustrated in the strategic oscillation steps of the algorithm. If the strategic oscillation procedure is iterated in *MAX_NIO* times, then the intensification procedure is started.
2. *Intensification procedure using*. The intensification procedure begins at the condition that no edge is selected after a long tune. It forces the current solution to be improved. We tray to construct a better solution from the last M best solutions by using the edge addition rule. If a better solution is not found then terminate the oscillation procedure, otherwise go to the diversification procedure.

4.9. Local minimum

This is a descent method to find a local minimum in the neighborhood of each spanning tree T of the last M elite solutions. It explores exhaustively the neighborhood set of each tree as follows: Let T be an elite spanning tree and $\{e_1, e_2, \ldots, e_{n-1}\}$ be its set of edges. Set $\boldsymbol{x}_T^{lm} := \boldsymbol{x}$, where $\boldsymbol{x}$ denote the characteristic vector of T and $\boldsymbol{x}_T^{lm}$ is a 0–1 m-vector. For $i = 1$ to $n - 1$ find an edge e_i' in the graph G such that $T' = T - e_i + e_i'$ forms a spanning tree and $z(\boldsymbol{x}_T^{lm}) < z(\boldsymbol{x}')$, where $\boldsymbol{x}'$ denotes the corresponding representation of the tree T'. At each iteration i, if e_i' exists then set $e_i := e'_i$ and $\boldsymbol{x}_T^{lm} := \boldsymbol{x}'$. Repeat this procedure until no improvement exists during one iteration of the procedure. If $z(\boldsymbol{x}_T^{lm}) < z(\boldsymbol{x}^b)$, then set $\boldsymbol{x}_T^{lm} := \boldsymbol{x}^b$, where $\boldsymbol{x}^b$ is the best solution found so far by the main algorithm. Apply this procedure to all M elite spanning tree solutions.

4.10. Termination criterion

The counter *UNRIter* counts the iterations where the best solution T^b is unrenewed. If *UNRIter* is greater than the threshold *Max_Iter*, the proposed algorithm goes to the last step and then terminates. The quality of the final solution and the computer running time are both influenced by the termination criterion.

5. Computational experiments

Let G be a complete undirected graph with n vertices and m edges, and let X denote the set of all possible spanning trees of the graph G represented by 0–1 m-vectors as it is described in the previous section. In this section we report some numerical results to test the performances of GAs and TS method described in this paper. We consider the two following spanning tree problems:

$$\text{Problem 1: minimize} \quad z(x) = x^t V x, \quad \text{s.t. } x \in X,$$

$$\text{Problem 2: maximize} \quad z(x) = \frac{ax + g}{\sqrt{x^t V x}} \quad \text{s.t. } x \in X,$$

where V is a real $m \times m$ matrix, a is a real m-vector, and g is a real value. Since $\max z(x) = -\min(-z(x))$, then a maximum problem can be considered as a minimum problem. Problem 1 has been studied by the authors Zhou et al. [26] using a GA approach which uses Prüfer numbers for its chromosome representation method. Problem 2 stands for a solution method arises from a fuzzy random minimum spanning tree problem formulated by Mermri et al. [20].

The experiments were conducted on graphs G with different number of vertices, n. Data in the above problems are generated randomly. The TS parameters are set as follows: maximum number of iterations in the improvement strategy $MAX\_k = 500$, maximum number of iterations in the oscillation strategy $MAX\_NIO = 5$, $MAX\_UNRIter = 10$, number of elite solutions $M = 10$ and number of add remove edge $a_1 = 3$. In the GA approaches, the parameters are set as follows: crossover probability $p_c = 0.4$, mutation probability $p_m = 0.01$, population size 120, and maximum number of generations 1000. In the sharing selection method, the threshold of dissimilarity between tow parents is taken as $\sigma_s = n/2$, and $\alpha = 1$.

Iteration parameters in GA and TS algorithms are set in such a way their run times will be close to each other, and therefore a comparison of their performances make sense. The algorithms were coded in C++ programming language and implemented on a computer with a CPU Celeron 1.7 GHz and RAM 252 MB.

Table 1
Best objective values (minimum)

Nodes	TS	GA-($\mu + \lambda$)	GA-scaling	GA-sharing
5	665.1	665.1	665.1	665.1
10	3157.5	3157.5	3157.5	3331.0
15	7569.3	8469.7	8728.3	8754.3
20	14768.1	17296.9	17371.8	16967.0
30	34915.7	42020.0	41763.2	41899.0
50	92943	112554	112052	112665

Tables 1–6 illustrate the computational results of Problems 1 and 2, by using TS method and GAs described in this paper. The expressions GA-scaling, GA-sharing and GA-($\mu + \lambda$) denote, respectively, GA using scaling selection method, GA using sharing selection method and GA using mixed strategy with ($\mu + \lambda$) selection and roulette wheel selection proposed by Zhou et al. [26] to solve q-MST problems.

Problem 1. In these computational tests, we remark that GAs are quite uncompetitive with TS method for solving edge-weighted q-MST problem. This justifies what has been discussed in Section 3, that GAs using Prüfer numbers for their encoding methods do not evolve effectively according to what is expected by the

Table 2
Average objective values

Nodes	TS	GA-($\mu + \lambda$)	GA-scaling	GA-sharing
5	665.1	665.1	665.1	665.1
10	3193.6	3280.4	3374.5	3386
15	7569.3	8909.5	8996.6	8943.9
20	14253.7	17463.5	17569.8	17506.3
30	35206.2	42319.6	42152.7	42158.4
50	93369.0	112749.8	112655.1	112636.6

Table 3
Average time in second

Nodes	TS	GA-($\mu + \lambda$)	GA-scaling	GA-sharing
5	0.02	0.3	0.01	0.15
10	1.6	1.3	0.9	1.7
15	4	4.9	4	5.5
20	9.9	20	14	16.2
30	95	90	91.2	95
50	1155	1171	1052	1060

The sign "–" means a very small time.

Table 4
Best objective values (maximum)

Nodes	TS	GA-($\mu + \lambda$)	GA-scaling	GA-sharing
6	0.729	0.729	0.729	0.729
7	1.141	1.141	1.141	1.141
8	2.062	1.870	2.062	2.062
9	1.543	1.372	1.412	1.432
10	1.485	1.275	1.399	1.346
15	2.367	1.719	1.600	1.576
30	4.958	2.042	2.046	2.200
50	6.865	1.925	1.702	1.739

Table 5
Average objective values

Nodes	TS	GA-($\mu + \lambda$)	GA-scaling	GA-sharing
6	0.729	0.729	0.729	0.729
7	1.141	1.141	1.141	1.141
8	2.062	1.774	2.000	1.931
9	1.543	1.238	1.322	1.302
10	1.485	1.228	1.320	1.277
15	2.334	1.448	1.505	1.488
30	4.918	1.870	1.946	1.948
50	6.771	1.670	1.650	1.668

Table 6
Average time in second

Nodes	TS	GA-($\mu + \lambda$)	GA-scaling	GA-sharing
6	0.6	0.3	1.2	
7	0.9	0.5	1.5	
8	1.1	0.7	1.8	
9	1.5	1	2.2	
10	2	1.3	2.7	
15	7.2	5	7	
30	120	85	92	
50	1374	1095	1098	

The sign "–" means a very small time.

genetic evolution concept. The best and average values obtained by TS are very close or the same for number of nodes less than or equal to 15. One conclude that the proposed TS method has good performances.

Problem 2. Computational results on Problem 2 show that the proposed TS method has better performances than GAs using Prüfer numbers for their encoding method.

6. Conclusion

In this paper we investigated the application of GAs for solving nonlinear MST problems and proposed a TS algorithm for solving some kinds of these problems. We presented the most known tree representations and basic concepts of GA. Then, we discussed why of GAs using Prüfer numbers for their chromosome representations, cannot work effectively for solving nonlinear MST problems. Next, we constructed an algorithm base on TS method for solving some nonlinear MST problems. These problems include q-MST and MST problems with objective function depending on linear and quadratic terms in their objective functions. The construction of this TS algorithm was motivated on a q-MST problem. Computational results showed that GAs are quite uncompetitive with TS for solving this kind of MST problems.

References

[1] F-N. Abulai, R-L. Wainwright, D-A. Schoenefeld, Deteninant facorization: a new encoding scheme for spanning tree applied to the probabilistic minimum spanning tree problem, in: Proceeding of the Sixth International Conference on Genetic Algorithms, Morgan Kaufman, San Francisco, 1995, pp. 470–477.
[2] R. Ahuja, T. Magnati, J. Orlin, Network Flows: Theory, Algorithm and Applications, Prentice Hall, New Jersey, 1993.
[3] C. Bazlamaçci, K. Hindi, Minimum-weight spanning tree algorithms: a survey an empirical study, Computer & Operations Research 28 (2001) 767–785.
[4] C. Blum, M.-J. Blesa, New metaheuristic approaches for the edge-weighed *k*-cardinality tree problem, Computer & Operations Research 32 (2005) 1355–1377.

[5] A. Cayley, A theorem on trees, Quarterly Journal of Mathematics 23 (1889) 376–378.
[6] M.C. Cunha, L. Ribeiro, Tabu search algorithms for water network optimization, European Journal of Operational Research 157 (2004) 746–758.
[7] J.-L. François, C. Martin-del-Campo, R. François, L.-B. Morals, A practical optimization procedure for radial fuel lattice design using tabu search with a multiobjective function, Annals of Nuclear Energy 30 (2003) 1213–1229.
[8] M. Gen, R. Cheng, S.-S. Oren, Network design techniques using adapted genetic algorithms, Advances in Engineering Software 32 (2001) 731–744.
[9] F. Glover, Future paths for integer programming and links to artificial intelligence, Computers & Operations Research 13 (5) (1986) 533–549.
[10] F. Glover, Tabu Search-Part I, ORSA Journal on Computing 1 (1989) 190–206.
[11] F. Glover, Tabu Search-Part II, ORSA Journal on Computing 2 (1990) 4–32.
[12] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, Dordrecht, 1997.
[13] D-E. Goldberg, Genetic Algorithms in search, Optimization, and Machine learning, Addison-Wesley, 1989.
[14] J. Gottlieb, B-A. Julstrom, G-R Raidl, F. Rothlauf, Prüfer numbers: a poor representation os spanning trees for evolutionary search, in: Proceedings of the Genetic and Evolutionary Compution Conference (GECCO 2001), Morgan Kaufman Publishers, 2001, pp. 343–350.
[15] S. Hanafi, A. Freville, An efficient tabu search approach for the 0–1 multidimensional knapsack problem, European Journal of Operational Research 106 (1998) 659–675.
[16] J.-H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, 1975.
[17] J. Jarvis, D. Whited, Computational experience with minimum spanning tree algorithms, Operations Research Letters 2 (1) (1983) 36–41.
[18] M. Krishnamoorthy, A. Ernst, Y.M. Sharaiha, Comparison of algorithms for degree constrained minimum spanning tree, Journal of Heuristics 7 (2001) 587–611.
[19] J.-B. Kruskal, On the shortest spanning subtree of a graph and traveling salesman problem, Proceedings of ACM 7 (1) (1956) 48–50.
[20] E.-B. Mermri, K. Katagiri, M. Sakawa, K. Kato, A tabu search algorithm for fuzzy random minimum spanning tree problems through the probability maximization model using possibility and necessity measures, in: Proceedings of the 2nd International Conference on Soft Computing and Intelligent Systems, 2004 (vol. CD-ROM).
[21] R.-C. Prim, Shortest connection networks and some generations, Bell System Technical Journal 36 (1957) 1389–1401.
[22] C.-C. Palmer, A. Kershenbaum, Representing trees in geometric algorithm. IMB T.J. Watson Research Center.
[23] H. Prüfer, Neuer beweis eines satzes uber permutationen, Archiv fur Mathematik und Physik 27 (1918) 742–744.
[24] F. Rothlauf, D-E. Goldberg, Prüefer numbers and genetic algorithms: a lesson how the low locality of an encoding can harm the performance of GAs, in: Proceedings of PPSN VI, LNCS, vol. 1917, Springer-Verlag, Berlin, 2000, pp. 395–404.
[25] M. Sakawa, Genetic Algorithms and Fuzzy Multiobjective Optimization, Kluwer Academic Publishers, 2002.
[26] G. Zhou, M. Gen, An effective genetic algorithm approach to the quadratic minimum spanning tree problem, Computers & Operations Research 25 (3) (1998) 229–237.
[27] G. Zhou, M. Gen, Genetic algorithm approach on multi-criteria minimum spanning tree problem, European Journal of Operational Research 114 (1999) 141–152.