Local Search Algorithms for the k-Cardinality Tree Problem

Christian Blum, Matthias Ehrgott
Fachbereich Mathematik
Universität Kaiserslautern
Postfach 3049, 67653 Kaiserslautern
Germany
e-mail: ehrgott@mathematik.uni-kl.de
fax: (49) 631 29082

Abstract

In this paper we deal with an $\mathbb{NP}$-hard combinatorial optimization problem, the k-cardinality tree problem in node weighted graphs. This problem has several applications, which justify the need for efficient methods to obtain good solutions. We review existing literature on the problem. Then we prove that under the condition that the graph contains exactly one trough, the problem can be solved in polynomial time. For the general $\mathbb{NP}$-hard problem we implemented several local search methods to obtain heuristic solutions, which are qualitatively better than solutions found by constructive heuristics and which require significantly less time than needed to obtain optimal solutions. We used the well known concepts of genetic algorithms and tabu search with useful extensions. We show that all the methods find optimal solutions for the class of graphs containing exactly one trough. The general performance of our methods as compared to other heuristics is illustrated by numerical results.

1 Introduction and Mathematical Formulation

1.1 The Problem

The k-cardinality tree problem is a combinatorial optimization problem which generalizes the well known minimum weight spanning tree problem. It consists in finding in a node- or edge-weighted graph $G = (V, E)$ a subtree with exactly k edges, such that the sum of the weights is minimal. Due to various applications, e.g. in oil-field leasing [21], facility layout [15], open pit mining [28], quorum-cast routing [7] and telecommunications [18] it has gained considerable interest in recent years.

The problem can be formulated as follows. Let $G = (V, E)$ be a graph with a weight function $w : E \to I\!N$ on the edges or $w : V \to I\!N$ on the nodes. We denote by $\mathcal{T}_k$ the set of all k-cardinality trees in G. Then the node-weighted problem is

$$\min_{T_k \in \mathcal{T}_k} \sum_{v \in V(T_k)} w(v) \tag{1}$$

and the edge-weighted problem is

$$\min_{T_k \in \mathcal{T}_k} \sum_{e \in E(T_k)} w(e). \tag{2}$$

1.2 $I\!N\!P$-Hardness

Several authors have proved independently that the edge-weighted k-cardinality tree problem (2) is $I\!N\!P$-hard, see [32], [14], [30]. In [30] it has been shown that the problem (2) is still $I\!N\!P$-hard if $w(e) \in \{1, 2, 3\}$ for all edges e and $G = K_n$, but polynomially solvable if there are only two distinct weights. For the node-weighted case (1), $I\!N\!P$-completeness has been shown in [13] and [9].

Several authors have considered special types of graphs. Note first that the node-weighted problem (1) is trivially solved when $G = K_n$. The results are that the problem is polynomially solvable if G is a tree, see [13] for the node-weighted case and [24] for the edge weighted case. It should also be noted that when G is a tree, (1) and (2) are equivalent (see [11]). Polynomial time algorithms for the node-weighted problem exist for interval graphs and co-graphs, [31]. On the other hand $I\!N\!P$-completeness results for the node-weighted case have been obtained for grid and split graphs, [31].

The edge-weighted problem (2) is $I\!N\!P$-complete for planar graphs and for points in the plane, when edge weights correspond to distances between the points, see [30]. In the same paper polynomial algorithms for decomposable graphs and graphs with bounded tree-width have been given. The same holds for (2) in the plane, when all points lie on the boundary of a convex region. In [8], the authors have focussed on properties of the distance matrix. They have assumed that $G = K_n$ and have proved several results (both $I\!N\!P$-completeness and polynomial time solvability) on the complexity of the problem with graded distance matrices.

1.3 Algorithms

Concerning methodology, both exact and heuristic algorithms have been developed, with a general focus on approximation algorithms. We first note that integer programming formulations have been presented in [14] and later in [17]. Based on detailed studies of the associated polyhedron in the former paper a Branch and Cut algorithm has been developed and implemented in [16]. The code and also implementations of most of the heuristics in [11] are documented in [10]. A Branch and Bound method is described in [7].

The heuristics mentioned are based on greedy and dual greedy strategies and also make use of dynamic programming approaches. Other constructive heuristics have been presented in [7].

A large body of literature is available on approximation algorithms for the problem. The papers published on this topic represent an ongoing improvement until finally a constant approximation factor could be obtained. A first result appeared in [31], where it has been proved that there is an $O(\sqrt{k})$-approximation algorithm for the node-weighted problem on grid graphs. Later there were two streams of research articles: one focusing on the problem on graphs G, the other dealing with the problem in the plane.

In the former, [30] could obtain a $2\sqrt{k}$-approximation, which has been improved in [3] to $O(\log^2 k)$. A first constant factor approximation with factor 17 has been derived in [3]. The currently best approximation guarantee is 3, proved in [17].

[30] contains an $O(k^{\frac{1}{4}})$-approximation for the problem (2)in the plane. Improved algorithms, with ratio of $O(\log k)$ are due to [18] and [29]. Further decrease to $O(\frac{\log k}{\log \log n})$ [12] and a first constant factor approximation ([4] (with the constant not specified) followed. Using guillotine subdivisions [25] develops a $2\sqrt{2}$-approximation for the l_2 metric, and a 2-approximation for the l_1 metric. Finally, a polynomial time approximation scheme has been given in [2].

1.4 Local Search and Metaheuristics

More recently, authors have successfully applied local search methods to the k-cardinality tree problem: In [23] a simple tabu search strategy, in [6] both a genetic algorithm and a tabu search method have been presented for the edge-weighted case.

It is in this field, that our contribution lies. Up to now local search methods have been developed for the edge-weighted case. However, in the applications the node-weighted case allows a more natural formulation, e.g. in the oil-field leasing context, as a k-cardinality tree problem. Therefore, from now on, we will assume that $G = (V, E)$ has a weight function $w : V \rightarrow I\!N$ on the nodes, and we consider formulation (1).

The aim of this paper is to show that meta-heuristic techniques can be successfully applied to the k-cardinality tree problem. In contrast to constructive methods, they contain some random steps to avoid a fixed sequence of steps. We will be able to show that the results are of better quality than known constructive heuristics and are much faster than the integer programming method.

The evolution of meta-heuristic techniques began in the early 80's. They are mainly used for $I\!N\!P$-hard optimization problems. Informally, a general meta-heuristic method can be characterized as follows. A **meta-heuristic** is an iterative generation process using a subordinate heuristic by combining different intelligent techniques to explore a given solution space.

Examples for meta-heuristics are genetic algorithms, tabu search, neural networks and simulated annealing. In this paper, we use the ideas of genetic algorithms and tabu search to develop algorithms to generate solutions of the k-cardinality tree problem. This paper is based on a Master's thesis of C. Blum [5]. We shall assume the reader to be familiar with the basics of genetic algorithms and tabu search.

The rest of the paper is organized as follows. In Section 2 we will prove that the k-cardinality tree problem can be solved in polynomial time if G does not contain structures called hurdles or if it contains structures called troughs exactly once. In Section 3 we present a simple local search algorithm with a subroutine to guide out of local minima and show that it finds an optimal solution of (1) on graphs with a single trough.

In Section 4 we introduce the Tree-Crossbreeding-Algorithm (TCA), a meta-heuristic technique using concepts of genetic algorithms. In Section 5 we present the algorithm Tree-Tabu-Search (TTS), a meta-heuristic technique using concepts of tabu search. For both methods we will state that they find an optimal solution for the class of graphs described in Section 2. Finally in Section 6 we report the results of numerical tests of our methods TCA and TTS compared to the heuristics of [11] on the same set of examples.

2 A Polynomially Solvable Special Case

From now on, we will deal with the node-weighted case only. An instance of the k-cardinality tree problem is given by a set of feasible solutions, i.e. the set $\mathcal{T}_k$ of k-cardinality trees of a graph $G = (V, E)$ and a node weight $w : V \to I\!N$. An instance will be denoted by $(\mathcal{T}_k, w)$. For a graph $G = (V, E)$ let $S := (V(S), E(S))$ be a subgraph of G. We denote by

$$V_{NH}(S) := \{v \in V \setminus V(S) : \exists \tilde{v} \in V(S) \text{ s.t. } [v, \tilde{v}] \in E\}$$

the set of all nodes adjacent to nodes in S (the neighbourhood of S).

In this section, we address the question, under which conditions the problem can be solved in polynomial time. We show that the absence of certain structures in graphs gives such a condition. This condition guarantees that the problem can be solved in polynomial time and will serve as a quality criterion for the local search methods developed in Sections 3, 4, and 5. The next two definitions specify the structure elements *trough* and *hurdle* in undirected node-weighted graphs.

Definition 1 *A connected subgraph $S = (V(S), E(S))$ of an undirected, node-weighted graph $G = (V, E)$ is called* **trough**, *if:*

i) $w(v_i) = w(v_j),\ \forall v_i, v_j \in V(S)$

ii) $w(v_k) > w(v_s),\ \forall v_s \in V(S),\ \forall v_k \in V_{NH}(S)$

Definition 2 *A connected subgraph $H = (V(H), E(H))$ of an undirected, node-weighted graph $G = (V, E)$ is called* **hurdle**, *if:*

i) *It exists a $c \in I\!R$ with: $c \leq w(v_i)\ \forall v_i \in V(H)$*

ii) $|V_{NH}(H)| \geq 2$

iii) *There is at least one pair of nodes v_i, v_j in the neighbourhood of H with:*

  1) $w(v_i) < c$ *and* $w(v_j) < c$

  2) *The path with smallest node-weight from v_i to v_j in G contains at least one node $v_k \in V(H)$*

In order to be able to count the number of hurdles in a graph, we introduce the following definition.

Definition 3 *Let $G = (V, E)$ be an undirected, node-weighted graph.*

- *A hurdle H of G is called minimal, if there is no hurdle $\tilde{H}$ in G with $V(\tilde{H}) \subset V(H)$.*
- $\mathcal{H}(G) := \{H : H$ *is called a minimal hurdle in G* $\}$
- *The cardinality of $\mathcal{H}(G)$ is called the number of hurdles in G.*

Note that a node-weighted graph always contains a trough, containing (at least) one node of minimal weight. The presence of troughs and hurdles in a graph is related in the following way.

Theorem 1 *Let $G = (V, E)$ be an undirected, node-weighted graph. If $|\mathcal{H}(G)| = 0$ then G contains exactly one trough.*

Proof:

Let $G = (V, E)$ be an undirected, node-weighted graph and assume that G contains two troughs S_1, S_2. Then $V(S_1) \cap V(S_2) = \emptyset$. Thus there are two nodes $v_k \in V(S_1)$ and $v_l \in V(S_2)$ such that the shortest path $p_{v_k,v_l} = v_k, \ldots, v_l$ contains a node of a hurdle H. To see this note that the sequence of nodes in p_{v_k,v_l} must contain a node v_h such that

$$w(v_k) \leq \ldots \leq w(v_j) < w(v_{j+1}) \leq \ldots \leq w(v_h)$$

and

$$w(v_h) \geq \ldots \geq w(v_{i-1}) > w(v_i) \geq \ldots \geq w(v_l).$$

For $c := \max\{w(v_{j+1}), w(v_{i-1})\}$ the graph G contains at least one hurdle H_1. □

In the following discussion, it will be convenient to assume that the nodes are numbered according to increasing node weights. Thus, $w(v_1) \leq w(v_2) \leq \ldots \leq w(v_n)$. Additionally we introduce the following notation. Let $w_1 < w_2 < \ldots < w_l$ be the different weights, then

$$V(G) = \{v_{1,w_1}, \ldots, v_{s_1,w_1}, \ldots, v_{1,w_l}, \ldots, v_{s_l,w_l}\}$$

Here all nodes v_{i,w_j} have equal weights for $i = 1, \ldots, s_j$. We also specify the subsets of nodes with equal weight

$$V(G)_{w_i} := \{v_{1,w_i}, \ldots, v_{s_i,wi}\},\ i = 1, \ldots, l$$

We note that a graph with only one trough may still contain hurdles, see Figure 1 for an example. However, graphs with only one trough have the following property.

Lemma 1 *Let G be a graph with only one trough S_1. Let $V' \subset V$ be a subset of nodes such that $w(v') \leq w(v)$ for all $v' \in V'$ and all $v \in V \setminus V'$. Then there exists an edge between a node of V' and a node of minimal weight in $V \setminus V'$*

Proof:

By the assumption on V' there exist d and r such that $V' = \{v_{1,w_1}, \ldots, v_{r,w_d}\}$. Let $\bar{V}$ denote the nodes of minimal weight in $V \setminus V'$. Therefore either $\bar{V} = V(G)_{w_d} \setminus V'$ or (if $r = s_d$) $\bar{V} = V(G)_{w_{d+1}}$.

Assume that in G there is no edge between a node $v \in V'$ and a node $\bar{v} \in \bar{V}$. Then all paths in G starting from a node $v \in V'$ to a node $\bar{v} \in \bar{V}$ contain a node v_{j,w_p} with $w_p > w(\bar{v})$. Hence, there are at least two troughs in G, contradicting the fact, that S_1 is the only trough in G $\square$

From Lemma 1 the following result is immediate.

Lemma 2 *Let G be a graph with only one trough S_1. Then S_1 is the subgraph induced by $V(G)_{w_1}$.*

We now proceed to show that the k-cardinality tree problem can easily be solved by a greedy algorithm, if G contains exactly one trough.

- **INPUT:** An instance $(\mathcal{T}_k, w)$ of the k-cardinality tree problem
- $V_{gr} := \emptyset$
- **WHILE** there is a $v \in V \setminus V_{gr}$ such that $V_{gr} \cup \{v\}$ induces a connected subgraph of G and $|V_{gr}| < k + 1$ **DO**

- – Choose $v \in V \setminus V_{gr}$ s.t. $V_{gr} \cup \{v\}$ induces a connected subgraph of G with minimal weight
  – $V_{gr} := V_{gr} \cup \{v\}$
- Choose a spanning tree T_{gr} of V_{gr}.
- **OUTPUT:** A k-cardinality tree T_{gr}

Algorithm 2.1: Greedy Algorithm

The Greedy Algorithm 2.1 is a heuristic for the k-cardinality tree problem. If G contains exactly one trough, it finds an optimal solution.

Theorem 2 *Let $G = (V, E)$ be an undirected, node-weighted graph with exactly one trough S_1, and $0 \leq k \leq n - 1$. Then the greedy algorithm solves the k-cardinality tree problem in $O(n^2)$ time and the solution T_{gr} satisfies the following property:*

$$w(v) \leq w(\tilde{v}) \;\; \forall v \in V_{gr} \;\; \forall \tilde{v} \in V \setminus V_{gr} \tag{3}$$

Proof:

The proof is by induction over the cardinality k.

First, let $k = 0$. In the one and only iteration of the greedy algorithm a node $v_{i,w_1} \in V(G)_{w_1}$ will be chosen. Clearly this is the optimal solution and (3) is fulfilled.

Let $k = h+1$ now and assume that the claim is true for $k \leq h$. According to the induction hypothesis after $h+1$ iterations the greedy algorithm has computed a set V_{gr} of $h+1$ nodes such that V_{gr} induces a connected subgraph of G, T_{gr} is a minimal weight h-cardinality tree, and V_{gr} fulfills (3).

Then, according to Lemma 1 there exists an edge between a node in V_{gr} and a node of minimal weight in $V \setminus V_{gr}$. Therefore the greedy algorithm will choose such a node $\bar{v}$ in iteration $h + 2$, $V_{gr} \cup \{\bar{v}\}$ has minimal weight among all connected subgraphs of G with $h + 2$ nodes. Furthermore (3) is still fulfilled. $\square$

As an immediate consequence of Theorem 2 we obtain the following Corollary.

Corollary 1 *Let G be a graph with exactly one trough and assume that all weights are different. Then the set of global optimal solutions of (1) is the set of all spanning trees of the subgraph of G induced by $\{v_1, \ldots, v_{k-1}\}$.*

Note that a complete graph contains exactly one trough, and according to Theorem 2 the k-cardinality tree problem can be solved in polynomial time on complete graphs. As we have reported earlier, the k-cardinality tree problem is also polynomially solvable on trees. Therefore very sparse and very dense graphs are the "easy" cases, whereas intermediate densities are difficult to handle. This behaviour, reported in earlier papers, see [11], can now be explained by the likely presence of multiple troughs. Since exact solutions can only be found for rather small instances (up to 30 nodes) in reasonable time, but practically relevant problems are much bigger (300 nodes in the oil-field leasing problem) one has to resort to heuristics.

We used local search methods built on knowledge of troughs and hurdles in graphs to obtain better solutions than the constructive heuristics known until now. In the following sections we will describe these methods.

3 A Simple Local Search Algorithm

The most important ingredient of a local search method is the neighbourhood function. Let T_k be a k-cardinality tree. Then $\mathcal{N}_{Swap}(T_k)$ is consisting of all k-cardinality trees which can be computed from T_k by removing one leaf v_l of T_k and adding one node of the neighbourhood of $V_{NH}(T_k \setminus v_l)$. This neighbourhood function has the advantage to be easy to compute, but has the disadvantage of providing a lot of bad local minima.

In order to guide the basic local search out of local minima, we added the following mechanism, called Edge-Permutation-Algorithm (EPA). To explain the idea, let us assume that T_k^{LM} is a local minimum according to the neighbourhood function $\mathcal{N}_{Swap}$. The Edge-Permutation-Algorithm 3.1 takes all the nodes of T_k^{LM} and all those edges of $E(G)$ joining two nodes of $V(T_k^{LM})$. Now we try to compute a new k-cardinality tree T_k consisting of the same nodes as T_k^{LM}, but not lying in a local minimum of $\mathcal{N}_{Swap}$. See Figure 1 for an example.

- **INPUT:** A k-cardinality tree T_k
- $E_{all} := \{e_{st} : v_s, v_t \in V(T_k), e_{st} \in E(G)\}$, $V_{all} := V(T_k)$
- $E(T_k^{EP}) := \emptyset$, $V(T_k^{EP}) := \emptyset$
- $T_k^{EP} := T_k^{EP} +$ argmin$\{w(v_s) : v_s \in V_{all}\}$
- $V_{all} := V_{all} \setminus V(T_k^{EP})$
- **REPEAT**
  - $v_{in} :=$ argmin$\{w(v_i) : v_i \in V_{NH}(T_k^{EP}) \cap V_{all}\}$

- $v_{source} := \text{argmin}\{w(v_i) : v_i \in V(T_k^{EP}) \cap I_V(v_{in}, T_k)\}$
- $T_k^{EP} := T_k^{EP} + v_{in} + e[v_{in}, v_{source}]$
- $V_{all} := V_{all} \setminus \{v_{in}\}$
- $E_{all} := E_{all} \setminus \{e[v_{in}, v_{source}]\}$

- **UNTIL** $(V_{all} = \emptyset)$
- **OUTPUT:** A k-cardinality tree T_k^{EP}

Algorithm 3.1: Edge-Permutation-Algorithm (EPA)

Grid Graph, $n = 9, k = 4$

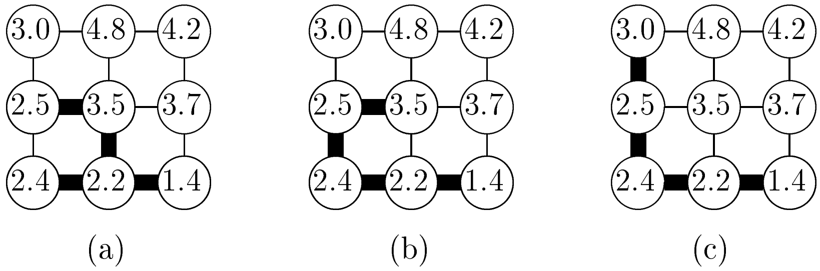

Figure 1: The Edge-Permutation-Algorithm

In Figure 1, (a) shows a 4-cardinality tree T_4 lying in a local minimum according to $\mathcal{N}_{Swap}$, (b) shows the 4-cardinality tree T_4^{EP}, the result of EPA not lying in a local minimum, and (c) shows a 4-cardinality tree $T_4 \in \mathcal{N}_{Swap}(T_4^{EP})$ better than T_4^{EP} according to the weight function w.

Algorithm 3.2 below shows the local search method Tree-Local-Search (TLS). The algorithm can be initiated by any feasible solution, e.g. a greedily generated one and used as a heuristic by itself. However, we will use it only as a subroutine for the Tree-Crossbreeding-Algorithm and the Tree-Tabu-Search method in Sections 4 and 5.

- **INPUT:** A k-cardinality tree T_k
- **REPEAT**
  - $T_k^s := \text{argmin}\{w(T_k^i) - w(T_k) : T_k^i \in \mathcal{N}_{Swap}(T_k)\}$
  - stopflag := true
  - **IF** $(w(T_k^s) < w(T_k))$ **THEN**

    * $T_k := T_k^s$, stopflag := false
  - **ELSE**
    * $T_k^{EP} :=$ **Edge-Permutation-Algorithm**(T_k)
    * **IF** T_k^{EP} is not a local minimum according to $\mathcal{N}_{Swap}$ **THEN**
      · $T_k := T_k^{EP}$
      · stopflag := false

- **UNTIL** (stopflag = true)
- **OUTPUT:** A local minimum $T_k^{LO} := T_k$.

Algorithm 3.2: Tree-Local-Search (TLS)

Now we prove that TLS finds a globally optimal solution of the k-cardinality tree problem, starting from an arbitrary feasible solution, whenever G is a graph with exactly one trough, and $w(v_i) \neq w(v_j)$ for all $v_i \neq v_j$. Then $S_1 = \{v_1\}$ is the only trough. For the rest of this section, we will make these assumptions. We need some technical Lemmas first, which show the relations between locally and globally optimal solutions. Note that even for graphs with exactly one trough, there exist local optima, which are not global ones, see Figure 1 for an example.

The first result says, that for a locally optimal solution, the weights of all leafs are less than the weights of all adjacent nodes.

Lemma 3 *Let T_k^{LO} be a local minimizer of (1) with respect to neighbourhood $\mathcal{N}_{Swap}$. Then $w(v_h) > w(v_l)$ for all $v_h \in V_{NH}(T_k^{LO})$ and all leafs $v_l \in V_L(T_k^{LO})$.*

Proof:

Assume that there is $v_j \in V_{NH}(T_k^{LO})$ with $w(v_j)$ less than the weight of the heaviest leaf of T_k^{LO}, v_l^{max}. Then, if there exists an edge e_{v_i,v_j} with $v_i \in V(T_k^{LO}) \setminus \{v_l^{max}\}$, one could exchange v_l^{max} and its incident edge in T_k^{LO} with v_i and e_{v_i,v_j} to obtain a k-cardinality tree with lower weight, a contradicition to the choice of T_k^{LO}.

Therefore no such edge exists and $e_{v_j,v_l^{max}}$ is the only edge between v_j and $V(T_k^{LO})$. Therefore all paths $p_{v_j v_s}$ from v_j to a leaf $v_s \in V_L(T_k^{LO}) \setminus \{v_l^{max}\}$ contain either v_l^{max} or a node $v_h \in V_{NH}(T_k LO) \setminus \{v_j\}$. Since $w(v_l^{max}) > w(v_j)$ and $w(v_h) > w(v_l^{max})$ (otherwise the first case would apply), this implies the existence of at least two troughs, a contradiction. $\square$

Now we can show that local and global optimal solutions always have nodes in common.

Lemma 4 *Let T_k^{LO} be a local and T_k^{GL} be a global minimizer of (1) with respect to neighbourhood $\mathcal{N}_{Swap}$. Then*

$$V(T_k^{GL}) \cap V(T_k^{LO}) \neq \emptyset \quad (4)$$
$$V_L(T_k^{LO}) \subset V(T_k^{GL}) \quad (5)$$

Proof:

Suppose $V(T_k^{GL}) \cap V(T_k^{LO}) = \emptyset$. Then all paths p_{v_l,v_1} from a leaf $v_l \in V_L(T_k^{LO})$ to v_1, which is contained in the only trough and in $V(T_k^{GL})$, contain a node $v_h \in V_{NH}(T_k^{LO})$. From Lemma 3 $w(v_l) < w(v_h)$ and $w(v_h) > w(v_1)$. This implies the existence of a second trough, which is a contradiction.

Now assume that there is at least one node $v_l \in V_L(T_k^{LO}) \setminus V(T_k^{GL})$. Since $V(T_k^{GL}) \cap V(V_k^{LO}) \neq \emptyset$ there is a node $v_h \in V(T_k^{GL}) \cap V_{NH}(T_k^{LO})$ with $w(v_h) < w(v_l)$, a contradiction to Lemma 3. □

Lemma 5 *Let $v_l := argmax\{w(v_s) : v_s \in V_L(T_k^{LO}) \cap V(T_k^{GL})\}$, then*

$$v_i \in V(T_k^{LO}) \textit{ for } i = 1, \ldots, l-1. \quad (6)$$

Proof:

Assume the contrary. Then there exists a node $v_j \in \{v_1, \ldots, v_{l-1}\} \setminus V(T_k^{LO})$. Therefore there exists a node $v_h \in \{v_1, \ldots, v_{l-1}\} \cap V_{NH}(T_k^{LO})$. Since $w(v_h) < w(v_l)$ this is a contradiction to Lemma 3. □

With the preceding Lemmas 3 to 5, we can prove that the Edge-Permutation-Algorithm 3.1 indeed leads out of local minima.

Theorem 3 *Let T_k^{LO} be a local minimizer, which is not globally optimal Then the result T_k^{EP} of applying the Edge-Permutation-Algorithm to T_k^{LO} is not a local minimizer.*

Proof:

Let $v_l := \text{argmax}\{w(v_s) : v_s \in V_L(T_k^{LO}) \cap V(T_k^{opt})\}$. From Lemma 5 we conclude that $\{v_1, \ldots, v_{l-1}\}$ is a subset of $V(T_k^{LO})$.

Starting the Edge-Permutation-Algorithm with T_k^{LO} v_1 is chosen first and v_{i+1} is added after v_i until for some $j \geq l$ v_{j+1} is not contained in $V(T_k^{LO})$. For the tree T_{j-1} constructed then it holds $V(T_{j-1}) = V(T_{j-1}^{GL})$ (according to Corollary 1). Since $v_{j+1} \notin V(T_k^{LO})$ it follows that $v_{j+1} \in V_{NH}(T_k^{EP})$. The Edge-Permutation-Algorithm will only add nodes v_r to T_{j-1} for which $w(v_r) > w(v_{j+1})$. Furthermore at least one more leaf v_s is added and we have $w(v_s) > w(v_{j+1})$ for this leaf, too. Since $v_{j+1} \in V_{NH}(T_k^{EP})$, T_k^{EP} is not a local minimizer. □

Theorem 3 is the essential tool to show that TLS solves the k-cardinality tree problem on graphs with one trough and all weights different.

Theorem 4 *Let $G = (V, E)$ be an undirected, node-weighted graph with exactly one trough S_1 and $w(v_i) \neq w(v_j)\ \forall\ v_i \neq v_j$, i.e. $S_1 = \{v_1\}$. Then the Tree-Local-Search Algorithm 3.2 finds an optimal solution T_k^{GL} for (1) for an arbitrary starting solution T_k in polynomial time.*

Proof:

The Edge-Permutation-Algorithm is only applied, when a locales minimizer T_k^{LO} has been obtained. Then it constructs a tree T_k^{EP}, containing the same nodes as T_k^{LO}, but which according to Theorem 3 is not a local minimizer. After finitely many iterations a globally optimal solution must be obtained.

TLS performs at most $n - 2k$ iterations without calling EPA, until $|V(T_k) \cap V(T_k^{GL}| \geq 2$. Then at most $k-1$ iterations with EPA are necessary to obtain a globally optimal solution. EPA itself runs in $O(k^2)$ time. One iteration of TLS without EPA needs at most $O(k(n-k))$ operations. Thus, the overall time is at most $O(n^2k + nk^4)$. □

4 Tree-Crossbreeding-Algorithm (TCA)

The meta-heuristic Tree-Crossbreeding-Algorithm (TCA) is based on the basic parallel genetic algorithm introduced by Mühlenbein [26] in 1991. There are two main differences between a normal genetic algorithm and a parallel genetic algorithm:

- To find a crossbreeding mate for a given individual we search only among those individuals in a neighbourhood of the given individual. In terms of k-cardinality trees as individuals we have to search a crossbreeding mate just among those k-cardinality trees with at least one node in common with the given k-cardinality tree. The result of this strategy is a parallel search in different search directions.
- As a mutation operator we apply a local search method to all individuals build by crossover. Thus an individual improves its fitness during its lifetime.

So we can take benefit of the following properties:

- For the selection of a crossbreeding mate for a given individual it is not necessary to know the fitness of all individuals of the current population, because we search the crossbreeding mate only in a neighbourhood of the given individual. So the central control over the algorithm is lost and we get different search directions guiding to different local minima. But those search directions are not independent, because the neighbourhoods are overlapping. So we have an exchange of information between the different search directions.

- It is possible to implement a parallel genetic algorithm on parallel computers. However, we did not make use of this possibility.

- The use of local search as a mutation operator leads to better computation times.

Below, we describe the main steps of the procedure.

4.1 Generation of the Starting Population

We generate the starting population by by constructing randomly a k-cardinality tree starting from each of the n nodes of G. This may not lead to n different k-cardinality trees, but it ensures that all n nodes of G appear in some k-cardinality tree contained in the starting population.

4.2 The Selection Operator of TCA

For two k-cardinality trees to be crossbred we require them to have at least one node in common. So the selection mechanism has to find a crossbreeding mate for a given k-cardinality tree T_k in the neighbourhood of this individual consisting of all k-cardinality trees contained in the population which have at least one node in common with T_k.

Definition 4 *Let $(\mathcal{T}_k, w)$ be an instance of the k-cardinality tree problem and let $\mathcal{P}_{\mathcal{T}_k}$ be a population of k-cardinality trees. Then $\mathcal{N}_{CB} : \mathcal{P}_{\mathcal{T}_k} \rightarrow 2^{\mathcal{P}_{\mathcal{T}_k}}$ is defined in the following way:*

$$\mathcal{N}_{CB}(T_k^i) := \{T_k \in \mathcal{P}_{\mathcal{T}_k} : V(T_k^i) \cap V(T_k) \neq \emptyset\}$$

For a given k-cardinality tree T_k^i the crossbreeding mate is chosen by roulette-wheel-selection from the set $\mathcal{N}_{CB}(T_k^i)$.

- Generate a random number x with $0 \leq x \leq \sum_{T_k \in \mathcal{N}_{CB}(T_k^i)} w(T_k)$

- Continue choosing k-cardinality trees from $\mathcal{N}_{CB}(T_k^i)$ and summing up their weights until this sum is equal or greater than x. Take the last chosen k-cardinality tree T_k^{CB} as the crossbreeding mate of T_k^i.

Obviously this selection mechanism realizes the evolutionary principle of survival of the fittest. In each iteration of TCA for all k-cardinality trees T_k^i in the current population a crossbreeding mate will be selected in this manner. If it is not possible to find a crossbreeding mate, the k-cardinality tree will not appear in the next generation again. This, in a sense, means the end of one search direction which we cancel from the further process.

The selection mechanism implies to two different stopping criteria. Due to the phenomenon mentioned above the shrinking of the population size during the search process is possible. Therefore one stopping criterion is that a population contains only one k-cardinality tree. The second stopping criterion depends on the success of the different search directions. If in the current iteration of TCA none of the generated offspring is better (according to w) than its first parent, then none of the search directions was successful and we will stop the algorithm.

4.3 The Crossover Operators of TCA

After the selection process we have pairs (T_k^i, T_k^{CB}) of k-cardinality trees to be crossbred now. TCA uses two different crossover operators, the Union-Crossover and the Intersection-Crossover. So one pair of parents leads to two offspring, T_k^{UC} the result of Union-Crossover and T_k^{IC} the result of Intersection-Crossover. We will define T_k^C as the better of the two.

$$T_k^C := \operatorname{argmin}\{w(T_k^{UC}), w(T_k^{IC})\}$$

If T_k^C is better than T_k^i according to the weight function w, we take T_k^C as a member of the next generation, else we take T_k^i as a member of the next generation (replication). Algorithm 3.1 formalises the two crossover operators.

- **INPUT:** A pair (T_k^1, T_k^2) of k-cardinality trees
- $V_\cup := V(T_k^1) \cup V(T_k^2)$, $V_\cap := V(T_k^1) \cap V(T_k^2)$, $E(T_k) := \emptyset$
- $V(T_k) := \operatorname{argmin}\{w(v_i) : v_i \in V_\cap\}$
- **REPEAT**
  - Choose V_{pref}, see (7) and (8)
  - **IF** $V_{pref} = \emptyset$ **THEN**
    - $v_h := \operatorname{argmin}\{w(v_s) : v_s \in V_{NH}(T_k) \cap V_\cup\}$
  - **ELSE**
    - $v_h := \operatorname{argmin}\{w(v_s) : v_s \in V_{pref}\}$
  - $T_k := T_k + v_h + e[v_h, v_l]$, with $v_l \in V(T_k)$
- **UNTIL** $|V(T_k)| = k + 1$
- **OUTPUT:** A k-cardinality tree T_k

Algorithm 4.1: Union-Crossover and Intersection-Crossover

It remains to specify Line 5 in Algorithm 3.1. For Union-Crossover we set

$$V_{pref} := V_{NH}(T_k) \cap (V_{\cup} \setminus V_{\cap}). \tag{7}$$

So Union-Crossover crossbreeds two k-cardinality trees T_k^1 and T_k^2 by constructing an offspring T_k using the whole node set $V(T_k^1) \cup V(T_k^2)$. The starting node is chosen as the minimal node according to the weight function in the node set $V(T_k^1) \cap V(T_k^2)$. Besides, during the construction of the new k-cardinality tree T_k we prefer nodes from the node set $(V(T_k^1) \cup V(T_k^2)) \setminus (V(T_k^1) \cap V(T_k^2))$ against nodes from the node set $V(T_k^1) \cap V(T_k^2)$. This means we try to prefer the good properties – in our terms low weighted nodes – appearing in only one of the parents.

For Intersection-Crossover we specify Line 5 as

$$V_{pref} := V_{NH}(T_k) \cap V_{\cap}. \tag{8}$$

Contrary to Union-Crossover we now prefer nodes from the node set $V(T_k^1) \cap V(T_k^2)$ against nodes from the node set $(V(T_k^1) \cup V(T_k^2)) \setminus (V(T_k^1) \cap V(T_k^2))$ during the construction of the new k-cardinality tree T_k. This means we try to keep the good properties appearing in both parents if possible.

4.4 The Mutation Operator of TCA

As mentioned above we use a local search heuristic as mutation operator. We apply the Tree-Local-Search (TLS) Algorithm 3.2 presented in detail in Section 3.

Putting together all ingredients, we can formulate the Tree-Crossbreeding-Algorithm 4.2.

- **INPUT:** An instance $(\mathcal{T}_k, w)$ of the k-cardinality tree problem
- $t := 0$
- Generate the starting population $P_{\mathcal{T}_k}(t)$
- $T_k^{best} := \text{argmin}\{w(T_k^i) : T_k^i \in P_{\mathcal{T}_k}(t)\}$
- **REPEAT**
  - $t := t + 1$
  - $P_{\mathcal{T}_k}(t) :=$ **Selection-Crossover**$(P_{\mathcal{T}_k}(t-1))$

- $P_{\mathcal{T}_k}(t) :=$ **Tree-Local-Search**$(P_{\mathcal{T}_k}(t))$
- $T_k^t := \text{argmin}\{w(T_k^i) : T_k^i \in P_{\mathcal{T}_k}(t)\}$
- **IF** $w(T_k^t) < w(T_k^{best})$ **THEN** $T_k^{best} := T_k^t$

- **UNTIL** $(P_{\mathcal{T}_k}(t) = \emptyset)$ or $(P_{\mathcal{T}_k}(t) \subseteq P_{\mathcal{T}_k}(t-1))$
- **OUTPUT:** A k-cardinality tree $T_k^{sol} = T_k^{best}$

Algorithm 4.2: Tree-Crossbreeding-Algorithm (TCA)

As a consequence of using TLS as mutation operator, Theorem 4 implies Theorem 5.

Theorem 5 *Let $G = (V, E)$ be a graph with exactly one trough and $w : V \to I\!R_+$ be a weight function on the nodes such that $w(v_i) \neq w(v_j)$, whenever $i \neq j$. Then TCA finds a minimum weight k-cardinality tree in polynomial time, irrespective of the initial solution (starting population).*

For computational results of TCA we refer to Section 6.

5 Tree-Tabu-Search (TTS)

Tabu search techniques have been described independently by Glover [19], [20] and Hansen [22]. Glover defines Tabu Search as *" ... a higher level heuristic procedure for solving optimization problems, designed to guide other methods (or their component processes) to escape the trap of local optimality."* In the years since then tabu search has been applied successfully for many combinatorial optimization problems, see [1] and [27] for extensive references.

In this section we describe a Tabu Search procedure for the k-cardinality tree problem. Its main features are the use of two tabu lists with gaps, i.e. for moves that are tabu, but happen to fall in a gap of a tabu list, the tabu status is overruled. Furthermore, we use an intensification strategy based on second best moves and we apply the Edge-Permutation-Algorithm of Section 4.4 to overcome local minima.

5.1 Neighbourhood and Moves

We use the neighbourhood $\mathcal{N}_{Swap}$ already defined in Section 3. Note that this choice guarantees that the graph on which the local search operates is connected. According to the neighbourhood, a move m_T is defined as a 4-tuple $m_T = (v_{in}, e_{in}, v_{out}, e_{out})$, where v_{in}

and e_{in} are the node and edge added to the tree and v_{out} and e_{out} define the leaf and its incident edge to be deleted from the tree. The application of a move to a k-cardinality tree T_k is describes as follows:

$$T_k \oplus m_T := (V(T_k) \cup \{v_{in}\} \setminus \{v_{out}\}, E(T_k) \cup \{e_{in}\} \setminus \{e_{out}\}) \,.$$

In each step, a best non tabu move is chosen. Let the weight of a move be $w(m_T) := w(v_{in}) - w(v_{out})$, then we choose a non tabu move with smallest weight. The resulting tree $T_k \oplus m_T$ will have weight $w(T_k) + w(m_T)$.

5.2 Tabu Lists

We use two tabu lists TL_{in} and TL_{out} of equal length. Both lists store the s_{tl} most recent nodes v_{in}, respectively v_{out} that have been added to (deleted from) the current tree T_k. Experiments have shown that dynamic lists yield better results than static ones. We implemented dynamic lists using gaps G_{in} and G_{out}. A gap is realized as a tuple (g_1, g_2), where g_1 is the position of the list, where the gap starts and g_2 is the length of the gap. Moves falling into a gap are considered non tabu.

Consequently, a move m_T is tabu if $v_{in} \in TL_{in}$ or $v_{out} \in TL_{out}$ but not both are in the gaps. The length of the tabu lists is set as

$$s_{tl} := \max\left\{4, \min\left\{\left\lfloor \frac{k}{3} \right\rfloor, \left\lfloor \frac{n-k}{3} \right\rfloor\right\}\right\}.$$

This guarantees that the list is not too short (for very small or very large k). The choice of the minimum ensures that the lists do not get too large. The longest lists are encountered when $k = n/2$, for which the problem is known to be hardest (cf. Section 6). We found that this definition yields good results for all values of k.

5.3 The Diversification Phase

The whole procedure is controlled by two types of counters f_s and f_g, the diversification and global frequencies, for each node. The global frequency $f_g(v_i)$ is incremented whenever a new tree is encountered, the diversification frequency $f_s(v_i)$ only if a new solution is generated at the beginning of a diversification phase.

Initially all counters are equal to zero. When a diversification phase starts, tabu lists are initialised, gaps are set to begin at position one with zero length and a k-cardinality tree is computed greedily according to global frequencies $f_g(v_i)$ (in particular, the whole algorithm begins with a randomly generated tree). This tree is then passed on to the intensification phase.

The whole algorithm stops, when the average frequency f_s is greater than two. For large values of k, i.e. bigger trees, this stopping criterion will be achieved after fewer diversifications than for small k. On the other hand, larger k will imply that a larger part of the solution space will be explored during intensification, and fewer diversifications are necessary. Thus the stopping criterion makes for a balance between diversification and intensification.

- **INPUT:** An instance $(\mathcal{T}_k, w)$ of the k-cardinality tree problem
- $T_k^{ts} := nil$, $level2 := true$
- $s_{tl} := \max\{4, \min\{\lfloor \frac{k}{3} \rfloor, \lfloor \frac{n-k}{3} \rfloor\}\}$
- Initialise counters f_g and f_s to 0
- **WHILE** (level2 = true) **DO**
  - Initialise tabu lists TL_{in}, TL_{out}
  - Initialise gaps $G_{in} := (1, 0)$, $G_{out} := (1, 0)$
  - Compute a k-cardinality tree T_k^s according to f_g
  - $M_T := \{m_T : T_k^s \oplus m_T \in \mathcal{N}_{Swap}(T_k^s)\}$
  - Increase f_g and f_s according to T_k^s
  - **IF** $(w(T_k^s) < w(T_k^{ts}))$ or $(T_k^{ts} =$ nil$)$ **THEN** $T_k^{ts} := T_k^s$
  - $T_k^{l1} := T_k^s$
  - M-Sit $:= nil$
  - $t := 0$, $impr := 0$, $sb := 0$
  - $level1 := true$
  - **TTS-Level1**
  - **IF** $f_s > 2$ on average **THEN** $level2 := false$
- **OUTPUT:** A k-card-Tree T_k^{ts}

Algorithm 5.1: Tree-Tabu-Search (TTS-Level2): Diversification

5.4 The Intensification Phase

In the intensification phase new windows are set, in case the tabu lists are full. Then we test, if the current solution T_k is a local minimum, i.e. the weight of all moves $w(m_T) \geq 0$. In this case, we apply the Edge-Permutation-Algorithm described in Section 4.4 to overcome the local minimum. Then we compute the best move (irrespective of tabu status). Should it be tabu but yield a better result than the best solution obtained so far, tabu status is overruled. Otherwise we apply the best non tabu move. In any case, we record the second best non tabu move and the situation in which it occurred.

The best second best strategy works as follows. We always store the status of the complete process for the best second best move encountered in the current intensification phase. We use a counter *impr*, which counts the number of iterations without improvement. Should this counter achieve a threshold $\lfloor \frac{k}{3} \rfloor$ we stop the process and continue at the situation stored for the best second best move and apply this move. Thus we abandon an apparently unsuccessful direction, go back to an older situation and continue with another promising search direction.

Besides *impr* two counters control the intensification. Counter t determines when new gaps are computed. This happens, whenever $t \bmod \lfloor \frac{s_{tl}}{2} \rfloor = 0$. Both length and starting position of the gaps are chosen randomly, given that the lengths are not greater than half the list length. Counter sb is incremented whenever we apply the best second best move. Intensification is stopped when sb reaches a threshold $\lfloor \frac{k}{3} \rfloor$.

- **WHILE** $level1 = true$ **DO**
  - **IF** $((t \bmod \lfloor \frac{s_{tl}}{2} \rfloor) = 0)$ and (TL_{in} and TL_{out} full) **THEN**
    - Set new gaps G_{in} and G_{out}
  - $impr := impr + 1$, $t := t + 1$
  - **IF** $w(M_T) \geq 0 \; \forall m_T \in M_T$ **THEN**
    - $T_k^s :=$ **Edge-Permutation-Algorithm**(T_k^s)
    - $M_T := \{m_T : T_k^s \oplus m_T \in \mathcal{N}_{Swap}(T_k^s)\}$
  - $m_T^1 := \text{argmin}\{w(m_T) : m_T \in M_T\}$
  - $M_T^{nt} := \{m_T : m_T \in M_T, m_T \text{ non tabu}\}$
  - (* Overrule criterion *)
    IF (m_T^1 is tabu) **THEN**
    - **IF** $(w(T_k^s) + w_m(m_T^1)) \geq w(T_k^{ts})$ **THEN**
      - $m_T^1 := nil$
      - $m_T^1 := \text{argmin}\{w(m_T) : m_T \in M_T^{nt}\}$
  - $m_T^2 := nil$

- $m_T^2 := \text{argmin}\{w(m_T) : m_T \in M_T^{nt}, m_T \neq m_T^1\}$
- **Best-SecondBest**
- **IF** $(sb - 1) > \lfloor \frac{k}{3} \rfloor$ **THEN** $level1 := false$
  ELSE
  * $T_k^s := T_k^s \oplus m_T^1$
  * **FOR** all $v_i \in V(T_k^s)$ **DO** $f_g(v_i) := f_g(v_i) + 1$
  * **IF** $w(T_k^s) < w(T_k^{l1})$ **THEN**
    · $impr := 0$
    · $T_k^{l1} := T_k^s$
  * **IF** $w(T_k^s) < w(T_k^{ts})$ **THEN** $T_k^{ts} := T_k^s$
  * Change TL_{in} and TL_{out} according to m_T^1
  * $M_T := \{m_T : T_k^s \oplus m_T \in \mathcal{N}_{Swap}(T_k^s)\}$

Algorithm 5.2: Tree-Tabu-Search (TTS-Level1): Intensification

As for Algorithm TCA in Section 4 we are able to prove a quality criterion for the TTS algorithm 5.1/5.2. Because we use the Edge-Permutation-Algorithm in the intensification phase, we have the following result.

Theorem 6 *Let G be a graph with exactly one trough and such that $w(v_i) \neq w(v_j)$ whenever $i \neq j$, then TTS solves the k-cardinality tree problem in polynomial time.*

6 Computational Results

We carried out numerical tests to establish the quality of our algorithms. The methods were compared with both optimal solutions obtained by integer programming methods described in [14] and the heuristics proposed in [11]. In the latter paper both a greedy strategy as a generalization of Prim's algorithm for finding a minimal spanning tree and a dual greedy strategy, which eliminates nodes with high weights from the graph until a k-cardinality tree remains have been developed.

We used sets of 20 randomly generated connected graphs and grid graphs, with 10, 20, and 30 nodes, respectively. The general graphs had between 149 and 277 edges, the grid graphs between 39 and 47 edges for the 30 nodes test set. The weights were randomly generated integers between one and 100, and we tested all cardinalities k with $2 \leq k \leq n-2$. (Actually, this was the same set of examples used in [11].) The limitations are due to the fact, that larger instances could not be solved optimally in a reasonable amount of CPU time. The number of tests carried out therefore is 140 for 10 nodes, 340 for 20 nodes and 540 for 30 nodes, and this for both general and grid graphs.

In Table 1 we list the percentage of instances in which our heuristics found an optimal solution.

	TCA general	TTS general	TCA grid	TTS grid
n=10	100	100	100	99
n=20	99	100	96	96
n=30	100	100	95	90

Table 1: Percentage of Instances Solved Optimally (Rounded)

The following observations are made. Results on general graphs are better than on grid graphs. This behaviour (observed earlier in [11]) can now be explained by the number of troughs. Dense graphs tend to have only few troughs (note that K_n contains a single trough), a fact that due to the results of Section 2 is expected to enhance performance. The grid graphs, with fewer edges will likely have many troughs and the algorithms will not work as well. Not surprisingly the results are better for small than for big instances. Note also that on complete graphs (i.e. maximal density, and minimal number of troughs) the problem is polynomially solvable by our methods.

In Figures 2 and 3, we illustrate the performance of TCA and TTS as compared to the greedy (KCP, for k-Cardinality-Prim) and dual greedy (DG1) approaches from [11]. The scale is the deviation from optimality in percent. It turns out that TCA and TTS uniformly perform better than the constructive heuristics, with a mean deviation of less than 0.1 % for general graphs and less than 1 % for grid graphs.

It can also be seen that the performance improves as k increases from 10 to 30. This behaviour reflects expectations, as for large k the problem is closer and closer to the spanning tree problem, which can be solved in polynomial time. For very small k, the solution only consists of a few edges, and good solutions are obtained, too.

We also wanted to test the stability of the algorithms and ran 20 repetitions of both TTS and TCA for all cardinalities on a grid graph with 30 nodes (i.e. the case with worst results). The variance of the solution values obtained was at most 0.06 %.

Concerning computation times, we remark that it was not our main goal to have the implementations fine-tuned for speed. Clearly, computation times were lower for grid graphs than for general graphs, with TTS always being slower than TCA. The difference between the two was more significant for general graphs with TCA twice as fast as TTS, for grid graphs the difference was almost constant. Computation times reached a maximum of 100 (general graphs, $k = 23$), respectively 12 seconds (grid graphs, $k = 24$) on a Pentium PC with 100 MHz.

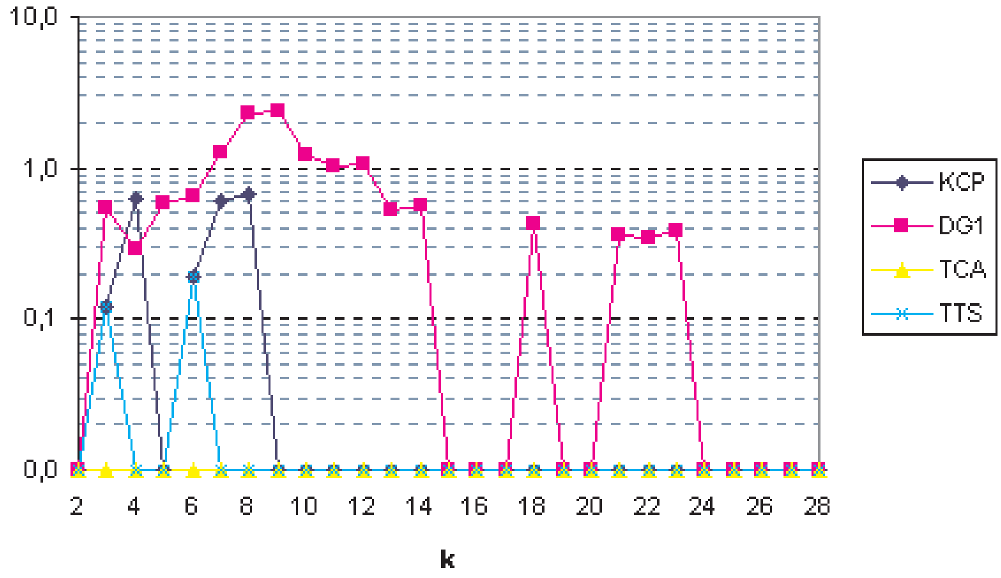

Figure 2: Deviation from Optimality in Percent, General Graphs

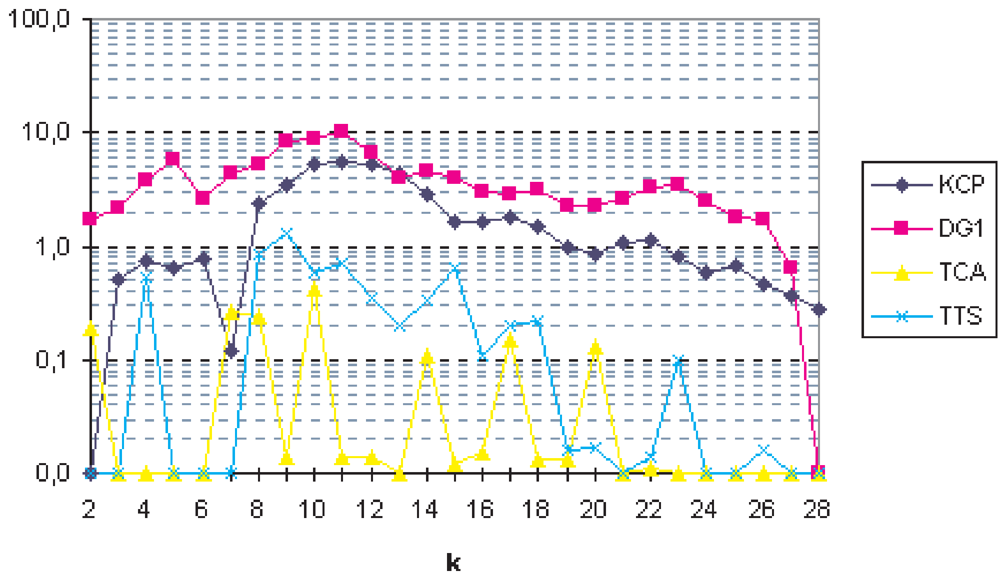

Figure 3: Deviation from Optimality in Percent, Grid Graphs

7 Conclusions

In this paper we have investigated the $\mathbb{NP}$-complete k-cardinality tree problem on node-weighted graphs. We have proved that the problem is solvable in polynomial time by the greedy algorithm on the class of graphs containing exactly one trough. The main purpose of the paper was to develop local search methods that yield good solutions. We presented both a genetic algorithm (TCA) and a tabu search strategy (TTS), and were able to prove that both solve the problem efficiently and optimally on the class of graphs introduced before.

Computational tests indicate that the algorithms provide very good, often optimal solutions for the problem in acceptable time (given the slow computer we had to use). It turned out that the problem is most difficult for cardinalities k approximately half the number of nodes, confirming earlier observations by other authors. Also tests confirmed the expectation that higher density of the graphs tends to make the problem easier, a behaviour that can be explained by the structural results presented.

To summarise, we can say that the algorithms discussed are good tools to solve the problem in applications mentioned in the introduction, taking into account that all applications discussed so far involve strategic rather than online decisions. As a direction of future research, we mention the generalization of the problem to connected subgraphs (not necessarily acyclic) of minimal weight and given cardinality. Slight modifications of our methods will be applicable for this problem.

References

[1] E. Aarts and J.K. Lenstra. *Local Search in Combinatorial Optimization.* John Wiley & Sons, Inc., 1997.

[2] S. Arora. Polynomial time approximation algorithms for euclidean tsp and other geometric problems. In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science*, pages 2–11. IEEE Society Press, 1996.

[3] B. Awerbuch, Y. Azar, A. Blum, and S. Vempala. Improved approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, pages 277–283, 1995.

[4] A. Blum, P. Chalasani, and A. Vempala. A constant-factor approximation for the k-MST problem in the plane. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, pages 294–302, 1995.

[5] C. Blum. Optimality criteria nd local search methods for node-weighted k-cardinality tree and subgraph problems (in german). Master's thesis, University of Kaiserslautern, Department of Mathematics, 1998.

[6] F. Catanas. Heuristics for the p-minimal spanning tree problem. Technical report, Centre for Quantitative Finance, Imperial College, London, 1997.

[7] S.Y. Cheung and A. Kumar. Efficient quorumcast routing algorithms. In *Proceedings of INFOCOM 94*, pages 840–847. IEEE, 1994.

[8] T. Dudas, B. Klinz, and G.J. Woeginger. The computational complexity of the k-minimum spanning tree problem in graded matrices. Technical Report 106, Karl-Franzens Universität Graz and Technische Universität Graz, 1997.

[9] M. Ehrgott. Optimization problems in graphs under cardinality restrictions (in German). Master's thesis, University of Kaiserslautern, Department of Mathematics, 1992.

[10] M. Ehrgott and J. Freitag. K_TREE / K_SUBGRAPH: A program package for minimal weighted K-cardinality-trees and -subgraphs. *European Journal of Operational Research*, 93(1):224–225, 1996. Code available at http://www.mathematik.uni-kl.de/~wwwwi/WWWWI/ORSEP/contents.html.

[11] M. Ehrgott, J. Freitag, H.W. Hamacher, and F. Maffioli. Heuristics for the k-cardinality tree and subgraph problem. *Asia Pacific Journal of Operational Research*, 14(1):87–114, 1997.

[12] D. Eppstein. Faster geometric k-point mst approximation. Technical Report 13, University of California, Irvine, 1995.

[13] U. Faigle and W. Kern. Computational complexity of some maximum average weight problems with precedence constraints. *Operations Research*, 42(4):688–693, 1994.

[14] M. Fischetti, H.W. Hamacher, K. Joernsten, and F. Maffioli. Weighted k-cardinality trees: Complexity and polyhedral structure. *Networks*, 24:11–21, 1994.

[15] L.R. Foulds and H.W. Hamacher. A new integer programming approach to (restricted) facilities layout problems allowing flexible facility shapes. Technical Report 1992-3, University of Waikato, Department of Management Science, 1992.

[16] J. Freitag. Minimal k-cardinality trees (in German). Master's thesis, University of Kaiserslautern, Department of Mathematics, 1993.

[17] N. Garg. A 3-approximation for the minimum tree spanning k vertices. In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science*, pages 302–309. IEEE Society Press, 1996.

[18] N. Garg and D. Hochbaum. An $o(\log k)$ approximation algorithm for the k minimum spanning tree problem in the plane. *Algorithmica*, 18:111–121, 1997.

[19] F. Glover. Tabu search: part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[20] F. Glover. Tabu search: part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[21] H.W. Hamacher and K. Joernsten. Optimal relinquishment according to the norwegian petrol law: A combinatorial optimization approach. Technical Report no. 7/93, Norwegian School of Economics and Business Administration, Bergen, 1993.

[22] P. Hansen. The steepest ascent mildest descent heuristic for combinatorial programming. Talk presented at the Congress on Numerical Methods in Combinatorial Optimization, Capri.

[23] K. Joernsten and A. Lokketangen. Tabu search for weighted k-cardinality trees. *Asia Pacific Journal of Operational Research*, 14(2):9–26, 1997.

[24] F. Maffioli. Finding a best subtree of a tree. Technical Report 91.041, Politecnico di Milano, Dipartimento di Elettronica, 1991.

[25] J.S.B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple new method for the geometric k-mst problem. In *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 402–408. SIAM, 1996.

[26] H. Mühlenbein. Evolution in time and space – the parallel genetic algorithm. In G. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 316–337. Morgan-Kaufman, 1991.

[27] I.H. Osman and G. Laporte. Metaheuristics: A bibliography. In *Annals of Operations Research*, volume 63, pages 513–623. J.C. Baltzer AG, Science Publishers, 1996.

[28] A.B. Philpott and N.C. Wormald. On the optimal extraction of ore from an open-cast mine. 1997.

[29] S. Rajagopalan and V.V. Vazirani. Logarithmic approximation of minimum weight k trees. unpublished manuscript, 1995.

[30] R. Ravi, R. Sundaram, M.V. Marathe, D.J. Rosenkrantz, and S.S. Ravi. Spanning trees – short or small. *SIAM Journal on Discrete Mathematics*, 9(2):178–200, 1996.

[31] G.J. Woeginger. Computing maximum valued regions. *Acta Cybernetica*, 10(4):303–315, 1992.

[32] A. Zelikovsky and D. Lozevanu. Minimal and bounded trees. In *Tezele Cong. XVIII Acad. Romano-Americane, Kishinev*, pages 25–26, 1993.