Obtaining Optimal k-Cardinality Trees Fast

MARKUS CHIMANI
Dortmund University of Technology
MARIA KANDYBA
Dortmund University of Technology
IVANA LJUBIĆ
University of Vienna
and
PETRA MUTZEL
Dortmund University of Technology

Given an undirected graph $G = (V, E)$ with edge weights and a positive integer number k, the k-Cardinality Tree problem consists of finding a subtree T of G with exactly k edges and the minimum possible weight. Many algorithms have been proposed to solve this NP-hard problem, resulting in mainly heuristic and metaheuristic approaches.

In this paper we present an exact ILP-based algorithm using directed cuts. We mathematically compare the strength of our formulation to the previously known ILP formulations of this problem, and show the advantages of our approach. Afterwards we give an extensive study on the algorithm's practical performance compared to the state-of-the-art metaheuristics.

In contrast to the widespread assumption that such a problem cannot be efficiently tackled by exact algorithms for medium and large graphs (between 200 and 5000 nodes), our results show that our algorithm not only has the advantage of proving the optimality of the computed solution, but also often outperforms the metaheuristic approaches in terms of running time.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Computations on discrete structures*; *Routing and layout*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*Graph algorithms*; *Network problems*; G.4 [**Mathematical Software**]: Efficiency

General Terms: Algorithms, Design, Experimentation, Performance, Theory

Additional Key Words and Phrases: Exact algorithm, k-Cardinality tree, comparison with metaheuristics, Branch & Cut, (prize collecting) Steiner tree

A preliminary version of this paper appeared in the Proceedings of the SIAM Workshop on Algorithm Engineering & Experiments 2008 (ALENEX08).
Authors' addresses: M. Chimani, M. Kandyba, P. Mutzel, Chair XI Algorithm Engineering, Dortmund University of Technology, Otto-Hahn-Str. 14, 44227 Dortmund, Germany. I. Ljubić, Dep. of Statistics and Decision Support Systems, University of Vienna, Brünnerstr. 72, 1210 Vienna, Austria. {markus.chimani,maria.kandyba,petra.mutzel}@tu-dortmund.de, ivana.ljubic@univie.ac.at
Maria Kandyba was supported by the German Research Foundation (DFG) through the Collaborative Research Center "Computational Intelligence" (SFB 531).
Ivana Ljubić was supported by the Hertha-Firnberg Fellowship of the Austrian Science Foundation (FWF).

1. INTRODUCTION

We consider the *k-Cardinality Tree problem* (KCT). Given an undirected graph $G = (V, E)$, an edge weight function $w : E \to \mathbb{R}$, and a positive integer number k, find a subgraph T of G that is a minimum weight tree with exactly k edges. The problem is also known as the *k-Minimum Spanning Tree problem* (k-MST), whereby the parameter k usually specifies the number of selected nodes, which in a tree is exactly one more than the number of edges. In most of the literature, the edge weights are restricted to be non-negative; in our paper we do not require this assumption. Even for the restricted case with edge weights $w(e) \in \{1, 2, 3\}$ for all edges e, the problem has been shown to be NP-hard [Ravi et al. 1996].

The KCT has been extensively studied in literature as it has various applications, e.g., in oil-field leasing, facility layout, open pit mining, matrix decomposition, quorum-cast routing, telecommunications, etc. [Blum and Ehrgott 2003]. The considered approaches range from heuristics, to metaheuristics, approximation algorithms and exact formulations.

The first exact approach was presented by Fischetti et al. [1994], by formulating an integer linear program (ILP) based on generalized subtour elimination constraints (GSEC). This formulation was implemented by Ehrgott and Freitag [1996] using a Branch-and-Cut approach. The resulting algorithm was only able to solve graphs with up to 30 nodes, which may be mainly due to the comparably weak computers in 1996. The ideas of such formulations have been picked up by the approximation algorithm community [Arora and Karakostas 2000; Blum et al. 1996; Garg 1996; 2005]. The central idea thereby is the ILP-based primal-dual scheme, which was proposed by Goemans and Williamson [1995] for the prize-collecting Steiner tree problem.

A large amount of research was devoted to the development of heuristic [Ehrgott et al. 1997; Blum 2007] and, in particular, metaheuristic methods [Joernsten and Lokketangen 1997; Blum and Ehrgott 2003; Urošević et al. 2004; Bui and Sundarraj 2004; Blum and Blesa 2005a; 2005b; Blum 2006]. See, e.g., [Blum 2006] for an extensive comparison of the latter approaches. An often used argument for metaheuristic approaches is based on the performance of the first exact approach reported above. The key argument is that exact methods for this NP-hard problem would require too much computation time and could only be applied to very small graphs [Blum and Ehrgott 2003; Brimberg et al. 2006].

In this paper we show that the traditional argument for metaheuristics over exact algorithms is deceptive on this and related problems. We propose a novel exact ILP-based algorithm which can indeed be used to solve all known benchmark instances of KCTLIB [Blum and Blesa 2003]—containing graphs of up to 5000 nodes—to provable optimality. Furthermore, our algorithm often, in particular on mostly all graphs with up to 1000 nodes, is faster than the state-of-the-art metaheuristic approaches that can neither guarantee nor assess the quality of their solution.

To achieve these results, we present an exact Branch-and-Cut algorithm for the KCT. Therefore we transform the problem into a similar directed and rooted problem called *k-Cardinality Arborescence problem* (KCA), and formulate an ILP for the latter, see Section 2. In the section thereafter, we provide polyhedral comparisons to other formulations existing for the KCT problem; most importantly this

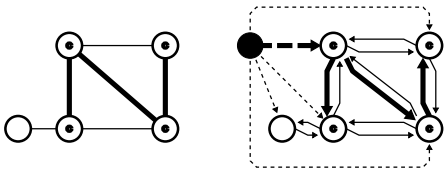


Fig. 1. An undirected KCT instance (left), and its directed KCA counterpart (right). A possible solution for $k = 3$ is denoted by bold edges/arcs.

is the GSEC formulation mentioned above, as well as formulations based on undirected cuts and multi-commodity flow, respectively. In Section 4, we summarize other problems similar to the KCT—in particular the NKCT problem, the node weighted variant of KCT—and show how they can also be transformed into a KCA problem and therefore modeled using our ILP formulation.

In Section 5, we describe the resulting Branch-and-Cut algorithm in order to deal with the exponential ILP size. A large part of the paper is then devoted to an extensive experimental study in Section 6, where we analyze the performance of our algorithm and compare it with the state-of-the-art metaheuristics for the KCT. We conclude with remarks regarding the practical applicability of our approach to other problem instances or variants.

2. DIRECTED CUT APPROACH

2.1 Transformation into the k-Cardinality Arborescence Problem

Let $D = (V_D, A_D)$ be a directed graph with a distinguished root vertex $r \in V_D$ and arc costs c_a for all arcs $a \in A_D$. The k-Cardinality Arborescence problem (KCA) consists of finding a weight minimum rooted tree T_D with k arcs which is directed from the root outwards. More formally, T_D has to satisfy the following properties:

(P1) T_D contains exactly k arcs,

(P2) T_D contains exactly $k + 1$ nodes, and

(P3) for all $v \in V(T_D) \setminus \{r\}$, there exists a directed path $r \to v$ in T_D.

We transform any given KCT instance $(G = (V, E), w, k)$ into a corresponding KCA instance $(G_r, r, c, k + 1)$ as follows, cf. Figure 1: We replace each edge $\{i, j\}$ of G by two arcs (i, j) and (j, i), introduce an artificial root vertex r and connect r to every node in V. Hence we obtain a digraph $G_r = (V \cup \{r\}, A \cup A_r)$ with $A = \{(i, j), (j, i) \mid \{i, j\} \in E\}$ and $A_r = \{(r, j) \mid j \in V\}$. For each arc $a = (i, j)$ we define the cost function $c(a) := 0$ if $i = r$, and $c(a) := w(\{i, j\})$ otherwise.

To be able to interpret each feasible solution T_{G_r} of this resulting KCA instance as a solution of the original KCT instance, we impose an additional constraint

(P4) T_{G_r} contains only a single arc of A_r.

If this property is satisfied, it is easy to see that a feasible KCT solution with the same objective value can be obtained by removing r from T_{G_r} and interpreting the directed arcs as undirected edges.

2.2 ILP for the KCA

To model the KCA as an ILP, we introduce two sets of binary variables:

$$x_a, y_v \in \{0,1\} \quad \forall a \in A \cup A_r, \forall v \in V.$$

The variables are 1, if the corresponding arc or vertex is in the solution and 0 otherwise.

Let $S \subseteq V$. The sets $E(S)$ and $A(S)$ are the edges and arcs of the subgraphs of G and G_r, respectively, induced by S. Furthermore, we denote by $\delta^-(S) = \{(i,j) \in A \cup A_r \mid i \in V \setminus S, j \in S\}$ the ingoing arcs of a set S. We define the shorthands $x(B) := \sum_{b \in B} x_b$ with $B \subseteq A \cup A_r$, and $y(W) := \sum_{w \in W} y_w$ with $W \subseteq V \cup \{r\}$. This allows us to give the following ILP formulation:

$$\begin{aligned}
\text{DCut:} \quad & \min \textstyle\sum_{a \in A} c(a) \cdot x_a && && (1)\\
\text{s.t.} \quad & x(A) = k && && (2)\\
& y(V) = k+1 && && (3)\\
& x(A_r) = 1 && && (4)\\
& x(\delta^-(S)) \geq y_v && \forall S \subseteq V, \forall v \in S && (5)\\
& x_a, y_v \in \{0,1\} && \forall a \in A \cup A_r, \forall v \in V. && (6)
\end{aligned}$$

The k-cardinality requirements (P1) and (P2) are modeled by (2) and (3), respectively. The root-out-degree constraint (4) ensures property (P4). The *dcut-constraints* (5) guarantee property (P3) via directed cuts.

Although this formulation is sound and practically applicable, we will reformulate the ILP based on the following observation.

Observation 2.1. *Any subgraph T_D satisfying (P1)–(P3) also satisfies: For all $v \in V(T_D) \setminus \{r\}$, v has in-degree 1 in T_D.*

Hence, considering the feasible integer solutions, we can replace the node cardinality constraint (3) by in-degree constraints

$$x(\delta^-(v)) = y_v \quad \forall v \in V \tag{7}$$

and obtain the ILP

$$\min \left\{ \sum_{a \in A} c(a) \cdot x_a \text{ subject to (2), (4)–(7)} \right\}. \tag{8}$$

While the original formulation is more compact, we will see in Section 5 that the in-degree constraints have certain advantages in practice. The most important property of this reformulation is that it retains the strength of the original formulation. From the polyhedral point of view, we can measure this strength by considering the LP-relaxation of the ILP, i.e., we replace the integer constraints (6) by

$$0 \leq x_a, y_v \leq 1 \quad \forall a \in A \cup A_r, \forall v \in V, \tag{9}$$

and thereby allow fractional solutions. We can show that in this context it is irrelevant whether we model (P2) directly via a cardinality constraint or indirectly with multiple in-degree constraints.

LEMMA 2.2. *By replacing the node cardinality constraint* (3) *by the in-degree constraints* (7) *we obtain an equivalent ILP and an equivalent LP-relaxation.*

PROOF. We show this equivalence by generating the constraints from each other. We have (7)⇒(3) as:

$$y(V) = \sum_{v \in V} y_v \overset{(7)}{=} \sum_{v \in V} x(\delta^-(v)) = x(A) + x(A_r) \overset{(2),(4)}{=} k + 1.$$

And show (3)⇒(7) by:

$$\begin{aligned} k+1 \overset{(3)}{=} y(V) = \textstyle\sum_{v \in V} y_v \overset{(5)}{\leq} \sum_{v \in V} x(\delta^-(v)) = x(A) + x(A_r) \overset{(2),(4)}{=} k+1 \\ \Longrightarrow y_v = x(\delta^-(v)) \quad \forall v \in V. \end{aligned}$$

□

3. POLYHEDRAL COMPARISONS

We investigate the polyhedral properties of our new formulation by comparing it to the other three available ILP formulations for the KCT problem. In the following, let $\mathcal{P}_D$ be the polyhedron corresponding to the DCUT LP-relaxation, i.e., $\mathcal{P}_D$ contains all points feasible for DCUT disregarding the integer properties of the variables:

$$\begin{aligned} \mathcal{P}_D := \{ \ (x,y) \in \mathbb{R}^{|A \cup A_r| + |V|} \mid 0 \leq x_e, y_v \leq 1 \\ \text{and } (x,y) \text{ satisfies (2)–(5)} \qquad \}. \end{aligned}$$

Hence, the objective value of the optimal fractional solution within $\mathcal{P}_D$ is a lower bound for the optimal integer solution. We can consider a formulation *stronger* than another, if its LP-solution gives a tighter lower bound than the other. This measure is particularly useful in the context of Branch-and-Cut algorithms, see Section 5: The final solution at the root node of the Branch-and-Bound tree, i.e., before we have to resort to branching, is exactly an optimal fractional solution of the LP-relaxation. In particular, in our application we can obtain this optimal fractional solution in polynomial time using cutting plane methods, cf. Section 5.

3.1 Generalized Subtour Elimination (GSEC)

The only reported ILP implementation regarding the KCT problem is due to Ehrgott and Freitag [1996], which uses the formulation by Fischetti et al. [1994]. This formulation is defined on the original, undirected graph and based on generalized subtour elimination constraints. We rephrase this approach to match our notation and show that both GSEC and DCUT are equivalent from the polyhedral point of view.

In order to distinguish between undirected edges and directed arcs we introduce the binary variables $z_e \in \{0,1\}$ for every edge $e \in E$, which are 1 if e is contained in the solution tree and 0 otherwise. For representing the selection of the nodes we use the y-variables as in the previous section. The constraints (13) are called the

gsec-constraints.

$$\begin{aligned}
\text{GSEC:} \quad & \min \textstyle\sum_{e\in E} c(e)\cdot z_e && && (10)\\
\text{s.t.} \quad & z(E) &= k && & (11)\\
& y(V) &= k+1 && & (12)\\
& z(E(S)) &\leq y(S\setminus\{t\}) && \forall S\subseteq V, |S|\geq 2, \forall t\in S & (13)\\
& z_e, y_v &\in \{0,1\} && \forall e\in E, \forall v\in V. & (14)
\end{aligned}$$

Let $\mathcal{P}_G$ be the polyhedron corresponding to the GSEC LP-relaxation, i.e.,

$$\mathcal{P}_G := \{\ (z,y)\in\mathbb{R}^{|E|+|V|} \mid 0\leq z_e, y_v\leq 1 \text{ and } (z,y) \text{ satisfies (11)–(13)}\ \}.$$

THEOREM 3.1. *The* GSEC *and the* DCUT *formulations have equally strong LP-relaxations, i.e.,*

$$\mathcal{P}_G = \text{proj}_z(\mathcal{P}_D),$$

where $\text{proj}_z(\mathcal{P}_D)$ *is the projection of* $\mathcal{P}_D$ *onto the* (z,y) *variable space with* $z_{\{i,j\}} = x_{(i,j)} + x_{(j,i)}$ *for all* $\{i,j\}\in E$. *The* y *variables are thereby projected using the identity function, i.e., their values remain identical.*

PROOF. We prove equality by showing mutual inclusion:

$\text{proj}_z(\mathcal{P}_D)\subseteq\mathcal{P}_G$: Any $(\bar z,\bar y)\in\text{proj}_z(\mathcal{P}_D)$ satisfies (11) by definition, and (12) by (7) and Lemma 2.2. Let $\bar x$ be the vector from which we projected the vector $\bar z$, and consider some $S\subseteq V$ with $|S|\geq 2$ and some vertex $t\in S$. We show that $(\bar z,\bar y)$ also satisfies the corresponding gsec-constraint (13):

$$\begin{aligned}
\bar z(E(S)) = \bar x(A(S)) &= \textstyle\sum_{v\in S}\bar x(\delta^-(v)) - \bar x(\delta^-(S))\\
&\overset{(7)}{=} \bar y(S) - \bar x(\delta^-(S)) \overset{(5)}{\leq} \bar y(S) - \bar y_t.
\end{aligned}$$

$\mathcal{P}_G\subseteq\text{proj}_z(\mathcal{P}_D)$: Consider any $(\bar z,\bar y)\in\mathcal{P}_G$ and a set

$$X := \{\ \ x\in\mathbb{R}_{\geq 0}^{|A\cup A_r|} \mid x \text{ satisfies (4)} \text{ and } x_{ij}+x_{ji} = \bar z_{\{ij\}}\ \forall (i,j)\in A\ \}.$$

Every such projective vector $\bar x\in X$ clearly satisfies (2). In order to generate the dcut-inequalities (5) for the corresponding $(\bar x,\bar y)$, it is sufficient to show that we can always find an $\hat x\in X$, which together with $\bar y$ satisfies the indegree-constraints (7). Since then, for any $S\subseteq V$ and $t\in S$:

$$\begin{aligned}
\hat x(\delta^-(S)) &= \textstyle\sum_{v\in S}\hat x(\delta^-(v)) - \hat x(A(S))\\
&\overset{(7)}{=} \bar y(S) - \bar z(E(S)) \overset{(13)}{\geq} \bar y_t.
\end{aligned}$$

We show the existence of such an $\hat x$ using a proof technique related to Goemans and Myung [1993, proof of Claim 2], where it was used for the Steiner tree problem.

An $\hat x\in X$ satisfying (7) can be interpreted as the set of feasible flows in a bipartite transportation network (N,L), with $N := (E\cup\{r\})\cup V$. For each

undirected edge $e = (u, w) \in E$ in G, our network contains exactly two outgoing arcs $(e, u), (e, w) \in L$. Furthermore, L contains all arcs of A_r. For all nodes $e \in E$ in N we define a supply $s(e) := \bar{z}_e$; for the root r we set $s(r) := 1$. For all nodes $v \in V$ in N we define a demand $d(v) := \bar{y}_v$.
Finding a feasible flow for this network can be viewed as a capacitated transportation problem on a complete bipartite network with capacities either zero (if the corresponding edge does not exist in L) or infinity. Note that in our network the sum of all supplies is equal to the sum of all demands, due to (11) and (12). Hence, each feasible flow in such a network will lead to a feasible $\hat{x} \in X$. Such a flow exists if and only if for every set $M \subseteq N$ without arcs leaving M (i.e., $\delta^+_{(N,L)}(M) = \emptyset$) the condition

$$s(M) \leq d(M) \tag{15}$$

is satisfied, where $s(M)$ and $d(M)$ are the total supply and the total demand in M, respectively, cf. [Gale 1957; Goemans and Myung 1993]. In order to show that this condition holds for (N, L), we distinguish between two cases; let $U := E \cap M$:

$r \in M$: Since r has an outgoing arc for every $v \in V$ and $\delta^+_{(N,L)}(M) = \emptyset$, we have $V \subset M$. Condition (15) is satisfied, since $s(r) = 1$ and therefore:

$$\begin{aligned} s(M) &= s(r) + \bar{z}(U) \leq s(r) + \bar{z}(E) \\ &= \bar{z}(E) + 1 \overset{(11),(12)}{=} \bar{y}(V) = d(M). \end{aligned}$$

$r \notin M$: Let $S := V \cap M$. We then have $U \subseteq E(S)$. If $|S| \leq 1$ we have $U = \emptyset$ and therefore (15) is automatically satisfied. For $|S| \geq 2$, the condition is also satisfied, since for every $t \in S$ we have:

$$\begin{aligned} s(M) = \bar{z}(U) \leq \bar{z}(E(S)) &\overset{(13)}{\leq} \bar{y}(S) - \bar{y}_t \\ &\leq \bar{y}(S) = d(M). \end{aligned}$$

□

Even though both DCut and Gsec are polytope-wise equivalent, DCut offers advantages in practice, as we will discuss in Section 5.

3.2 Undirected Cuts (UCut)

Garg [1996] considers approximation algorithms for the KCT. To this ends, he presents an ILP based on undirected cuts, whose LP-relaxation he subsequently uses for lower bounds. We will call this formulation UCut in the following. The formulation also uses a root node r', but instead of an artifical root as we use for DCut, it selects the node among the original nodes. Since it cannot be guaranteed that r' is contained in the optimal solution, he considers $|V|$ ILPs, one for each possible choice of the root node r'. We rephrase the original formulation, adapted to our notation. The z variables are used analogously as for the Gsec formulation.

$$\begin{aligned} \text{UCut:} \quad & \min \textstyle\sum_{e \in E} c(e) \cdot z_e && && (16) \\ \text{s.t.} \quad & y(V) = k + 1 && && (17) \\ & z(\delta(S)) \geq y_v && \forall S \subseteq V \setminus \{r'\}, \forall v \in S && (18) \\ & z_e, y_v \in \{0, 1\} && \forall e \in E, \forall v \in V. && (19) \end{aligned}$$

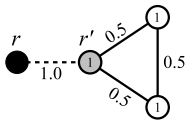


Fig. 2. Proof of Theorem 3.2. The shown solution is feasible for UCut* and UCut, with and without the artificial root r, respectively. There is no fractional orientation which would be feasible for DCut.

Here, $\delta(S)$ denotes the set of all edges for which exactly one of the incident nodes is in S. Garg assumes non-negativity for the edge weights which is why he does not have to restrict the number of selected edges.

In contrast to the formulation by Garg, we can augment G with an artificial root node which is connected to all other nodes, instead of selecting any original node as the root node. By ensuring that only one of these new edges is selected analogously to (4), we obtain a formulation which directly solves the KCT problem via undirected cuts, instead of requiring linearly many ILPs. We denote this formulation as UCut*.

Let $\mathcal{P}_U^*$ be the polyhedron of the LP-relaxation of UCut*, and let $\mathcal{P}_U$ be the union of all LP-relaxation polyhedra considered by UCut for any choice of the root r'. Using the projection defined in Theorem 3.1 we can show the following theorem.

Theorem 3.2. DCut *is strictly stronger than* UCut *and* UCut*, *i.e.,*

$$\text{proj}_z(\mathcal{P}_D) \subset \mathcal{P}_U, \text{ and } \text{proj}_z(\mathcal{P}_D) \subset \mathcal{P}_U^*.$$

Proof. Clearly, each feasible point in $\mathcal{P}_D$ is feasible in $\mathcal{P}_U$ and $\mathcal{P}_U^*$ using the projection proj_z. On the other hand, we use a traditional argument to identify fractional solutions feasible for the undirected formulations, but infeasible for DCut, cf. Figure 2. Assume a complete graph on 3 nodes, where each vertex variable is set to 1, and each edge variable is set to 0.5. For the formulation requiring an artificial node, connect such an additional root vertex to this triangle via a single edge whose variable is set to 1.

This structure forms a feasible solution for UCut*. In case of UCut, choose any node on the triangle as r' and we have a feasible solution (ignoring the artificial root). However, for DCut there is no possible x-variable setting inducing the described z-variable values such that the in-degree constraints are satisfied for both two nodes which are not adjacent to the artificial root. □

Overall, we can deduce that DCut is a better choice for an exact approach than UCut, since its improved strength will in general lead to tighter bounds and fewer branches in Branch-and-Cut algorithms.

3.3 Multi-Commodity Flow (Mcf)

Another traditional approach for similar network design problems are multi-commodity flow formulations. We can straight-forwardly formulate an ILP (Mcf) based on this concept for the KCA, as it was done by Ljubić [2004] for the prize-collecting Steiner tree problem (PCST), and augment it with cardinality inequalities. In such a formulation, we have one commodity t_v for each node in $v \in V$, and we send y_v units of t_v from the artificial root r to v.

Analogously to the proof in [Ljubić 2004], which shows the equivalence of DCut and Mcf for the PCST, we obtain the following theorem.

THEOREM 3.3. *The LP-relaxation of* MCF *for the KCA is equivalent to* GSEC *and* DCUT.

The MCF formulation requires only a polynomial number of variables and constraints. However, the sheer number of variables becomes a practical drawback of this approach. In addition to the x and y variables, we require $|V| \cdot |A|$ variables to model the flow. As we know from similar problems [Ljubić et al. 2006; Chimani et al. 2007; **?**], this leads to poor performance of multi-commodity flows in practice, compared to directed-cut based approaches which allow efficient separation of their exponentially many constraints, cf. Section 5.2.

4. APPLICABILITY TO RELATED PROBLEMS

The main focus of this paper is on the above KCT problem. However, the KCT is only the most prominent within a class of similar or related NP-hard problems that can be found in the literature. Interestingly, we can formulate most of them as a KCA problem or slight variations thereof, which allows us to use our DCUT approach for these problems as well.

4.1 Node-weighted/All-weighted k-Cardinality Tree problem

The Node-weighted k-Cardinality Tree problem (NKCT) is defined analogously to the KCT but its weight function $w' : V \to \mathbb{R}$ uses the nodes as its basic set, instead of the edges (see, e.g., [Brimberg et al. 2006] for the list of references). We can also consider the general All-weighted k-Cardinality Tree problem (AKCT), where a weight-function w for the edges, and a weight-function w' for the nodes are given.

As Segev [1987] did for Steiner trees, we can use Observation 2.1 to eliminate the node variables in the objective function. Recall that any KCA solution is a tree directed from the root outwards. Each vertex in the solution (except for the root) has in-degree 1. Disregarding the root node, this allows us to establish a one-to-one correspondence between each selected arc and its target node. Whenever we select an arc, we know that we have to pay the cost of its target node v, and we will never pay the node weight multiple times since we will only select a single arc having v as its target node. This observation allows us to precompute the node weights of the NKCT and AKCT problems into the arc weights of the KCA.

By generalizing the transformation for the KCT problem we can therefore transform any given NKCT instance $(G = (V, E), w', k)$ or AKCT instance $(G = (V, E), w, w', k)$ into a corresponding KCA instance $(G_r, r, c, k + 1)$ as follows. As before let $G_r = (V \cup \{r\}, A \cup A_r)$. For all $(i, j) \in A \cup A_r$ we have $c((i, j)) := w'(j)$ for the NKCT, and $c((i, j)) := w(\{i, j\}) + w'(j)$ or $c((i, j)) := w'(j)$ if $i = r$, for the AKCT. By this transformation we implicitly choose $w'(r) = 0$ which is correct since the root node is artificial and should not be part of the objective function.

4.2 k-STP and k-PCSTP

The traditional Steiner tree problem (STP) consists of finding a weight-minimum tree in an edge weighted (non-negative weights) graph that spans a set of *terminal nodes*, i.e., a certain subset of the graph's nodes. Chudak et al. [2001] considered the k-Cardinality Steiner Tree Problem (k-STP) as a hybrid of the STP and the KCT, i.e., we ask for the weight-minimum tree spanning k terminal nodes. In

the paper, the authors present an ILP based on undirected graphs and use it for deriving an approximation algorithm. However, the k-STP is only considered from this theoretical point of view. To our knowledge, there are no widely available benchmark instances nor any experimentally comparable algorithmic approaches for this problem class.

Analogously to the k-STP, we can consider the k-Cardinality Prize-Collecting Steiner Tree Problem (k-PCSTP). The traditional prize-collecting Steiner tree problem differs from the STP in that not all terminal nodes have to be contained in the solution tree, but each terminal node has a *prize* (or *profit*) which is subtracted from the tree's cost if the node is selected. The k-PCSTP is then the problem of finding a prize-collecting Steiner tree with the side-condition that k terminal nodes have to be selected. The k-STP problem is then the special case of the k-PCSTP where all prizes of the terminal nodes are 0. Note that the general k-PCSTP does not require the edge weights nor the node prizes to be non-negative.

The ideas for the KCA can be directly used to give a directed-cut formulation for the k-PCSTP and its subtypes. We can integrate the node prizes into the arc costs as discussed for the NKCT above, drop the cardinality requirement on the edges, apply a node cardinality requirement analogous to (3) only to the terminal nodes and add all in-degree constraints (7), in order to guarantee that the solution forms a tree.

5. BRANCH-AND-CUT ALGORITHM

Based on our DCut formulation, we developed and implemented a Branch-and-Cut algorithm. For a general description of the Branch-and-Cut scheme see, e.g., [Wolsey 1998]. In such an algorithm, we start with an initial *partial LP*, i.e., the ILP without the integrality properties and only considering a certain subset of all constraints. We solve the partial LP in order to obtain a *current fractional solution*. A *separation routine* then tries to identify constraints of the full constraint set of the ILP which the current fractional solution violates. We then add these constraints to our partial LP and reiterate these steps. If at some point we cannot find any violated constraints, we have to resort to *branching*, i.e., we generate two disjoint subproblems, e.g., by fixing a variable to 0 or 1. By using the fractional solutions as lower bounds, and some heuristic solution as an upper bound, we can prune irrelevant subproblems. In every node of the resulting Branch-and-Bound tree, we apply the separation strategy again.

A particularly interesting theorem central to LP theory by Grötschel, Lovász, and Schrijver (cf., e.g, [Wolsey 1998]) shows the equivalence of optimization and separation, i.e., if we can solve the separation problem in polynomial time, we can also solve the underlying LP-problem in polynomial time. We will see that this is indeed the case for the exponentially many dcut-constraints. Hence we can obtain the optimal fractional solution of the LP-relaxation in polynomial time at the root node of the Branch-and-Bound tree.

Ehrgott and Freitag [1996] developed a Branch-and-Cut algorithm based on the Gsec formulation. Note that the dcut-constraints are sparser than the gsec-constraints, which usually leads to a faster optimization in practice. This conjecture was experimentally confirmed, e.g., by Ljubić et al. [2006] for the related

prize-collecting Steiner tree problem, where a directed-cut based formulation was compared to a GSEC formulation. The former was both faster in overall running time and required 1–2 orders of magnitude fewer iterations. Hence we can expect our DCUT approach to have advantages over GSEC in practice. In Section 5.2 we will discuss the formal differences in the performances between the DCUT and the GSEC separation algorithms.

5.1 Initialization

Our algorithm starts with the root-out-degree constraint (4), the edge cardinality constraint (2), and all in-degree constraints (7). We prefer the in-degree constraints over the node cardinality constraint (3), as they strengthen the initial partial LP. As the proof of Lemma 2.2 shows, all in-degree constraints, together with the root-out-degree and the edge cardinality constraint already induce the node cardinality constraint, and hence it is not necessary to consider the latter. On the other hand, in order to generate the in-degree constraints, we would require multiple dcut-constraints, which are not available in the initial partial LP.

For the same reason, we add the orientation-constraints

$$x_{ij} + x_{ji} \leq y_i \quad \forall i \in V, \forall \{i,j\} \in E \tag{20}$$

to our initial ILP. Intuitively, these constraints ensure a unique orientation for each edge, and require for each selected arc that both incident nodes are selected as well. These constraints do not actually strengthen the DCUT formulation as they represent the gsec-constraints for all two-element sets $S = \{i,j\} \subset V$: from the proof of Theorem 3.1, we know that these inequalities can be generated with the help of (7) and (5). Nonetheless the addition of the constraints (20) to the initial partial LP speeds up the algorithm significantly, as they do not have to be separated explicitly by the Branch-and-Cut algorithm. This was first observed by Ljubić [2004] for the PCST and also confirmed by our own experiments.

We also tried *asymmetry constraints* [Ljubić 2004] to reduce the search space by excluding symmetric solutions:

$$x_{rj} \leq 1 - y_i \quad \forall i,j \in V, i < j. \tag{21}$$

They assure that for each KCA solution, the vertex adjacent to the root is the one with the smallest possible index. Anyhow, we will see in our experiments that the quadratic number of these constraints becomes a hindrance for large graphs and/or small k in practice.

5.2 Separation

The dcut-constraints (5) can be separated in polynomial time via the traditional maximum-flow separation scheme: We consider G_r as a network and interpret the x-values of the current fractional LP solution as arc capacities. We compute the maximum-flow in G_r from r to each $v \in V$ using the implementation of [**?**]. If the flow is less than y_v, we extract one or more of the induced minimum (r,v)-cuts and add the corresponding constraints to our model. In order to obtain multiple cuts with a single separation step we also use nested- and back-cuts [Koch and Martin 1998; Ljubić et al. 2006]. Indeed, using these additional cuts significantly speeds up the computation.

Recall that in a general separation procedure we search for the most violated inequality of the current LP-relaxation. In order to find the most violated inequality of the DCut formulation, or to show that no such inequality exists, we construct the flow network only once and perform at most $|V|$ maximum-flow calculations on it. This is a main reason why the DCut formulation performs better than Gsec in practice: a single separation step for Gsec requires $2|V| - 2$ maximum-flow calculations, as already shown by Fischetti et al. [1994]. Furthermore, the corresponding flow network is not static over all those calculations, but has to be adapted prior to each call of the maximum-flow algorithm.

Our test sets, as described in Section 6, also contain grid graphs. In such graphs, it is easy to detect and enumerate all 4-cycles by embedding the grids into the plane and traversing all faces except for the single large one. In any planar graph there is only a linear number of faces. For grids with n nodes we even know that there are at most $(\sqrt{n}-1)^2 = (n+1-2\sqrt{n})$ 4-cycles. Note that due to our transformation, all 4-cycles are bidirected. Let $\mathcal{C}_4$ be the set of all bidirected 4-cycles. A cycle $C \in \mathcal{C}_4$ then consists of 8 arcs and $V[C]$ gives the vertices on C. By enumerating these cycles, we obtain a separation routine for gsec-constraints on them:

$$\sum_{a \in C} x_a \leq \sum_{i \in V[C] \setminus \{v\}} y_i \quad \forall C \in \mathcal{C}_4, \forall v \in V[C]. \tag{22}$$

5.3 Upper Bounds and Proving Optimality

In the last decade, several heuristics and metaheuristics have been developed for the KCT problem. Traditional Branch-and-Cut algorithms allow to use such algorithms as *primal heuristics*, giving upper bounds that the Branch-and-Cut algorithm can use for bounding purposes when branching. The use of such heuristics is two-fold: (a) They can be used as start-heuristics, giving a good initial upper bound before starting the actual Branch-and-Cut algorithm, and (b) they can be run multiple times during the exact algorithm, using the current fractional solutions as an additional input, or *hint*, in order to generate new and tighter upper bounds on the fly.

Let h be a primal bound obtained by such a heuristic. We can add this bound to our LP as

$$\sum_{a \in A} c(a) \cdot x_a \leq h - \Delta.$$

Here, $\Delta := \min\{c(a) - c(b) \mid c(a) > c(b),\ a, b \in A\}$ denotes the minimal difference between any two cost values. If the resulting ILP is found to be infeasible, we have a proof that h was optimal, i.e., the heuristic solution was optimal.

As our experiments reveal, our algorithm is already very successful without the use of any primal heuristic. Hence we compared our heuristic-less Branch-and-Cut algorithm (DC^-) with one using a *perfect heuristic*: a (hypothetical) algorithm that requires no running time and gives the optimal solution. We can simulate such a perfect heuristic by using the optimal solution obtained by a prior run of DC^-. We can then measure how long the algorithm takes to discover the infeasibility of the ILP. We call this algorithm variant DC^+. If the runtime performance of DC^- and DC^+ are similar, we can conclude that using any heuristic for bounding is not necessary.

6. EXPERIMENTAL RESULTS

We implemented our algorithm in C++ using CPLEX 9.0 and LEDA 5.0.1. The experiments were performed on 2.33 GHz Intel Xeon. Although the machine offers two processing units, each process was restricted to 1 CPU and 2 GB RAM. We tested our algorithm on all instances of the KCTLIB [Blum and Blesa 2003] which consists of the following benchmark sets:

(BX) The set by Blesa and Xhafa [2000] contains 35 4-regular graphs with 25–1000 nodes. The value of k is fixed to 20. The results of Blum and Blesa [2005b] have already shown that these instances are easy, which was confirmed by our experiments. Our algorithm needed on average 0.85 seconds per instance to solve them to optimality, the median was 0.08 seconds.

(BB) The set by Blum and Blesa [2005b] is divided into four subsets of dense, sparse, grid and 4-regular graphs, respectively, with different sizes of up to 2500 nodes. We use the notation $k_{\text{rel}} = X\%$ to denote that k is chosen to be $X\%$ of $n = |V|$. E.g., for a graph with 2500 nodes, $k_{\text{rel}} = 10\%$ means that we choose $k = 250$. Each instance of the benchmark set has to be solved for $k_{\text{rel}} = \{10\%, \ldots, 90\%\}$[1], and additionally for $k = 2$ and $k = n - 2$. The most successful known metaheuristics for (BB) are the hybrid evolutionary algorithm (HyEA) [Blum 2006] and the ant colony optimization algorithm (ACO) [Bui and Sundarraj 2004].

(UBM) The set by Urošević et al. [2004] consists of large 20-regular graphs with 500–5000 nodes which were generated randomly. The values for k are defined as for (BB) by using $k_{\text{rel}} = \{10\%, \ldots, 50\%\}$. Urošević et al. [2004] presented a variable neighborhood decomposition search (VNDS), which is still the best known metaheuristic for this benchmark set. However, there are no published results on the behaviour of VNDS on any other test instances.

Remark 6.1. The choices $k = 2$ and $k = n - 2$ for (BB) are rather insignificant for the analysis of general KCT algorithms. We can solve the former case optimally in $O(|V||E|)$ by building BFS trees of depth 2 for all vertices, thereby enumerating all connected edge pairs. The latter can be solved to optimality by considering the graph G^v—the graph G without the node v—for all $v \in V$. For each such graph we compute a minimum spanning tree and choose the minimum among them as the solution. Hence we require $O(|E||V| \log |V|)$ time.

Our computational experiments on (UBM) show that all instances can be solved to provable optimality. Except for 20 out of the 350 instances, all of them can even be solved in under two hours. The longer running instances are some large ones with 4000–5000 nodes which require 40–50% of edges to be in the solution.

Table I gives the average running times of our algorithm, as well as the the running times of the VNDS metaheuristic by Urošević et al. [2004]. Please note that even though the latter times are larger, we can expect them to be substantially smaller on recent machines, as they where achieved on a Pentium II with 450 MHz. Unfortunately, this machine was too old to be considered in the current SPEC performance evaluation tests [Spec 2008], so we cannot fairly compare these running

[1]For the grid instances, the values k_{rel} differ slightly.

Table I. Average running times of DC$^-$ for (UBM) in seconds. The line with * considers $k_{\text{rel}} = 50\%$ for $|V| \leq 2000$ and is achieved by using the asymmetry constraints (21). The times in small font after the slashes are the times of VNDS reported in [Urošević et al. 2004]. We also give the average gap between the optimum and the best solutions obtained by VNDS. Note that the times for VNDS were achieved on a much slower machine (cf. text).

nds	**500**	**1000**	**1500**	**2000**	**3000**	**4000**	**5000**
k_{rel}							
10%	2.8/15.3	13.0/52.9	44.4/123	52.2/152	148/225	279/534	506/719
20%	2.6/46.9	14.4 /95.9	52.3/153	120/211	255/627	781/955	1296/1089
30%	4.8/37.4	27.5/157	98.0/221	256/242	584/1135	1631/1083	4126/1381
40%	4.8/48.2	41.4/174	184.3/219	400/323	1205/915	3803/1481	7770/1693
50%	8.0/47.25	65.1/34.7	372/123	652/352	2212/785	8812/1468	14853/1892
—*	5.6/47.25	30.7/34.7	79.6/123	180/352	—	—	—
avg. time	4.1/39.0	25.4/103	91.7/168	202/256	881/737	2853/1104	5919/1270
gap	1.5%	0.1%	0.1%	0.2%	0.2%	0.3%	0.3%

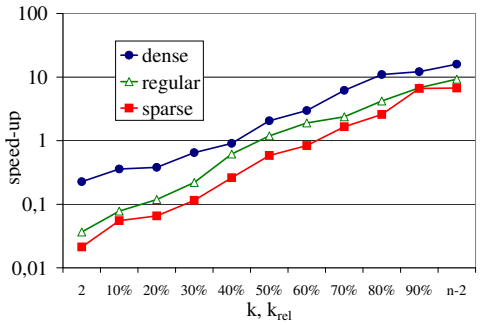


Fig. 3. Speed-up factors for dense, regular and sparse graphs with $|V| \leq 2000$ obtained when asymmetry constraints (21) are included in the initial LP

times. Rough estimates suggest a performance difference of about 150×. Overall, the VNDS metaheuristic by Urošević et al. [2004] is clearly faster than our exact approach, but on the other hand, it does not give optimal solutions. Table I also shows the average differences between the optimal solutions and the previously best known solutions (BKS), obtained by VNDS. We can observe that the running times of the VNDS approach do not increase as strongly as our approach with increasing k. We also report that the gaps of VNDS usually decrease with higher k.

In the following we will concentrate on the more common and diverse benchmark set (BB), and compare our results to those of HyEA and ACO. An additional advantage of this benchmark set is that the available data of the metaheuristics are much more detailed. Unless specified otherwise, we always report on the DC$^-$ algorithm, i.e., the Branch-and-Cut algorithm without using any heuristic for upper bounds.

Algorithmic Behaviour. Figure 3 illustrates the effectiveness of the asymmetry constraints (21) depending on increasing relative cardinality k_{rel}. We measured the speed-up by the quotient $\frac{t_\emptyset}{t_{\text{asy}}}$, where t_{asy} and $t_\emptyset$ denote the running time with and without using (21), resprectively. The constraints allow a speed-up by more than an order of magnitude for sparse, dense and regular graphs, but only for large cardinality $k \geq \frac{n}{2}$. Our experiments show that for smaller k, a variable x_{ri}, for some

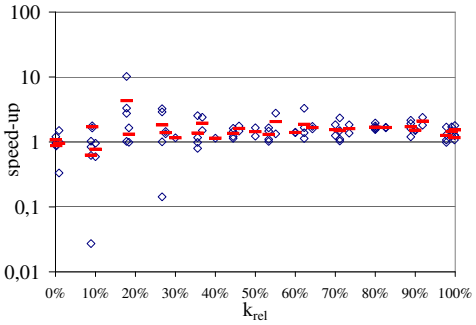


Fig. 4. Speed-up factors for the grid instances of (BB) when gsec-constraints (22) are separated. We have a diamond-shaped datapoint for each instance and k_{rel} value. This datapoint gives the corresponding speed-up factor achieved by the use of gsec-constraints. The short horizontal bars denote the average speed-up over all instances per k_{rel}.

$i \in V$, is quickly set to 1 and stays at this value until the final result. In these cases the constraints cannot help and only slow down the algorithm. Interestingly, the constraints are never profitable for the grid instances. For graphs with more than 2000 nodes using (21) is not possible due to memory restrictions, as the $\mathcal{O}(|V|^2)$ many asymmetry constraints are too much to handle. Hence, we ommitted these graphs in Figure 3.

We can validate these observations by reconsidering (UBM). In Table I we see the improvement achieved by introducing the asymmetry constraints for $k_{\text{rel}} = 50\%$.

We also report on the experiments with the special gsec-constraints (22) within the separation routine for the grid graphs. The clear advantage of these constraints is shown in Figure 4, which shows the speed-up factor $\frac{t_\emptyset}{t_{\text{gsec}}}$ obtained by the use of these constraints.

Based on these results we apply a simple rule for all the remaining experiments: We include the assymmetry constraints for all non-grid instances with less than 2500 nodes and $k \geq \frac{n}{2}$. For the grid instances we always separate the gsec-constraints (22).

In Table II, we show that the computation time is not only dependent on the graph size, but also on the density of the graph. Generally, we leave table cells empty if there is no problem instance with the according properties.

As described in Section 5.3, we also investigate the influence of primal heuristics in our Branch-and-Cut algorithm. For the tested instances with 1000 nodes the comparison of the running times of DC^+ and DC^- is shown in Figure 5. In general, our experiments show that DC^+ is only 10–30% percent faster than DC^- on average, even for the large graphs. Hence, we conclude that a bounding heuristic is not crucial for the success of our algorithm.

Runtime Comparison. Table III summarizes the average and median computation times of our algorithm, sorted by size and categorized according to the special properties of the underlying graphs. We can observe that performance does not differ significantly between the sparse, regular and dense graphs, but that the grid instances are more difficult and require more computational power. This was also noticed in [Ehrgott et al. 1997; Blum and Ehrgott 2003; Brimberg et al. 2006].

The behaviour of DC^- also has a dependency on k, see Figures 6(a), 6(c) and 6(d): for the sparse, dense, and regular instances the running time increases with increasing k for up to 50%. For larger k it remains relatively stable. In contrast to this,

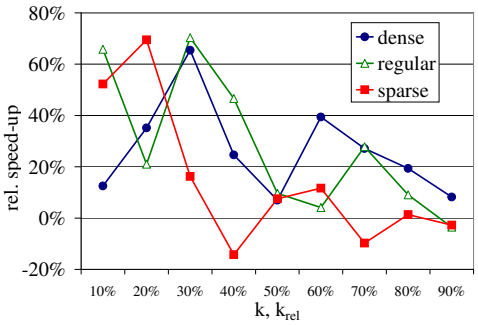


Fig. 5. Relative speed-up $\frac{(t_{\mathrm{DC}^-} - t_{\mathrm{DC}^+})}{t_{\mathrm{DC}^-}}$ (in percent) of DC^+ compared to DC^- for the instances with 1000 nodes.

Table II. Average CPU time (in seconds) over k_{rel} values of 10%, 20%, ..., 50%, sorted by the average degree of the graphs.

avg. deg	set	≤500 nodes	1000 nodes
2.5	(BB)	1.0	8.1
4	(BB)	0.9	11.6
10	(BB)	2.2	18.8
20	(UBM)	4.6	32.3
36.3	(BB)	6.3	—

Table III. Average/median CPU time (in seconds) and the average speed-up factor of DC^- to HyEA for the instance set (BB). Cells are left empty if there exists no instance matching the given criteria.

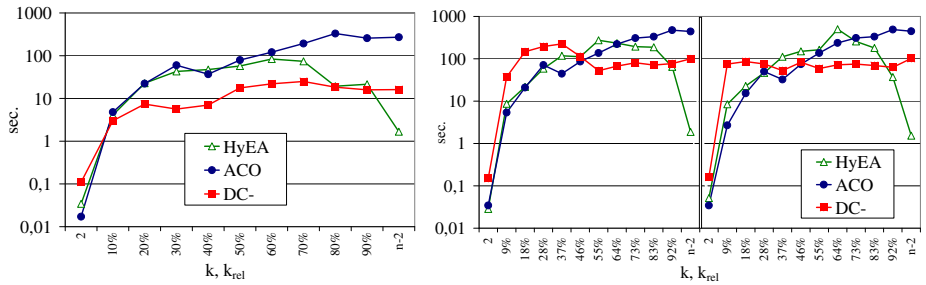
# nodes	500		1000–1089		2500	
group	avg/med	$\frac{t_{\mathrm{HyEA}}}{t_{\mathrm{DC}^-}}$	avg/med	$\frac{t_{\mathrm{HyEA}}}{t_{\mathrm{DC}^-}}$	avg/med	$\frac{t_{\mathrm{HyEA}}}{t_{\mathrm{DC}^-}}$
sparse	1.3/1.3	2.7	12.6/15.9	2.9	640.9/245.5	0.4
regular	1.0/1.0	3.9	15.3/16.9	7.5	—	—
dense	5.0/5.1	4.9	19.2/21.4	2.9	—	—
grid	7.0/1.0	0.6	82.4/74.9	1.7	3101.2/1973.7	0.2

solving the grid instances (cf. Figure 6(b)) is more difficult for the relatively small k-values.

The original experiments for HyEA and ACO were performed on an Intel Pentium IV, 3.06 GHz with 1GB RAM and a Pentium IV 2.4 GHz with 512MB RAM, respectively. Using the well-known SPEC performance evaluation [Spec 2008], we computed scaling factors of both machines to our computer: For the running time comparison we divided the times given in [Blum 2006] and [Bui and Sundarraj 2004] by 1.75 and 2.33, respectively. Anyhow, note that these factors are elaborate guesses at best and are only meant to help the reader to better evaluate the relative performance.

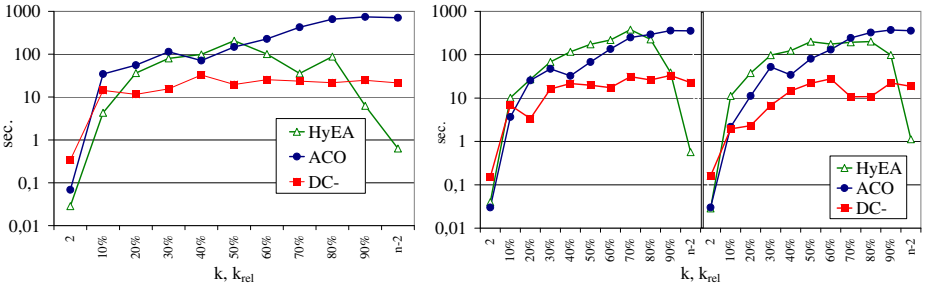
Table III additionally gives the average factor of $\frac{t_{\mathrm{HyEA}}}{t_{\mathrm{DC}^-}}$, i.e., the running time of our algorithm compared to (scaled) running time of HyEA. Analogously, Figure 6 shows the CPU time in (scaled) seconds of HyEA, ACO and our algorithm.

We observe that our DC^- algorithm performs better than the best metaheuristics in particular for the medium values of k, i.e., $40-70\%$ of $|V|$, on all instances with up to 1089 nodes, except for the very dense graph `le450_15a.g` with 450 nodes and 8168 edges, where HyEA was slightly faster. Interestingly, the gap between the



(a) sparse steind5.g (1000 nodes, 1250 edges)

(b) grid 33x33-1 and 33x33-2 (1089 nodes, 2112 edges)



(c) dense steind15.g (1000 nodes, 1250 edges)

(d) 4-regular g1000-4-1 and g1000-4-5.g (1000 nodes, 2000 edges)

Fig. 6. Running times of DC$^-$, HyEA, and ACO (in seconds) for instances of (BB) with ∼1000 nodes, depending on k. The figures for the grid and regular instances show the times for two different instances of the same type, respectively.

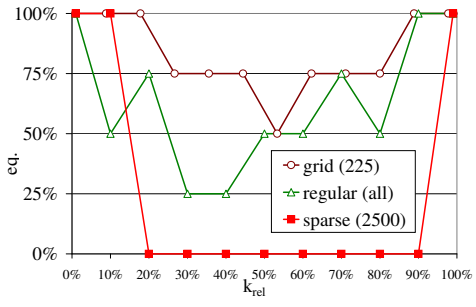


Fig. 7. Dependency of the BKS quality on k_{rel}, for selected instances. The vertical axis gives the percentage of the tested instances for which the BKS provided in [Blum and Blesa 2003] are optimal.

Table IV. Quality of previously best known solutions (BKS) provided in [Blum and Blesa 2003] for selected instances. "**eq.**" denotes the number of instances for which the BKS was optimal. For the other instances where BKS was not optimal, we give the average relative gap ($\mathbf{gap}_{\text{bks}}$) between OPT and BKS. For all instances we also give the average relative gap ($\mathbf{gap}_{\text{avg}}$) between the average solution of the metaheuristic and OPT—including the 0-gaps for optimally solved instances. All gaps are given in percent. Cells marked as "n/a" cannot be computed as the necessary data for ACO is not available.

instance	($\|V\|$,$\|E\|$)	**eq.**	$\mathbf{gap}_{\text{bks}}$	$\mathbf{gap}_{\text{avg}}$ **ACO**	$\mathbf{gap}_{\text{avg}}$ **HyEA**
regular g400-4-1.g	(400,800)	10/11	0.09	0.07	0.04
regular g400-4-5.g	(400,800)	8/11	0.19	0.31	0.35
regular g1000-4-1.g	(1000,2000)	7/11	0.07	0.65	0.12
regular g1000-4-5.g	(1000,2000)	3/11	0.08	0.45	0.35
sparse steinc5.g	(500,625)	11/11	—	0.97	0.06
sparse steind5.g	(1000,1250)	11/11	—	0.48	0.11
sparse steine5.g	(2500,3125)	3/11	0.13	n/a	0.23
dense le450a.g	(450,8168)	11/11	—	n/a	0.04
dense steinc15.g	(500,2500)	11/11	—	0.36	0.02
dense steind15.g	(1000,5000)	10/11	0.22	0.38	0.04
grid 15x15-1	(225,400)	13/13	—	1.27	0.18
grid 15x15-2	(225,400)	13/13	—	2.04	0.12
grid 45x5-1	(225,400)	4/13	0.54	n/a	1.22
grid 45x5-2	(225,400)	10/13	0.08	n/a	0.13
grid 33x33-1	(1089,2112)	3/12	0.31	1.70	0.57
grid 33x33-2	(1089,2112)	3/12	0.39	2.48	0.49
grid 50x50-1	(2500,4900)	2/11	0.95	n/a	1.27
grid 50x50-2	(2500,4900)	2/11	0.55	n/a	0.82

heuristic and the optimal solution tends to be larger especially for medium values of k (cf. next paragraph and Figure 7 for details).

Solution Quality. For each instance of (BB) we compared the previously best known solutions, see [Blum and Blesa 2003], with the optimal solution obtained by our algorithm, in order to assess their quality. Most of the best known solutions (BKS) were found by HyEA, followed by ACO. Note that these solutions where obtained by taking the best solutions over 20 independent runs per instance. In Table IV we show the number of instances for which we proved that BKS was in fact not optimal, and give the corresponding gap $\text{gap}_{\text{bks}} := \frac{\text{BKS}-\text{OPT}}{\text{OPT}}$ (in percent) per graph, averaged over the settings for k. Here OPT denotes the optimal objective value obtained by DC$^-$ and BKS denotes the best known solution obtained by either ACO or HyEA. Analogously, we give the gaps $\text{gap}_{\text{avg}} := \frac{\text{AVG}-\text{OPT}}{\text{OPT}}$ (in percent), where AVG denotes the average solution, over 20 runs, obtained by a metaheuristic. We observe that—concerning the solution quality—metaheuristics work quite well on instances with up to 1000 nodes and relatively small k. In particular, for $k = 2$ and $k = n - 2$ they always found an optimal solution.

Branch-and-Cut Specific Statistics. We conclude this part of the experimental study by analyzing certain propertes of DC$^-$ to better understand why it performs that well. Table V shows that the gaps between the LP-relaxation obtained at the root node of the Branch-and-Bound tree and the integer optimal solution are very

Table V. Behaviour of DC$^-$, depending on graph type, size ($|V|$), k_{rel} (in %), and additional constraints ($\emptyset$ denotes no additional constraints, *asym* and *gsec* denote the constraints (21) and (22), respectively). We give the number of generated dcut-constraints (*cuts*), the number of additional Branch-and-Bound nodes apart from the root (*B&B*), the gap between the LP-relaxation at the root and the optimal integer solution (*gap*), the overall computation time (*time*) and the percentage of that time spent at the root problem (*%rt*). The table ignores the irrelevant settings $k = 2$ and $k = |V| - 2$.

			sparse, dense, regular						**grid**					
			500		**1000**		**2500**		**225**		**1089**		**2500**	
	k_{rel}	$\lvert V\rvert$	$\emptyset$	asym	$\emptyset$	asym	$\emptyset$		$\emptyset$	gsec	$\emptyset$	gsec	$\emptyset$	gsec
cuts	**≤33**		15.3	6.3	39.7	11.2	64.0		622.3	674.2	961.7	710.0	2978.3	2924.0
	34–66		38.1	5.3	115.7	14.5	209.3		241.4	178.5	752.7	499.0	1770.0	1358.0
	≥66		115.5	6.3	250.7	22.2	460.0		107.8	69.9	410.3	242.0	1173.7	7130.0
B&B	**≤33**		7.0	2.9	8.1	3.8	5.3		84.7	108.8	7.3	7.7	9.5	16.3
	34–66		2.0	1.1	5.8	1.0	5.0		5.8	9.1	3.3	4.8	28.5	2.8
	≥66		0.7	0.3	0.8	0.5	7.7		0.3	0.4	1.7	1.8	0.8	0.5
gap	**≤33**		0.139%		0.032%		0.008%		2.645%		0.053%		0.046%	
	34–66		0.006%		0.003%		0.001%		0.148%		0.002%		0.001%	
	≥66		0.001%		0.001%		0.000%		0.006%		0.001%		0.000%	
time	**≤33**		1.9	7.2	16.9	59.6	45.3		14.5	24.7	152.5	102.5	6827.1	6901.9
	34–66		4.6	3.8	26.3	26.6	301.8		3.5	3.1	166.8	97.4	3201.4	2226.8
	≥66		24.1	3.2	71.2	21.9	1260.9		1.2	0.8	117.8	72.6	2790.6	1672.6
%rt	**≤33**		67.1%	80.3%	68.8%	76.1%	75.9%		18.6%	26.4%	85.16%	83.3%	82.5%	69.1%
	34–66		93.3%	88.9%	90.9%	93.3%	88.6%		80.4%	82.6%	95.9%	94.1%	97.2%	98.8%
	≥66		96.3%	96.7%	96.5%	97.1%	92.8%		98.9%	97.8%	95.9%	98.0%	99.0%	99.5%

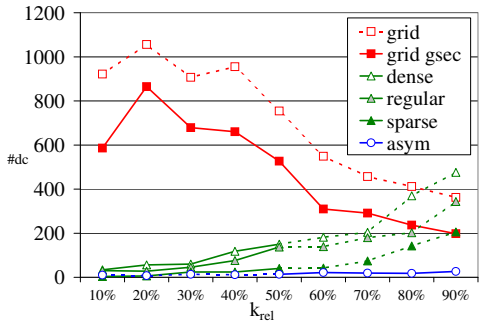


Fig. 8. Dependency of the number of generated dcut-constraints on k_{rel}, for instances with 1000–1089 nodes. The solid lines denote the parameter choices used in the comparative study. When using the asymmetry constraints (asym), the lines for sparse, regular, and dense graphs become visually indistinguishable; hence we show their average.

Table VI. Average running times (in seconds) of DC$^-$ using asymmetry constraints (21), for (UBM) instances with higher k_{rel}.

$\lvert V\rvert$ / k_{rel}	**500**	**1000**	**1500**	**2000**
60%	6.3	33.0	68.0	177.9
70%	5.0	27.8	59.8	165.6
80%	5.0	30.7	63.2	156.5
90%	4.4	32.4	53.8	151.1

tight and in fact often optimal. Furthermore, we observe that we need only very few cuts and branches to solve the whole ILPs.

The most interesting fact—in accordance with the runtime dependency on k—is visualized in Figure 8, selecting the graphs with 1000 nodes as a representative example. We see that for the sparse, regular, and dense graphs, the number of separated dcut-constraints grows with increasing k. Also observe that when using the asymmetry constraints (21), the number of necessary dcut-constraints drops further. In contrast to these observation, we see that for the grid graphs the number of required cuts is actually decreasing with growing k.

Further Instances. We investigated our algorithm's behaviour on two additional sets of graphs, not considered in other practical papers for the KCT problem.

(UBM) with larger cardinality: In contrast to the originally suggested cardinalities for (UBM), we also performed tests for $k_{\text{rel}} = \{60\%, \ldots, 90\%\}$. This allows us to investigate our algorithm's behaviour for large cardinalities and the influence of the asymmetry constraints (21). Table VI summarizes the average running times of our algorithm. We are able to solve all instances to provable optimality. The running times are even slightly decreasing for the larger k values, due to the help of (21). Overall, these results, as well as the speedup observed in Table I, confirms our finding that these constraints are beneficial for $k_{\text{rel}} \geq 50\%$.

Hypercubes: This benchmark set was introduced by Rossetti et al. [2001] and is also part of SteinLib [Koch et al. 2003]. It contains 6 hypercube graphs of dimension 6–12 with edge weights uniformly distributed in the interval [100,110]. A hypercube of dimension d has 2^d nodes interconnected to form the edges of a hypercube. We chose this benchmark set, as these graphs turned out to be highly challenging for the Steiner tree problem, as, e.g., the structure and the similar edge weights do not allow strong preprocessing strategies. In constrast to these

Table VII. Average running times for the hypercube instances.

dimension	**6**	**7**	**8**	**9**	**10**	**11**	**12**
# of nodes	64	128	256	512	1024	2048	4096
avg. time in sec.	0.03	0.15	1.02	8.73	44.63	289.77	2732.34

observation, our experiments show that in the context of the KCT and for the tested values of $k_{\text{rel}} = \{10\%, \ldots, 90\%\}$, these instances can be efficiently solved using DCut, cf. Table VII.

7. CONCLUSION

With the presented algorithm, we are able to optimally solve real-world sized instances of the k-Cardinality tree problem. In particular, we can solve all widely used benchmark instances to provable optimality. The running times are comparable, and for up to 1000 nodes even better, than the state-of-the-art metaheuristics.

Our paper shows that recent advances in computational power and ILP-solvers, if used in conjunction with strong ILP-formulations, allow exact algorithms to become feasible or even preferable alternatives to other, e.g., metaheuristic, approaches. Our results can therefore be seen as an example that we should not easily give up on exact algorithms only because the problem is NP-hard or because old approaches did not show their practical applicablility.

Although not the focus of this paper, we also applied our algorithm to known NKCT instances, cf. [Brimberg et al. 2006], using the transformation described in Section 4.1. Note that the transformation introduces unfortunate symmetries in the resulting KCA instance, as all arcs leading into the same node have identical costs. It would be worthwhile to investigate whether such a structure can be exploited via additional constraints, strengthening the DCUT formulation. For our experiments we use a permutation strategy on the arc weights to break the symmetry and thereby improve the practical behaviour of DC$^-$. Preliminary experiments show that we can solve all grid instances considered by Brimberg et al. [2006] (containing graphs with up to 2500 nodes) to provable optimality in under two hours. For graphs with up to 1600 nodes we require under half an hour. The special purpose metaheuristic for the NKCT is faster than our exact approach, though not optimal in most cases.

Finally, we want to conclude that even though there are (often LP-based) approximation algorithms for the KCT problem, we are unable to evaluate their practical performance and applicability. Unfortunately, they are only considered in a theoretical setting and there are no published implementations or studies. We hope that at some point the research in this interesting field will be taken into practice to experimentally evaluate its advantages and drawbacks.

ACKNOWLEDGMENT

We would like to thank Christian Blum and Dragan Urošević for kindly providing us with test instances and sharing their experience on this topic.

REFERENCES

Arora, S. and Karakostas, G. 2000. A $(2+\epsilon)$-approximation algorithm for the k-MST problem. In *Proc. ACM-SIAM Symposium on discrete algorithms (SODA'00)*. ACM Press, 754–759.

Blesa, M. J. and Xhafa, F. 2000. A C++ implementation of tabu search for k-cardinality tree problem based on generic programming and component reuse. In *Proc. Net.ObjectDays 2000*. tranSIT GmbH, 648–652.

Blum, A., Ravi, R., and Vempala, S. 1996. A constant-factor approximation algorithm for the k-MST problem. In *Proc. ACM Symposium on Theory of Computing (STOC'96)*. ACM Press, 442–448.

Blum, C. 2006. A new hybrid evolutionary algorithm for the huge k-cardinality tree problem. In *Proc. Genetic and Evolutionary Computation Conference (GECCO'06)*. ACM Press, 515–522.

Blum, C. 2007. Revisiting dynamic programming for finding optimal subtrees in trees. *European Journal of Operational Research 177,* 1, 102–115.

Blum, C. and Blesa, M. 2003. KCTLIB – a library for the edge-weighted k-cardinality tree problem. `http://iridia.ulb.ac.be/ cblum/kctlib/`.

Blum, C. and Blesa, M. 2005a. Combining ant colony optimization with dynamic programming for solving the k-cardinality tree problem. In *Proc. International Work-Conference on Artificial Neural Networks (IWANN'05)*. LNCS, vol. 3512. Springer, 25–33.

Blum, C. and Blesa, M. J. 2005b. New metaheuristic approaches for the edge-weighted k-cardinality tree problem. *Computers & OR 32*, 1355–1377.

Blum, C. and Ehrgott, M. 2003. Local search algorithms for the k-cardinality tree problem. *Discrete Applied Mathematics 128,* 2–3, 511–540.

Brimberg, J., Urošević, D., and Mladenović, N. 2006. Variable neighborhood search for the vertex weighted k-cardinality tree problem. *European Journal of Operational Research 171,* 1, 74–84.

Bui, T. N. and Sundarraj, G. 2004. Ant system for the k-cardinality tree problem. In *Proc. Genetic and Evolutionary Computation Conference (GECCO'04)*. LNCS, vol. 3102. Springer, 36–47.

Chimani, M., Kandyba, M., and Mutzel, P. 2007. A new ILP formulation for 2-root-connected prize-collecting Steiner networks. In *Proc. European Symposium on Algorithms (ESA'07)*. LNCS, vol. 4698. Springer, 681–692.

Chudak, F. A., Roughgarden, T., and Williamson, D. P. 2001. Approximate k-MSTs and k-Steiner trees via the primal-dual method and Lagrangean relaxation. In *Proc. Integer Programming and Combinatorial Optimization (IPCO '01)*. LNCS, vol. 2081. Springer, 60–70.

Ehrgott, M. and Freitag, J. 1996. K_TREE/K_SUBGRAPH: a program package for minimal weighted k-cardinality tree subgraph problem. *European Journal of Operational Research 1,* 93, 214–225.

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

Fischetti, M., Hamacher, W., Jornsten, K., and Maffioli, F. 1994. Weighted k-cardinality trees: complexity and polyhedral structure. *Networks 24*, 11–21.

Gale, D. 1957. A theorem on flows in networks. *Pacific Journal of Mathematics 7*, 1073–1082.

Garg, N. 1996. A 3-approximation for the minimum tree spanning k vertices. In *Proc. Symposium on Foundations of Computer Science (FOCS'96)*. IEEE Computer Society, 302–309.

Garg, N. 2005. Saving an epsilon: a 2-approximation for the k-MST problem in graphs. In *Proc. ACM Symposium on Theory of Computing (STOC'05)*. ACM Press, 396–402.

Goemans, M. X. and Myung, Y. 1993. A catalog of Steiner tree formulations. *Networks 23*, 19–28.

Goemans, M. X. and Williamson, D. P. 1995. A general approximation technique for constrained forest problems. *SIAM Journal on Computing 24,* 2, 296–317.

Joernsten, K. and Lokketangen, A. 1997. Tabu search for weighted k-cardinality trees. *Asia Pacific Journal of Operational Research 14*, 9–26.

Koch, T. and Martin, A. 1998. Solving Steiner tree problems in graphs to optimality. *Networks 32*, 207–232.

Koch, T., Martin, A., and Voss, S. 2003. SteinLib – an updated library on Steiner tree problems in graphs. `http://elib.zib.de/steinlib`.

Ljubić, I. 2004. Exact and memetic algorithms for two network design problems. Ph.D. thesis, Technische Universität Wien.

Ljubić, I., Weiskircher, R., Pferschy, U., Klau, G., Mutzel, P., and Fischetti, M. 2006. An algorithmic framework for the exact solution of the prize-collecting Steiner tree problem. *Mathematical Programming, Series B 105,* 2–3, 427–449.

Ravi, R., Sundaram, R., Marathe, M. V., Rosenkrantz, D. J., and Ravi, S. S. 1996. Spanning trees – short or small. *SIAM Journal of Discrete Mathematics 9*, 128–200.

Rossetti, I., de Aragão, M. P., Ribeiro, C., Uchoa, E., and Werneck, R. F. 2001. New benchmark instances for the Steiner problem in graphs. In *Proc. Metaheuristics International Conference (MIC'01)*. 557–561.

Segev, A. 1987. The node-weighted Steiner tree problem. *Networks 17,* 1, 1–17.

Spec. 2008. Standard performance evaluation corporation. `http://www.spec.org/`.

Urošević, D., Brimberg, J., and Mladenović, N. 2004. Variable neighborhood decomposition search for the edge weighted k-cardinality tree problem. *Computers & OR 31,* 8, 1205–1213.

Wolsey, L. A. 1998. *Integer programming.* Wiley-Interscience.

Received April 2008; revised Month Year; accepted Month Year