Heuristics for the K-Cardinality Tree and Subgraph Problems

Matthias Ehrgott *
Universität Kaiserslautern

Jörg Freitag
Colonia Insurance

Horst W. Hamacher *
Universität Kaiserslautern

Francesco Maffioli
Politecnico di Milano†

Abstract

In this paper we consider the problem of finding in a given graph a minimal weight subtree or connected subgraph, which has a given number of edges. These NP-hard combinatorial optimization problems have various applications in the oil industry, in facility layout and graph partitioning. We will present different heuristic approaches based on spanning tree and shortest path methods and on an exact algorithm solving the problem in polynomial time if the underlying graph is a tree. Both the edge- and node weighted case are investigated and extensive numerical results on the behaviour of the heuristics compared to optimal solutions are presented. The best heuristics yielded results within an error margin of less than one percent from optimality for most cases. In a large percentage of tests even optimal solutions have been found.

1 Introduction

The problem of finding in a given graph a subtree of minimal weight with a fixed number K of edges, called the K-cardinality tree problem, is a combinatorial optimization problem first described in [HJM91]. Its structure was investigated in detail in [FHJM94]. A closely related problem is the K-cardinality subgraph problem, which was the topic of a thesis by the first author [Ehr92].

There are several applications which require to find a connected subset of a given graph. The first application is in oil field leasing in the Norwegian part of the North Sea, see [HJ93]. The policy of the Norwegian Government is to give blocks of an oil field to the companies allowing them to explore these fields for six years. After this period at least half of the block has to be returned, under the restriction that the returned part is connected. This problem can be modeled mathematically using a grid graph, the nodes of which represent the subsquares of the blocks. Subsquares have a size of one longitudinal by one latitudinal minute. Edges are introduced between two nodes if the corresponding subsquares are neighbours. The objective is then to find a connected subgraph of the grid graph of minimal value and containing at least half of the nodes. The part of the block which is represented by this subgraph is then returned. The values of the subsquares will be established by the company based on their explorations during the first six years.

A second possible application is in the area of facilities layout and graph partitioning, see [FH92]. In facilities layout we consider a total office area or machine hall which has to be subdivided to accommodate the particular rooms. An office is then a connected subarea consisting of a given number a_l of unit squares. Feasible solutions are provided by a partition of a grid graph G into L connected subgraphs of a_l nodes, $l = 1, \ldots, L$.

*Partially supported by a grant of the Deutsche Forschungsgemeinschaft and grant ERBCHRXCT930087 of the European HC&M Programme

†Partially supported by grant ERBCHRXCT930087 of the European HC&M Programme

The outline of the paper is as follows. Section 2 provides the mathematical formulation of the two problems under consideration. We also review complexity results and lower bounds. In Section 3 we will present heuristics for both problems for the case of edge-weighted graphs. Section 4 deals with the node-weighted case. We will prove that for K-Cardinality Tree the node-weighted case can be transformed to the edge weighted case and give heuristics for K-Card Subgraph on node-weighted graphs, where the transformation is not valid. Finally in Section 5 the numerical results for the heuristics developed in Sections 3 and 4 are given. We will show that the heuristics we present yield very good solutions (in 80% of the test problems an optimal solution was found) within a few seconds, whereas exact methods were not applicable to find optimal solutions for larger problems.

2 Mathematical Formulations

First we will provide a formal graph theoretical definition of the two problems considered in this paper. Let $G = (V, E, w)$ be an undirected edge-weighted graph. The number of nodes of G is given by $|V| = n$, the number of edges by $|E| = m$. Edge weights are defined according to (edge-) weight function $w : E \to I\!R_+$. For any subset $E' \subseteq E$ we denote the weight of E' by $w(E') = \sum_{e \in E'} w(e)$.

Definition 1
- *A* K-cardinality tree *is a subtree T of G such that $|E(T)| = K$. The* K-Card Tree *problem is to find a K-cardinality tree of minimal weight, i.e.*

$$\min\{w(E(T)) \,|\, T \text{ is a K-cardinality tree}\}$$

- *A* K-cardinality subgraph *is a connected subgraph S of G such that $|E(S)| = K$. The* K-Card Subgraph *problem is to find a K-cardinality subgraph of minimal weight, i.e.*

$$\min\{w(E(S)) \,|\, S \text{ is a K-cardinality subgraph}\}$$

Notice that we do not require a K-cardinality subgraph to be an induced subgraph of G. We just understand it as a subset of edges, which together with the nodes incident to these edges form a connected graph. It is also important to note that K may range from 1 to $n-1$ for K-Card Tree, whereas for K-Card Subgraph values for K up to $m-1$ are possible. If $K = n-1$ K-Card Tree reduces to the Minimum Spanning Tree problem, and hence is easily solvable. In general, however, both problems are NP-hard, as was shown in [FHJM94]. We also refer to [Woe92], where NP-completeness of the problem on grid graphs has been shown. The interesting cases occur for $K \geq 3$, since for $K \in \{1, 2\}$ both problems are also easily solvable. (In fact the problems are solvable in polynomial time if K is fixed, i.e. not a part of the input, by just checking all $K-$element subsets of edges.)
Exact algorithms for both problems have been derived on the basis of a Branch&Cut approach, for which formulations as 0-1 Programming problems are necessary. To this end binary variables for nodes and edges are introduced. Let S, T be a subgraph, respectively subtree of G and define

$$x(e) = \begin{cases} 1 & e \in E(T), E(S) \quad \text{respectively} \\ 0 & \text{otherwise} \end{cases}$$

$$y(i) = \begin{cases} 1 & i \in V(T), V(S) \quad \text{respectively} \\ 0 & \text{otherwise} \end{cases}$$

The 0-1 Programming formulation corresponding to K-Card Tree is

$$\min \sum_{e \in E} w(e)x(e) \tag{1}$$

$$\begin{aligned}
\sum_{e \in E} x(e) &= K && (2)\\
\sum_{i \in V} y(i) &= K+1 && (3)\\
\sum_{e \in E(U)} x(e) &\leq \sum_{i \in U} y(i) - y(t) \quad \forall U \subseteq V,\ |U| \geq 2,\ t \in S && (4)\\
& \text{where } E(U) = \{[i,j] \in E \,|\, i,j \in U\}
\end{aligned}$$

The interpretation of (1) - (4) is as follows. Minimize total edge weight subject to the constraints that K edges and $K+1$ nodes are chosen and (4) is fulfilled. Hence (4) should guarantee acyclicity, which it indeed does, since it implies that from all edges which have both endnodes in a subset U of $V(G)$ at most $|U|-1$ can be chosen. A formal proof of the correctness of (1)-(4) as 0-1 Programming formulation for K-Card Tree is given in [FHJM94].
The corresponding formulation for K-Card Subgraph cannot make use of (3) and (4). But it must guarantee connectedness, which means that whenever a node s in some subset U of $V(G)$ and a node t in $V \setminus U$ are contained in a solution, i.e. $y(s) = y(t) = 1$, there must be at least one edge with one endnode within U and the other outside U. This is expressed by (7) below. Furthermore if an edge $e = [i,j]$ belongs to a solution it must hold that $y(i) = y(j) = 1$, i.e. both endnodes also belong to the solution. Thus the 0-1 Programming formulation for K-Card Subgraph is given by (5)-(9).

$$\min \sum_{e \in E} w(e)x(e) \tag{5}$$

$$\begin{aligned}
\sum_{e \in E} x(e) &= K && (6)\\
\sum_{e \in E(U:V \setminus U)} x(e) &\geq y(s) + y(t) - 1 \quad \forall U \subseteq V,\ s \in U,\ t \in V \setminus U && (7)\\
& \text{where } E(U:V \setminus U) = \{[i,j] \in E \,|\, i \in U, j \in V \setminus U\}\\
x(e) &\leq y(i) \quad \forall e = [i,j] \in E && (8)\\
x(e) &\leq y(j) \quad \forall e = [i,j] \in E && (9)
\end{aligned}$$

For a proof we refer to [Ehr92] and [Ehr95].
The structure of the polyhedra defined by the convex hull of all integral solutions of (2)-(4) and (6)-(9) have been investigated in detail in [FHJM94] and [Ehr92], [Ehr95] respectively. The most important results for our purposes are two separation algorithms for inequalities (4) and (7), based on network flow techniques, which were used to implement Branch&Cut algorithms for both problems. For more detailed analysis of these algorithms and also for implementations we refer to [FHJM94], [Ehr92], [Ehr95], [Fre93] and especially [EF96]. We used these implementations to find the optimal solutions for the numerical tests presented in Section 5.
We should note that for the special case where G is itself a tree the K-Card Tree problem (and thus the K-Card Subgraph problem, since both problems coincide in that case) is solvable in polynomial time. An algorithm based on Dynamic Programming was given in [Maf91] and implemented in [Fre93]. Another algorithm for this special case has been developed in [FK94].
Lower bounds for both problems can easily be obtained. The lower bound for an instance of K-Card Subgraph is given by the K edges with the K smallest weights. This subset of edges can be shown to be the solution of the linear relaxation of (5)-(9) without constraints (7), i.e. $\min \sum_{e \in E} w(e)x(e)$ subject to (6), (8), (9) and $x(e), y(i) \geq 0$, which therefore always has an integral optimal solution.
For K-Card Tree we can apply Kruskal's [Kru56] well-known greedy algorithm for the Minimum Spanning Tree problem to G and stop as soon as K edges have been selected. Obviously the result

is an acyclic subgraph of G with K edges, i.e. a K-cardinality forest. This solution is also the optimal solution of the Integer Program (1)-(4) without constraint (3).
This indicates that the difficulty of solving K-cardinality tree and K-cardinality subgraph problems is caused by the connectivity requirement. Recall that connectivity for trees is implied by acyclicity and the fact that the number of nodes is the number of edges plus one. Thus relaxing (3) for K-Card Tree and (7) for K-Card Subgraph we omit the connectivity property and the resulting problems are easy.
The following section will be devoted to heuristic methods for the problems under discussion. We will present several different approaches for the K-Card Tree problem, some of which will be modified and adapted for K-Card Subgraph.

3 Heuristics

This section is subdivided. In Section 3.1 we will describe three classes of heuristics for K-Card Tree which were developed in [Fre93]. Section 3.2 is concerned with the heuristics for K-Card Subgraph. Throughout, the input is always an edge-weighted graph $G = (V, E, w)$ together with a cardinality K, as described in Section 1. The solution produced by the heuristic will always be denoted by T_{heur} for K-Card Tree heuristics and by S_{heur} for K-Card Subgraph heuristics. We also remark that the formulation of the methods is rather informal. For implementations we refer to [Fre93] and [Ehr92]. The codes are also available within the Operations Research Software Exchange Program, see [EF96].

3.1 Heuristics for K-Card Tree

A straightforward approach to solve K-Card Tree is to look at spanning tree algorithms. In Section 2 we already mentioned that Kruskal's algorithm can be used to derive a lower bound. The first class of heuristics will also be based on a greedy approach. The first heuristic proceeds as the second well-known algorithm for minimum spanning trees, the one by Prim [Pri57].

K-CardPrim (K-CaPr)

```
1  For all s ∈ V
       V(T_s) := {s}, E(T_s) := ∅
       For k = 1, ..., K
           Let e ∈ E \ E(T_s) be an edge of minimal weight such that
               T_s ∪ {e} is a tree
           T_s := T_s ∪ {e}
2  T_heur := argmin{w(T_s) | s = 1, ..., n}
```

The algorithm constructs n K-cardinality trees, starting once from each node s of G according to Prim's method. That means edges of minimal weight are added to the current solution, which is initially set to $\{s\}$, until it contains K edges. In each step it is guaranteed that the current solution is a subtree of G. Finally the best of the n trees is chosen as heuristic solution.
Since Kruskal's greedy approach in general will only produce a K-cardinality forest it cannot be applied as a heuristic. But we may start from the given graph G and delete edges in a greedy fashion until a K-cardinality tree is encountered. In each step we have to make sure that the remaining graph contains at least one connected component with at least $K + 1$ nodes. The procedure stops as soon as a K-cardinality tree is found.

DualGreedy1 (DuGr1)

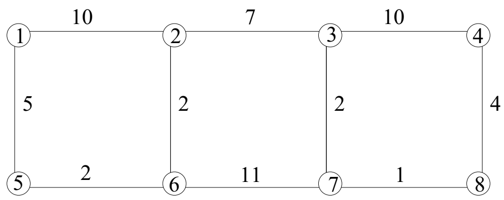
1 $T := G$
2 Repeat
  Let $e \in E(T)$ be an edge of maximal weight such that
    $T \setminus \{e\}$ has at least one component with at least $K + 1$ nodes
  $T := T \setminus (\{e\} \cup \{\text{components of } T \text{ with at most } K \text{ nodes}\})$
  until $|E(T)| = K$
3 $T_{heur} := T$

Instead of checking if a large enough component exists it is also possible to require that the graph resulting from deletion of an edge remains connected. This is done in the following version of the dual greedy approach.

DualGreedy2 (DuGr2)

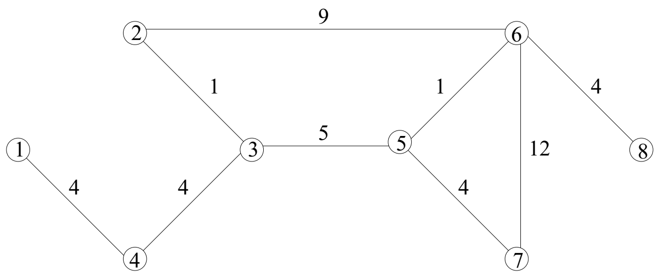
1 $T := G$
2 Repeat
  Let $e \in E(T)$ be an edge of maximal weight such that
    $T \setminus \{e\}$ is connected and has at least $K + 1$ nodes
  $T := T \setminus \{e\}$
  until $|E(T)| = K$
3 $T_{heur} := T$

Numerical tests (see Section 5) indicated that the performance of **DualGreedy2** is bad for small values of K. However it yields better results than **DualGreedy1** for larger K. **DualGreedy1** on the other hand is good for small K. Moreover the dual greedy heuristics often found good or optimal solutions exactly in those cases where **K-CardPrim** did not.
The example given in Figure 1 shows why both versions are considered: in general it is not possible to say that one dominates the other.
We have found small examples illustrating nonoptimality for all the heuristics to be presented below. Especially in the cases where we included several slightly different versions of one approach examples showing that none of the versions is dominating the other(s) are known. However we will omit them for the sake of brevity.
The second class of algorithms is based on shortest path methods. The idea of the approach is the following: First try to find paths that have at most K edges and extend these paths to K-cardinality trees using Prim's method. Before we can describe these methods we will have to present a modified version of Dijkstra's Shortest Path algorithm [Dij59], which accomplishes the construction of paths of at most K edges.
We need two labels, $len[x]$ and $dist[x]$ to denote the number of edges and the total weight of the best path from specified node s to x found so far. The node from which the labeling is done, i.e. the one which has most recently been labeled permanently, is denoted by *recent*. What we have to do is to change the labeling procedure in such a way that a node x is only relabeled if the path involving *recent* is shorter (i.e. has a smaller weight) than the current label $dist[x]$ and does not have more than K edges. The choice of the next node which gets a permanent label is only among nodes to which the best path found so far has not more than K edges.



Graph 1, K=3

DualGreedy1 solution: 3-7, 7-8, 4-8, weight 7
DualGreedy2 solution: 2-3, 3-7, 7-8, weight 10



Graph 2, K=3

DualGreedy1 solution: 5-6, 5-7, 6-8, weight 9
DualGreedy2 solution: 2-3, 3-5, 5-6, weight 7

Figure 1: Example Illustrating DualGreedy1 and DualGreedy2 for K-Card Tree

K-Dijkstra-A

1 $W := \{s\}$
$dist[x] := \infty \quad \forall x \in V \setminus \{s\}, \quad dist[s] := 0$
$len[x] := \infty \quad \forall x \in V \setminus \{x\}, \quad len[s] := 0$
$recent := s$
2 While $W \neq V$
a For all $x \notin W$ such that $[recent, x] \in E$
If $(dist[x] > dist[recent] + w([recent, x]))$ and $(len[recent] + 1 \leq K)$
then $dist[x] := dist[recent] + w([recent, x]), len[x] := len[recent] + 1$
b If $\{y \,|\, len[y] < K\} \neq \emptyset$
then $x := \text{argmin}\{dist[y] \,|\, y \notin W, len[y] < K\}$, $W := W \cup \{x\}, recent := x$
else $W := V$

K-Dijkstra-A will produce paths from node s to some other nodes of V the number of edges of which is at most K. But if the algorithm finds a path from s to, say, t it is not necessarily the shortest path between s and t with at most K edges.
As mentioned before in **K-Dijkstra-A** a new label for node x is set if the path using *recent* has strictly smaller weight and not more than K edges. We could change that in the following manner. Provided that the path using *recent* has at most K edges, relabel x if this path has smaller weight or more edges. Thus in some cases, when the path using *recent* has the same weight as given by $dist[x]$ and x would not be relabeled, it will be in this version, if it has more edges than the path found so far. That means the priority is more on the cardinality of the paths instead of on the weights as in the first version. Hence paths of cardinality K will be found more often. The second modified Dijkstra version is obtained by substituting **2a** in **K-Dijkstra-A** by

For all $x \notin W$ such that $[recent, x] \in E$
If $len[recent] + 1 \leq K$
then if $(dist[x] > dist[recent] + w([recent, x]))$ or $(len[recent] + 1 > len[x])$
then $dist[x] := dist[recent] + w([recent, x]), len[x] := len[recent] + 1$

The second version is called **K-Dijkstra-B**. The next step is to incorporate **K-Dijkstra-A** and **K-Dijkstra-B** in a heuristic for the K-Card Tree problem. We first apply the modified Dijkstra algorithms to G once for each node s of V. The resulting set of paths $\{P_{ij} \mid i, j \in V\}$ is checked for paths of k edges, where $k \in \{1, \ldots, K\}$. Note that possibly $P_{ij} = \emptyset$, which means that there is no path from i to j of no more than K edges. For each k the best of these paths is chosen and extended, if necessary, to a K-cardinality tree. The best of the resulting trees defines the heuristic solution.

DijkstraPrim-A(B)

1 For all $s \in V$
Apply K-Dijkstra-A(B)
Let $\{P_{ij} \mid i, j \in V\}$ be the resulting set of paths
Let $dist[i, j]$ be the weights and let $len[i, j]$ be the cardinalities of P_{ij}
2 For $k = 1, \ldots, K$
If $\{(i, j) \in V \times V \mid len[i, j] = k\} \neq \emptyset$
then $P_k = \text{argmin}\{dist[i, j] \mid len[i, j] = k; i, j \in V\}$
Apply Prim's method to extend P_k to a K-cardinality tree T_k
3 $T_{heur} := \text{argmin}\{w(T_k) \mid k = 1, \ldots, K\}$

The final heuristic combines **DijkstraPrim-A** and **DijkstraPrim-B**, by choosing as heuristic solution the better solution of the two algorithms. It should be remarked that in general neither of the two versions performs better than the other.

DijkstraPrim (DiPr)

1 Let T_A be the K-cardinality tree resulting from DijkstraPrim-A
2 Let T_B be the K-cardinality tree resulting from DijkstraPrim-B
3 $T_{heur} := \text{argmin}\{w(T_A), w(T_B)\}$

In **DijkstraPrim** we extended "short" paths to K-cardinality trees. But since the problem of finding shortest paths from a specified node s of G to all other nodes having at most K edges is polynomially solvable (see [Ehr92]) it would also be possible to use exact algorithms instead of the modified Dijkstra versions presented above. But numerical tests indicated that both the quality of the results and time (due to increased complexity) are worse for the resulting heuristics, see [Fre93]. We will therefore not go into details here.
The last class of heuristics is based on the Dynamic Programming algorithm of [Maf91] for K-cardinality subtrees of trees. This algorithm will be called **DynamicTree**. For an implementation of the algorithm, which has a complexity of $O(nK^2)$, see [Fre93]. The idea of the heuristic is simple: Find a "good" spanning tree of G and then apply **DynamicTree**. Proceeding in that way we can guarantee that at least an optimal K-cardinality tree in the spanning tree, to which **DynamicTree** was applied, is found.
The performance of this approach depends very much on the choice of the spanning tree, which need not necessarily be a minimum spanning tree. Others sometimes yield better solutions. We will present three versions which differ only in the spanning trees to which we apply **DynamicTree**. The first version uses spanning trees found by Prim's algorithm.

DynamicPrimTree (DPT)

1 For all $s \in V$
  Apply Prim's algorithm starting at s to find a minimum spanning tree ST_s
  Apply DynamicTree to ST_s to find a K-cardinality tree T_s
2 $T_{heur} := \text{argmin}\{w(T_s)|s = 1, \ldots, n\}$

To have a still greater variety of spanning trees to start with we combine the **DynamicTree** algorithm with the path heuristics we described before. First paths as in **DijkstraPrim** are generated, these are this time extended to spanning trees of G before **DynamicTree** is applied.

DynamicDijkstraPath (DDP)

1 For all $s \in V$
  Apply K-Dijkstra-A and K-Dijkstra-B
 Let $\{P_{ij}^A \,|\, i,j \in V\}$ be the set of paths resulting from K-Dijkstra-A
 Let $dist[i,j]^A$ be the weights and $len[i,j]^A$ be the cardinalities of P_{ij}^A
 Let $\{P_{ij}^B \,|\, i,j \in V\}$ be the set of paths resulting from K-Dijkstra-B
 Let $dist[i,j]^B$ be the weights and $len[i,j]^B$ be the cardinalities of P_{ij}^B
2 For $k = 1, \ldots, K$
  If $\{(i,j) \in V \times V \,|\, len^A[i,j] = k\} \neq \emptyset$
   then $P_k^A := \text{argmin}\{dist^A[i,j] \,|\, len^A[i,j] = k;\, i,j \in V\}$
    Apply Prim's method to extend P_k^A to a spanning tree ST_k^A
    Apply DynamicTree to ST_k^A to find a K-cardinality tree T_k^A
  If $\{(i,j) \in V \times V \,|\, len^B[i,j] = k\} \neq \emptyset$
   then $P_k^B := \text{argmin}\{dist^B[i,j] \,|\, len^B[i,j] = k;\, i,j \in V\}$
    Apply Prim's method to extend P_k^B to a spanning tree ST_k^B
    Apply DynamicTree to ST_k^B to find a K-cardinality tree T_k^B
3 $T_{heur} := \text{argmin}\{w(T_k^X) \,|\, X \in \{A,B\}, k \in \{1, \ldots, K\}\}$

In the last version we will again use **K-Dijkstra-A(B)**. But now we make use of the fact that the sets of paths $\{P_{ij} \,|\, i,j \in V\}$ found by these algorithms define subtrees of G, which follows from the definition of Dijkstra's algorithm. (This property in general does not hold for the set of shortest paths of at most K edges). However in general the number of edges of these subtrees is not known and will in most cases be greater than K. Hence, applying Prim's method to these subtrees we construct spanning trees before we apply **DynamicTree**.

DynamicDijkstraTree (DDT)

1 For all $s \in V$
  Apply K-Dijkstra-A and K-Dijkstra-B to find two subtrees PT_s^A, PT_s^B of G
  Apply Prim's method to extend PT_s^A, PT_s^B to spanning trees ST_s^A, ST_s^B
  Apply DynamicTree to ST_s^A, ST_s^B yielding T_s^A, T_s^B
2 $T_{heur} := \text{argmin}\{w(T_s^X) \,|\, s \in \{1, \ldots, n\}, X \in \{A,B\}\}$

We should remark that none of the three dynamic tree heuristics is generally better than another. Given any of the three there are examples where this one outperforms the other two.

3.2 Heuristics for K-Card Subgraph

Compared to the variety of approaches for K-cardinality trees we only have one approach for this problem. It is the counterpart of the greedy type heuristics of Section 3.1.

K-CardPrim-Subgraph (K-CaPr-S)

1 For all $s \in V$
  $V(S_s) := \{s\}, E(S_s) := \emptyset$
  For $k = 1, \ldots, K$
    Let $e \in E \setminus E(S_s)$ be an edge of minimal weight such that
      $S_s \cup \{e\}$ is connected
    $S_s := S_s \cup \{e\}$
2 $S_{heur} := \text{argmin}\{w(S_s) \mid s = 1, \ldots, n\}$

Hence we just relax the condition that S_s is acyclic. Thus the heuristic will produce subgraphs of G containing cycles, if edges generating cycles have smaller weight than others.
To apply the dual greedy approach we take the same heuristics as for K-Card Tree and modify the selection rule for edges: after deleting an edge the remaining graph has to contain a component with at least K edges, respectively has to remain connected. We get the two following heuristics.

DualGreedy1-Subgraph (DuGr1-S)

1 $S := G$
2 Repeat
  Let $e \in E(S)$ be an edge of maximal weight such that
    $S \setminus \{e\}$ has at least one component with at least K edges
  $S := S \setminus (\{e\} \cup \{\text{components of } S \text{ with less than } K \text{ edges}\})$
  until $|E(S)| = K$
3 $S_{heur} := S$

DualGreedy2-Subgraph (DuGr2-S)

1 $S := G$
2 Repeat
  Let $e \in E(S)$ be an edge of maximal weight such that
    $S \setminus \{e\}$ is connected
  $S := S \setminus \{e\}$
  until $|E(S)| = K$
3 $S_{heur} := S$

Comparing **K-CardPrim-Subgraph** and the dual greedy heuristics the same remarks we made in the case of K-Card Tree apply.
Concerning the other approaches of Section 3.1 we note that the path based approaches are not advantageous here since they unnecessarily stress acyclicity of (partial) solutions: a path is always a tree. Obviously the **DynamicTree**-type heuristics are not applicable for K-Card Subgraph. Furthermore it is not known if K-Card Subgraph is solvable polynomially for any specific class of graphs (except trees, since both problems coincide in that case). This would possibly provide an algorithm that can be transformed to a heuristic for the general case.

4 Node Weighted Problems

In applications, especially in the oil field leasing problem mentioned in Section 1, it sometimes seems more natural to formulate the problem for a graph with node weights instead of edge weights. The weight of a node would then just represent the value of the corresponding subsquare of the block in the oil field problem.
Thus let us now consider a node-weighted graph $G = (V, E, w_V)$ where $w_V : V \to I\!R_+$. The problem is to find a K-cardinality tree (subgraph) in G minimizing total node weight. Both K-Card Tree and K-Card Subgraph are also NP-hard for the node-weighted case as has been shown

in [Ehr92]. In the IP-formulations (1)-(4) ((5)-(9)) we only have to change the objective function to

$$\min \sum_{i \in V} w_V(i) y(i)$$

to deal with the new problem.
The heuristics, however, are designed for the edge-weighted case and could not directly be used for the node-weighted case. To avoid the development of new heuristics we will prove a transformation of K-Card Tree for node-weighted graphs to the edge-weighted case. Hence the methods of Section 3.1 will be applicable to solve node-weighted problems as well.
Given a node-weighted graph $G = (V, E, w_V)$ we will define the related edge-weighted graph $G' = (V', E', w)$ with weight-function $w : E' \to \mathbb{R}^+$, where $V' = V \cup \{v' \mid v \in V\}$, $E' = E \cup \{[v, v'] \mid v \in V\}$ and

$$w(e) = \begin{cases} w_V(v) & e = [v, v'] \\ M := (m+1)\max_{v \in V} w_V(v) & e \in E \end{cases}$$

Thus we just duplicate the nodes and introduce edges between the two copies of each node which carry the original node weight. The edges of G are charged with a large weight M. A property which is implied by the definition of G' is given in Lemma 1, the proof of which is straightforward.

Lemma 1 *Let $S' = (V(S'), E(S'))$ be a connected subgraph of G'. Then $S = (V(S), E(S))$ is a connected subgraph of G, where $E(S) = E(S') \backslash \{[v, v'] \mid [v, v'] \in E(S)\}$ and $V(S) = V(S') \backslash \{v' \mid v' \in E(S')\}$.*

As a corollary we state the implication of Lemma 1 for subtrees of G'.

Corollary 1 *Any $(2K + 1)$-cardinality tree in G' contains at most $K + 1$ edges of the type $[v, v']$.*

We want to relate $(2K + 1)$-cardinality trees of G' and K-cardinality trees of G. In that respect the most important property of the transformation is that for minimal weight $(2K + 1)$-cardinality trees of G' we can prove the converse of Corollary 1.

Lemma 2 *Any minimal weight $(2K + 1)$-cardinality tree in G' contains at least $K + 1$ edges of the type $[v, v']$.*

Proof:
Let T be a minimal weight $(2K + 1)$-cardinality tree of G' and assume that T contains not more than K edges of $\{[v, v'] \mid v \in V\}$. Then $w(T) \geq (K + 1)M + K \min_{v \in V} w_V(v)$. Now let T' be any K-cardinality tree of G and define $T^* = T' \cup \{[v, v'] \mid v \in V(T')\}$. By the choice of M we conclude

$$w(T^*) \leq KM + (K + 1) \max_{v \in V} w_V(v) < (K + 1)M \leq w(T)$$

contradicting optimality of T (Note that we only consider cases where $K \leq m - 1$).

□

Combining Corollary 1 and Lemma 2 it is easy to prove the validity of the transformation, as stated in Theorem 1.

Theorem 1 *T is a minimal K-cardinality node weighted tree in G if and only if $T \cup \{[v, v'] \mid v \in V(T)\}$ is a minimal $(2K + 1)$-cardinality edge-weighted tree in G'.*

This transformation makes use of the fact that a tree with K edges has $K + 1$ nodes. As there is no similar result for general connected subgraphs the transformation fails for the K-Card Subgraph problem. Hence we modify the heuristics for K-Card Subgraph of Section 3.2 in order to have methods to deal with node-weighted graphs in K-Card Subgraph. The presentation of these heuristics will close this section.

K-CardPrim-NodeWeight (K-CaPr-NW)

1 For all $s \in V$
  $V(S_s) := \{s\}, E(S_s) := \emptyset$
  Repeat
    Let $i_0 \in V \setminus V(S_s)$ be a node such that
      $\min_{i \in V \setminus V(S_s)} \left\{ \frac{w_V(i)}{\min(K - |V(S)|, |\{[i,j] : j \in V(S_s)\}|)} \right\}$ is attained at i_0
    $V(S_s) := V(S_s) \cup \{i_0\}$
    $E(S_s) := E(S_s) \cup \{[i_0, j] : j \in V(S_s)\}$
  until $|E(S)| = K$

2 $S_{heur} := \text{argmin}\{w(S_s) \mid s = 1, \dots, n\}$

That means starting once from each node we add additional nodes and incident edges in such a way that the weight per additional edge is as small as possible. The denominator in the minimum operation is equal to the maximal number of edges we can include in S when adding node i.
The dual greedy procedure on the other hand starts with the graph G and deletes nodes and their incident edges. Again the node to be deleted is chosen according to the maximal weight reduction per deleted edge. Additionally we require that a connected graph is retained in each step. When no more nodes can be deleted we will eventually have to remove some edges to get a K-cardinality subgraph. This can be done without changing the weight of the solution found so far.
Another version of dual greedy type, where only components of the graph which contain enough edges are considered turned out to be inferior in almost all tests and we will not describe it here.

DualGreedy-NodeWeight (DuGr-NW)

1 $S := G$

2 Repeat
  Let $i_0 \in V(S_s)$ be a node such that
    $\max_{i \in V(S)} \left\{ \frac{w_V(i)}{|\{[i,j] : j \in V(S)\}|} \right\}$ is attained at i_0 and
    $E(S) \setminus \{[i,j] : j \in V(S)\}$ is connected and has at least K edges
  $V(S) := V(S) \setminus \{i_0\}$
  $E(S) := E(S) \setminus \{[i_0, j] : j \in V(S)\}$
 until no i_0 can be found

3 Delete $|E(S)| - K$ edges from S such that it remains connected

4 $S_{heur} := S$

5 Numerical results

In this section we will present the results of computational tests carried out with the heuristics of Sections 3 and 4.
We begin the section by summarizing the worst-case complexities of all the heuristics in Table 1.

Greedy heuristics	K-CaPr	Kn^2
	DuGr1	$n^2(m-K)$
	DuGr2	$n^2(m-K)$
Path heuristics	DiPr-A	$n^3+Kn^2+K^2n$
	DiPr-B	$n^3+Kn^2+K^2n$
	DiPr	$n^3+Kn^2+K^2n$
DynamicTree heuristics	DPT	$n^3+n^2K^2$
	DDP	$n^3+K^3n+Kn^2$
	DDT	$n^3+n^2K^2$
Subgraph EdgeWeight heuristics	K-CaPr-S	Kn^2
	DuGr1-S	$n^2(m-K)$
	DuGr2-S	$n^2(m-K)$
Subgraph NodeWeight heuristics	K-CaPr-NW	Kn^2
	DuGr-NW	$n^2(m-K)$

Table 1: Computational Complexity of K-Card Tree Heuristics

The complexities can be subsumed under $O(n^3)$, except for the dynamic tree and dual greedy heuristics which in the worst case need $O(n^4)$ operations. But the more detailed analysis gives an indication to the average running times of the algorithms. However we remark that for the heuristics for the node-weighted problem the worst case bounds will only be attained if in each step only one edge is included or deleted. This situation will occur only very rarely.
We do not claim to have the best implementations possible, since no special data structures such as binary trees or heaps (see e.g. [Meh84]) have been used. Therefore it is possible to reduce worst case bounds for the heuristics which, however, was not the intention of this paper.
All heuristics were implemented in TURBO-PASCAL and run on PC's. The implementations can be found in [EF96]. As mentioned in Section 1 optimal solutions for the test problems were obtained by a Branch&Cut algorithm implemented in C++ on an IBM RS 6000 using CPLEX 2.1 as LP-solver. These Branch&Cut algorithms can also be found in [EF96].
The tests were carried out in the following way: We randomly generated 20 connected graphs and 20 grid graphs with 10, 20, and 30 nodes (i.e. 120 in total for K-Card Tree and 120 for K-Card Subgraph). The minimal and maximal number of edges are are given in Table 2 below.

Nodes	Edges					
	K-Card Tree		K-Card Subgraph		K-Card Subgraph Node-Weight	
	Graphs	Grid Graphs	Graphs	Grid Graphs	Graphs	Grid Graphs
10	9-35	12-13	9-45	12-13	9-38	12-13
20	24-120	25-30	30-153	26-30	40-148	25-30
30	179-235	39-48	100-268	35-47	149-272	39-47

Table 2: Graphs for Numerical Tests

All K-Card Tree heuristics were tested with randomly generated weights between 1 and 100 according to three distributions: exponential, uniform, and normal. As there was no significant difference in the overall behaviour, we will only present the results for the uniform distribution. Results for exponentially or normally distributed weights were often slightly better.
For the K-Card Subgraph heuristics we therefore restricted ourselves to uniformly distributed weights, but we also tested weights depending on the number of nodes, i.e. distributed uniformly in the interval [1,2n], which also did not lead to a significant change in the results.
The tests were carried out for cardinality K ranging from 3 to $n-2$ (K-Card-Tree) and 3 to $\frac{3}{2}n$ (K-Card Subgraph). However we remark that very small values of K ($K < 10$) do not seem to

be very reasonable for practical examples. There larger values of K will appear most often, e.g. $K = \frac{n}{2}$ in the oil field leasing problem.
As the results for smaller examples on 10 and 20 nodes were generally better than for 30 nodes we will only present the results for the series with 30 nodes.
That means, considering tests for graphs on 10, 20 and 30 nodes that 2880 K-Card Tree problems have been solved to optimality, plus the same number of problems on grid graphs. For K-Card Subgraph we solved 3316 problems on graphs and 3080 on grid graphs for the edge weighted case and 3303 respectively 3091 problems for the node-weighted case. The numbers differ because some of the randomly generated graphs did not have $\frac{3n}{2}$ edges. For each value of $K \in \{3, \ldots, n-2\}$ and $K \in \{3, \ldots, \frac{3}{2}\}$, respectively, we calculated the average relative deviation from the optimal objective value, based on the results of the 20 tests. These values are shown graphically in Figures 2 to 6.
Concerning running times we remark that even the most time consuming heuristics produced a solution for all 30 nodes examples within a few seconds. A complete series of up to 860 tests for one heuristic could be run within at most one hour of CPU-time. On the other hand a complete series of tests for graphs with 30 nodes took several days on the IBM RS6000. Furthermore it was not possible to solve a reasonably large number of problems on graphs with 40 nodes optimally. For this reason we couldn't test larger examples than the ones presented here. Recall that for examples on graphs with 30 nodes we have about 250 0-1 variables and on graphs with 40 nodes we would already have more than 400.
Let us summarize: All results we present are for graphs and grid graphs with 30 nodes and weights uniformly distributed in $\{1, \ldots, 100\}$. Looking at the results we make the following observations:

- Most of the heuristics were better for general graphs than for grid graphs, with few exceptions. E.g. **DDT** gave better results on grid graphs.
- Dual greedy approaches are not very good considered alone. But they show an interesting complementarity to the **K-CardPrim** approach for all problems: They find good or optimal solutions in cases where the latter doesn't. This is especially evident in the results for K-Card Subgraph.
- **DynamicTree** approaches were in most, although not all, cases the best for K-Card Tree.
- In most of the problems the best results were obtained for the smallest and largest values for K whereas for values in the middle of the range deviations were bigger. This is to be expected for all heuristics since the number of alternative solutions is largest for those values.
- Taking the best of the solutions any of the heuristics produced we calculated the column BEST. The improvement compared to any single heuristic is clear, indicating that each has its advantages. These results are very good for all problems.
- The quality of the heuristics is also indicated by the number of problems for which the optimal solution was found by at least one of them. These numbers (as percentage of the total number of problems) are given in Table 3.

Problem	Graph Type	Optimal
K-Card Tree	Graphs	94%
	Grid Graphs	90%
K-Card Subgraph Edge Weights	Graphs	90%
	Grid Graphs	72%
K-Card Subgraph Node Weights	Graphs	91%
	Grid Graphs	57%

Table 3: Percentage of Problems where Optimal Solution Has Been Found (30 Nodes Examples)

We conclude that the heuristics presented in Sections 3 and 4 yield very good results in a few seconds of time as indicated in Table 3 and Figures 2 - 6. Therefore they supposedly will perform well on larger real world examples, which cannot be solved optimally within a reasonable amount of time.
They furthermore can be incorporated as upper bounds within Branch&Cut algorithms to improve their performance. However this was beyond the scope of this paper and is a topic of future research. We also refer to the possibility of applying local search heuristics such as Tabu Search either from the beginning or using the solution of some of the heuristics as an initial solution. The former has been done in [JL96]. The latter however seems not to be very promising for the size of examples we studied here if we take into account the quality of the solutions our heuristcs produce and the increased time needed to apply local search methods.

References

[Dij59] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[EF96] M. Ehrgott and J. Freitag. A program package for solving k-cardinality tree and subgraph problems. *European Journal of Operational Research*, 1996. to appear.

[Ehr92] M. Ehrgott. Optimization problems in graphs under cardinality restrictions (in German). Master's thesis, University of Kaiserslautern, Department of Mathematics, 1992.

[Ehr95] M. Ehrgott. K-cardinality subgraphs. In *OR-Proceedings 94, Selected Papers of the International Conference on Operations Research Berlin, 30.8-2.9.1994*, pages 86–91. Springer: Berlin, Heidelberg, New-York, 1995.

[FH92] 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.

[FHJM94] M. Fischetti, H.W. Hamacher, K. Joernsten, and F. Maffioli. Weighted k-cardinality trees: Complexity and polyhedral structure. *Networks*, 24:11–21, 1994.

[FK94] U. Faigle and W. Kern. Computational complexity of some maximum average weight problems with precedence constraints. *Operations Research*, 42(4):688–693, 1994.

[Fre93] J. Freitag. Minimal k-cardinality trees (in German). Master's thesis, University of Kaiserslautern, Department of Mathematics, 1993.

[HJ93] H.W. Hamacher and K. Joernsten. Optimal relinquishment according to the norwegian petrol law: A combinatorial optimization approach. Technical Report 7/93, Norwegian School of Economics and Business Administration, Bergen, 1993. submitted to Applied Mathematical Modelling.

[HJM91] H.W. Hamacher, K. Joernsten, and F. Maffioli. Weighted k-cardinality trees. Technical Report 91.023, Politecnico di Milano, Dipartimento di Elettronica, 1991.

[JL96] K. Joernsten and A. Lokketangen. Tabu search for weighted k-cardinality trees. *Asia Pacific Journal of Operational Research*, 1996. to appear.

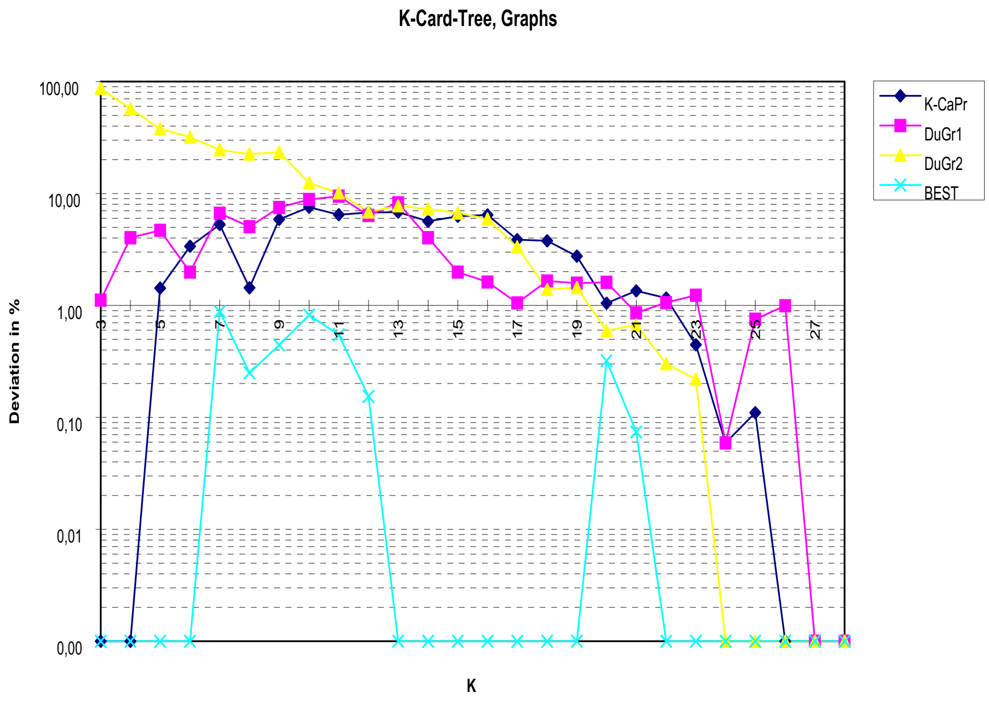
[Kru56] J.B. Kruskal. On the shortest spanning subtree of a graph and the travelling salesman problem. *Proceedings of the American Mathematical Society*, 7:48–50, 1956.

[Maf91] F. Maffioli. Finding a best subtree of a tree. Technical Report 91.041, Politecnico di Milano, Dipartimento di Elettronica, 1991.

[Meh84] K. Mehlhorn. *Data Structures and Algorithms 1: Sorting and Searching*. Springer Verlag, Berlin, 1984.

[Pri57] J.C. Prim. Shortest connection networks and some generalizations. *Bell System Technics Journal*, 36:1389–1401, 1957.

[Woe92] G.J. Woeginger. Computing maximum valued regions. *Acta Cybernetica*, 10(4):3030–315, 1992.



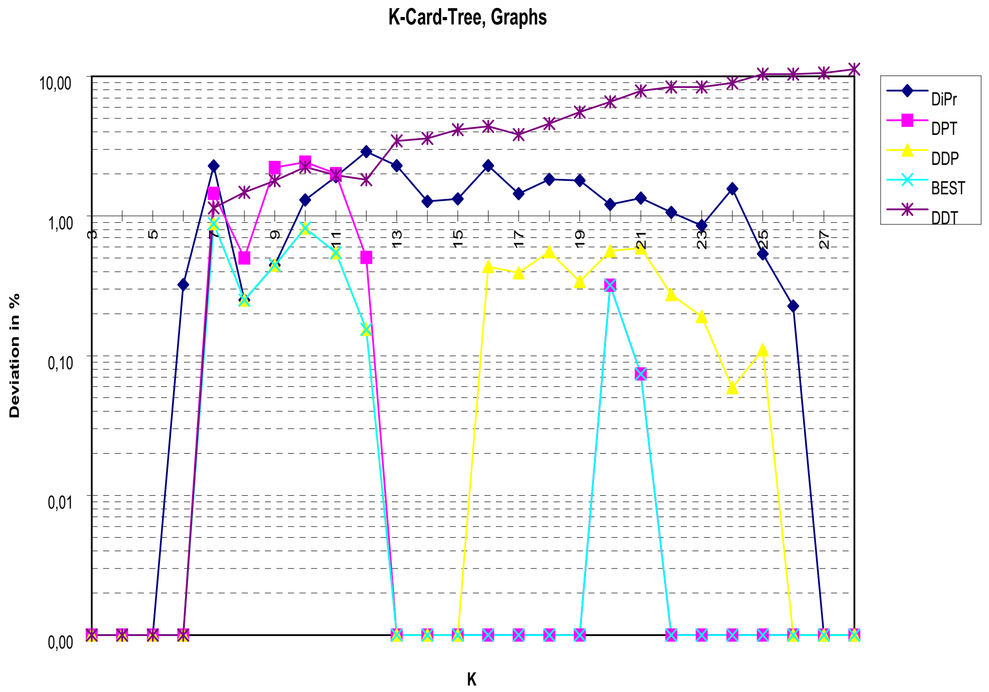
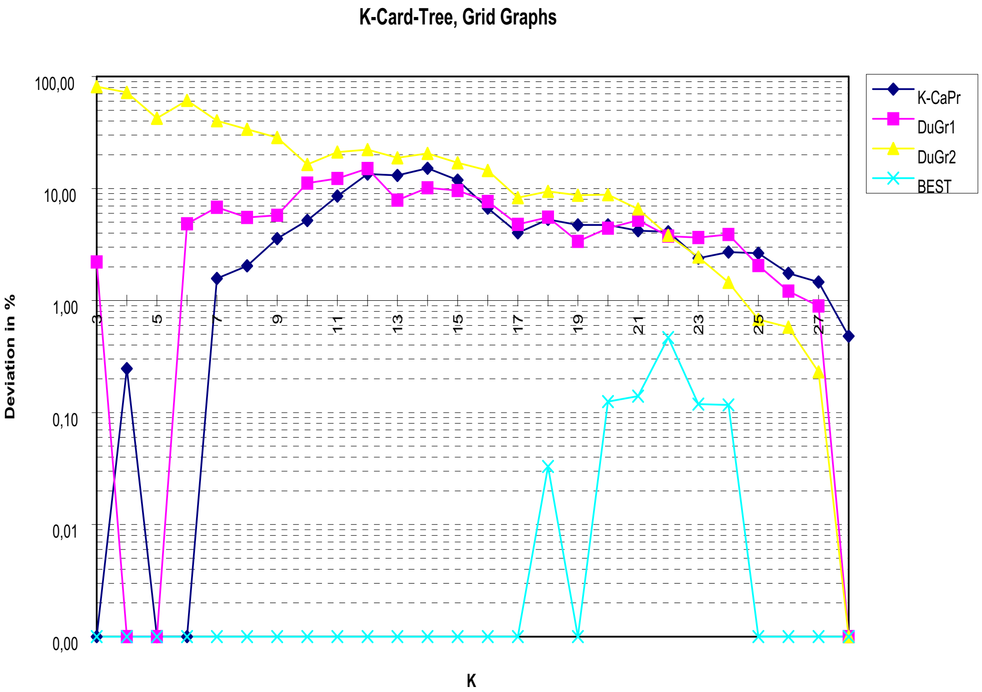


Figure 2: Results for K-Card Tree on Graphs (30 Nodes)



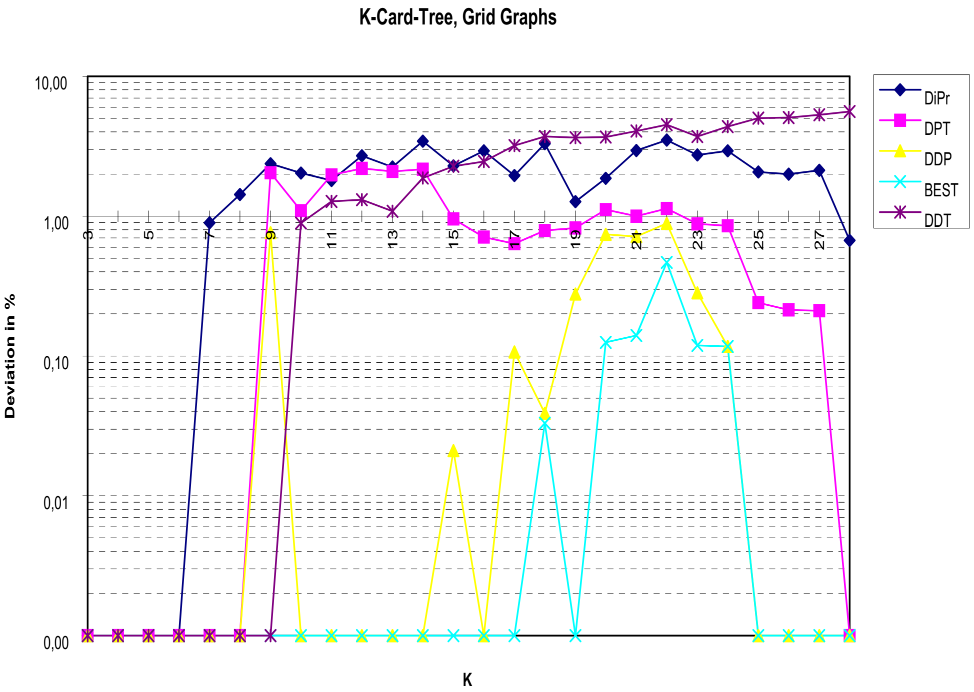
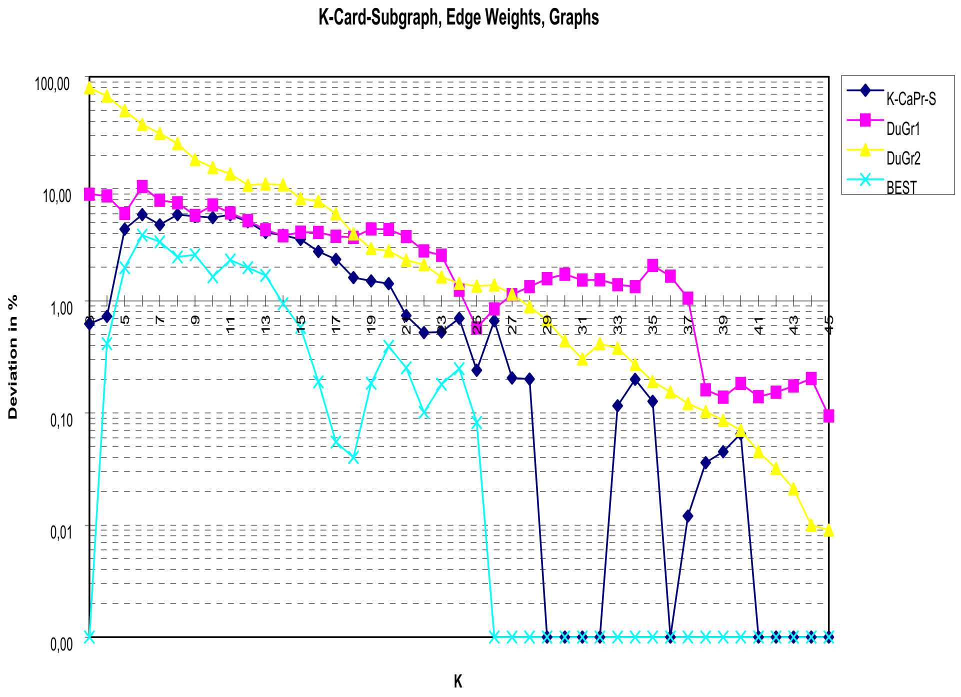


Figure 3: Results for K-Card Tree on Grid Graphs (30 Nodes)



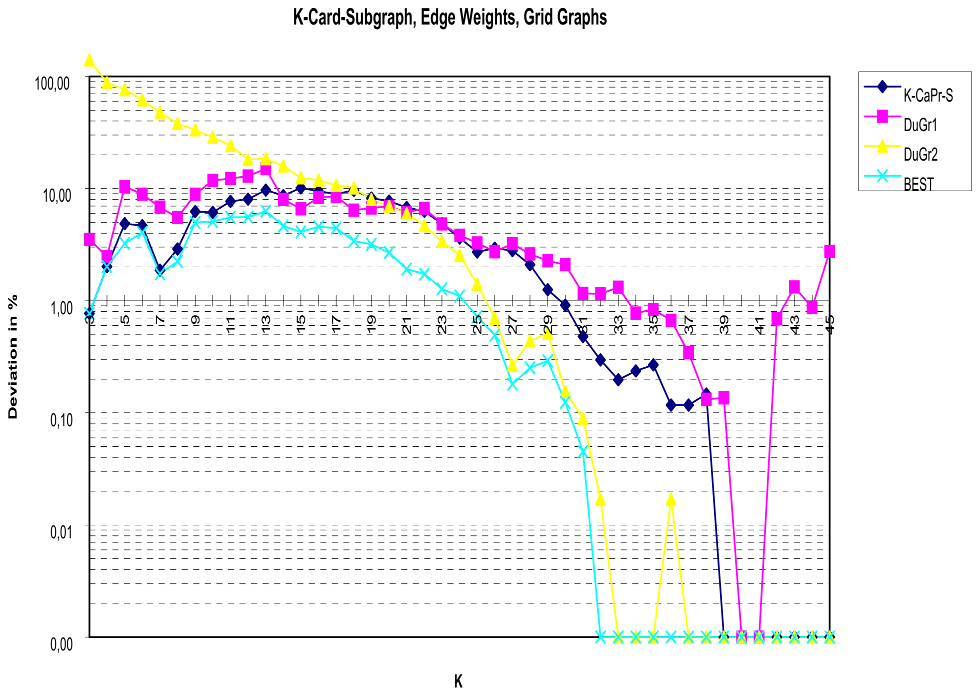
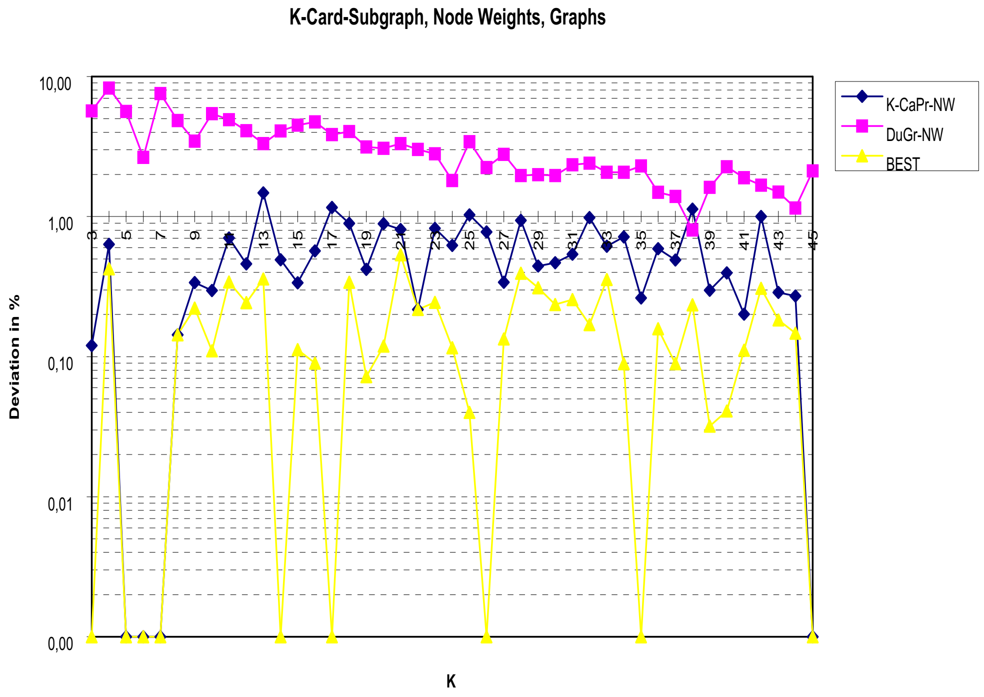


Figure 4: Results for K-Card Subgraph with Edge Weights (30 Nodes)



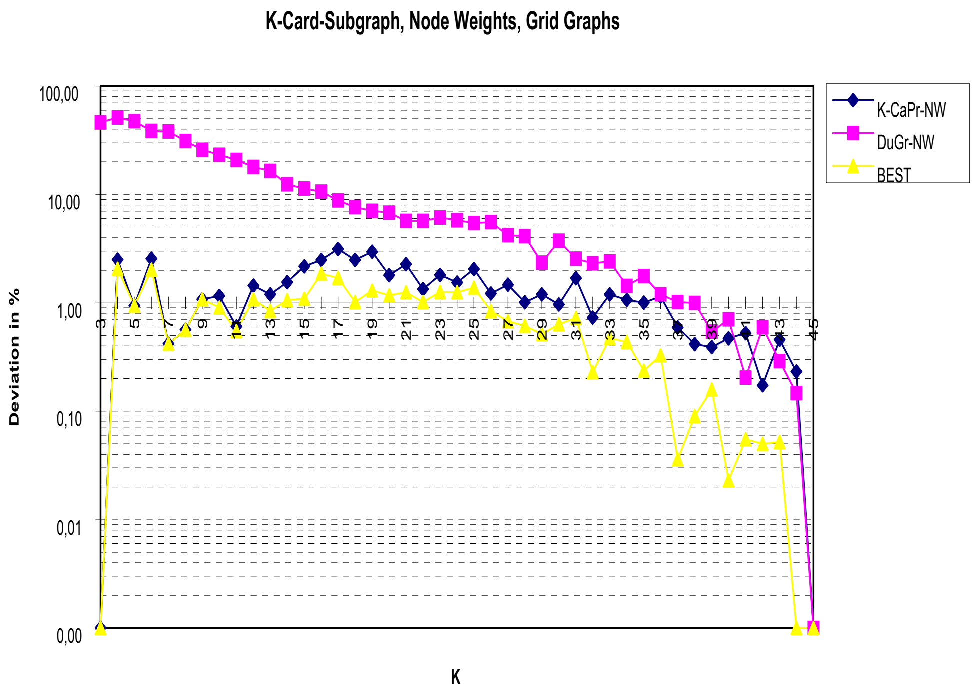


Figure 5: Results for K-Card Subgraph with Node Weights (30 Nodes)

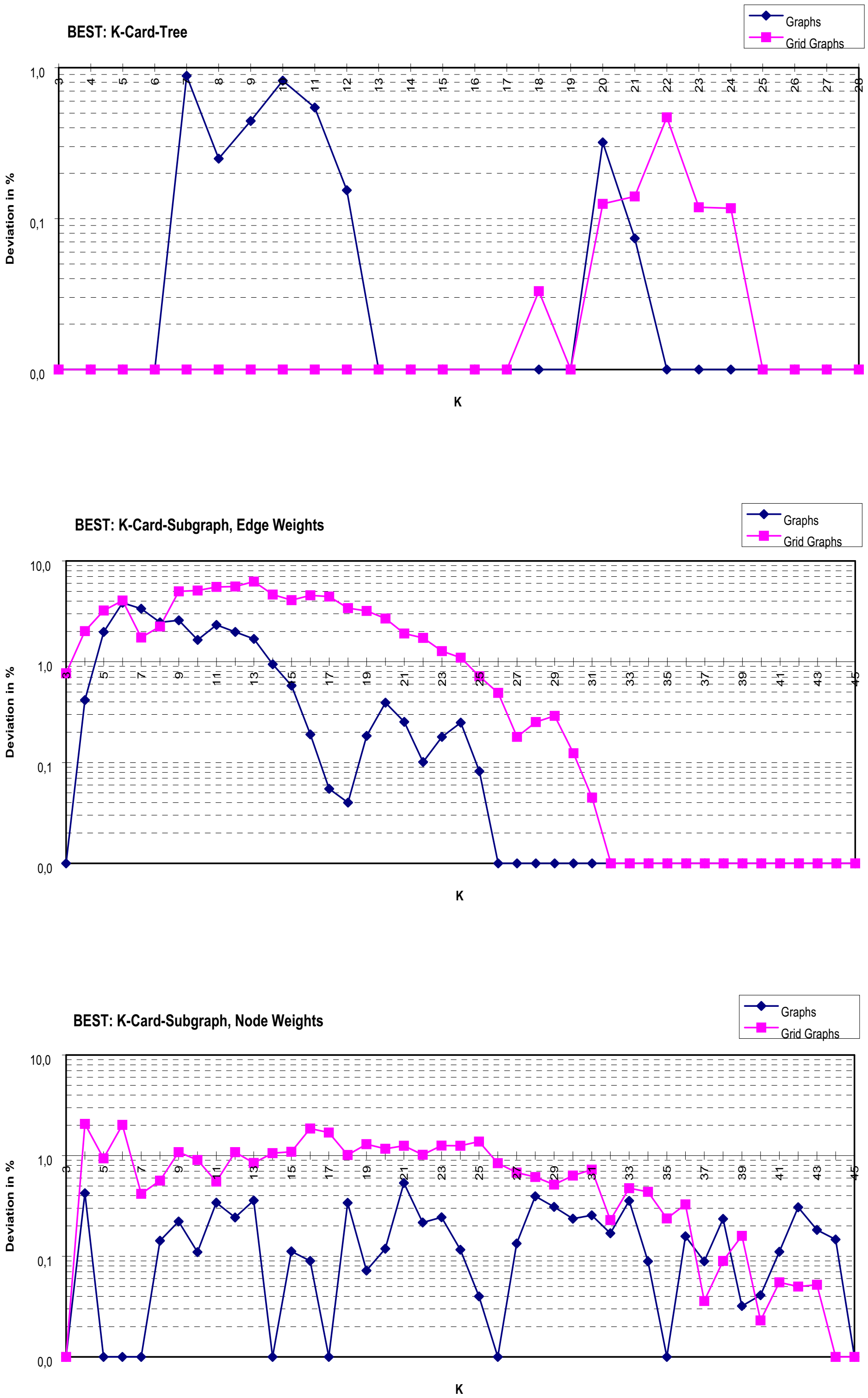


Figure 6: Results Choosing Best Solutions

K	K-CaPr	DuGr1	DuGr2	DiPr	DPT	DDP	DDT	BEST
3	0	1.111	86.861	0	0	0	0	0
4	0	4.017	56.962	0	0	0	0	0
5	1.429	4.684	37.817	0	0	0	0	0
6	3.384	1.983	31.978	0.323	0	0	0	0
7	5.289	6.665	24.649	2.288	1.454	0.882	1.141	0.822
8	1.435	5.055	22.423	0.250	0.5	0.250	1.475	0.250
9	5.885	7.501	23.420	0.444	2.222	0.444	1.789	0.444
10	7.548	8.810	12.460	1.300	2.431	0.821	2.247	0.821
11	6.458	9.450	10.134	1.917	2.008	0.545	1.955	0.545
12	6.757	6.368	6.839	2.891	0.505	0.154	1.824	0.154
13	6.812	8.237	7.756	2.291	0	0	3.450	0
14	5.653	4.017	7.285	1.270	0	0	3.591	0
15	6.260	1.987	6.692	1.324	0	0	4.153	0
16	6.425	1.616	5.957	2.296	0	0.435	4.388	0
17	3.867	1.052	3.341	1.445	0	0.392	3.830	0
18	3.781	1.651	1.392	1.830	0	0.556	4.581	0
19	2.751	1.583	1.444	1.792	0	0.339	5.540	0
20	1.046	1.610	0.593	1.211	0.320	0.560	6.562	0.320
21	1.348	0.856	0.669	1.345	0.074	0.593	7.882	0.074
22	1.164	1.060	0.301	1.061	0	0.274	8.390	0
23	0.446	1.234	0.218	0.853	0	0.191	8.390	0
24	0.059	0.059	0	1.565	0	0.059	8.963	0
25	0.110	0.755	0	0.536	0	0.110	10.322	0
26	0	0.995	0	0.227	0	0	10.390	0
27	0	0	0	0	0	0	10.580	0
28	0	0	0	0	0	0	11.270	0

Table 4: Deviations from Optimality in Percent: K-Card Tree on Graphs with 30 Nodes

K	K-CaPr	DuGr1	DuGr2	DiPr	DPT	DDP	DDT	BEST
3	0	2.206	81.240	0	0	0	0	0
4	0.246	0	71.889	0	0	0	0	0
5	0	0	42.472	0	0	0	0	0
6	0	4.841	61.056	0	0	0	0	0
7	1.576	6.983	40.321	0.895	0	0	0	0
8	2.038	5.501	33.780	1.430	0	0	0	0
9	3.561	5.770	28.609	2.376	2.043	0.762	0	0
10	5.190	11.195	16.402	2.035	1.092	0	0.892	0
11	8.569	12.313	21.113	1.795	1.967	0	1.276	0
12	13.477	15.107	22.914	2.708	2.195	0	1.310	0
13	13.066	7.886	18.827	2.246	2.085	0	1.085	0
14	15.124	10.133	20.618	3.431	2.165	0	1.872	0
15	11.870	9.577	16.922	2.286	0.954	0.021	2.279	0
16	6.699	7.652	14.460	2.929	0.708	0	2.458	0
17	4.022	4.786	8.269	1.948	0.635	0.107	3.191	0
18	5.296	5.543	9.428	3.299	0.789	0.039	3.713	0.033
19	4.723	3.380	8.719	1.267	0.821	0.278	3.644	0
20	4.732	4.428	8.826	1.863	1.115	0.741	3.683	0.125
21	4.202	5.210	6.578	2.947	1.000	0.716	4.059	0.140
22	4.108	3.765	3.821	3.500	1.133	0.886	4.503	0.468
23	2.384	3.651	2.445	2.741	0.878	0.283	3.711	0.119
24	2.710	3.885	1.455	2.939	0.852	0.117	4.368	0.117
25	2.652	2.046	0.677	2.069	0.240	0	5.022	0
26	1.752	1.212	0.578	1.993	0.213	0	5.071	0
27	1.462	0.896	0.231	2.123	0.211	0	5.314	0
28	0.479	0	0	0.671	0	0	5.607	0

Table 5: Deviations from Optimality in Percent: K-Card Tree on Grid Graphs with 30 Nodes

K	Graphs				Grid Graphs			
	K-CaPr-S	DuGr1	DuGr2	BEST	K-CaPr-S	DuGr1	DuGr2	BEST
3	0.625	8.967	79.966	0	0.770	3.515	140.037	0.770
4	0.729	8.643	67.170	0.417	2.012	2.470	87.773	2.012
5	4.362	6.011	49.874	1.981	4.873	10.403	76.434	3.228
6	5.887	10.483	37.645	3.857	4.678	8.889	61.834	4.048
7	4.774	7.883	31.259	3.362	1.868	6.820	47.631	1.740
8	5.891	7.522	25.324	2.461	2.897	5.518	38.059	2.238
9	5.682	5.807	18.200	2.578	6.224	8.883	33.432	4.985
10	5.525	7.175	15.481	1.646	6.144	11.786	28.667	5.084
11	5.864	6.107	13.616	2.318	7.671	12.307	24.228	5.531
12	5.089	5.199	10.761	1.979	8.035	12.932	18.161	5.563
13	4.090	4.315	11.016	1.685	9.674	15.054	18.404	6.258
14	3.858	3.809	10.825	0.943	8.657	8.002	15.924	4.645
15	3.538	4.099	8.174	0.580	10.140	6.611	12.475	4.095
16	2.758	4.069	7.793	0.190	9.458	8.296	11.890	4.564
17	2.349	3.777	5.980	0.055	8.928	8.523	10.666	4.446
18	1.612	3.679	3.977	0.040	9.647	6.407	10.150	3.412
19	1.511	4.396	2.941	0.184	8.224	6.720	8.004	3.189
20	1.430	4.350	2.792	0.393	7.747	6.930	6.956	2.679
21	0.739	3.756	2.309	0.253	6.771	6.145	6.006	1.908
22	0.521	2.796	2.112	0.101	6.274	6.679	4.680	1.726
23	0.527	2.558	1.630	0.180	4.872	4.824	3.370	1.273
24	0.699	1.245	1.434	0.249	3.621	3.840	2.544	1.099
25	0.240	0.574	1.357	0.082	2.723	3.276	1.399	0.718
26	0.664	0.847	1.386	0	2.938	2.731	0.698	0.492
27	0.205	1.139	1.153	0	2.790	3.220	0.265	0.180
28	0.201	1.343	0.886	0	2.091	2.613	0.437	0.252
29	0	1.581	0.674	0	1.253	2.262	0.512	0.291
30	0	1.729	0.439	0	0.915	2.089	0.156	0.124
31	0	1.535	0.303	0	0.478	1.164	0.089	0.045
32	0	1.545	0.415	0	0.296	1.150	0.017	0
33	0.116	1.394	0.378	0	0.197	1.316	0	0
34	0.200	1.342	0.271	0	0.236	0.775	0	0
35	0.127	2.073	0.191	0	0.268	0.834	0	0
36	0	1.665	0.154	0	0.117	0.668	0.017	0
37	0.012	1.056	0.121	0	0.117	0.344	0	0
38	0.036	0.161	0.103	0	0.147	0.132	0	0
39	0.045	0.138	0.086	0	0	0.136	0	0
40	0.065	0.183	0.070	0	0	0	0	0
41	0	0.140	0.045	0	0	0	0	0
42	0	0.153	0.032	0	0	0.687	0	0
43	0	0.174	0.021	0	0	1.324	0	0
44	0	0.203	0.010	0	0	0.865	0	0
45	0	0.094	0.009	0	0	2.754	0	0

Table 6: Deviations from Optimality in Percent: K-Card Subgraph with Edge Weights (Graphs and Grid Graphs with 30 Nodes)

K	Graphs			Grid Graphs		
	K-CaPr-NW	DuGr-NW	BEST	K-CaPr-NW	DuGr-NW	BEST
3	0.120	5.685	0	0	46.203	0
4	0.634	8.256	0.424	2.507	51.198	2.069
5	0	5.625	0	0.941	47.570	0.941
6	0	2.650	0	2.547	38.511	2.019
7	0	7.555	0	0.418	38.199	0.418
8	0.143	4.845	0.143	0.566	31.244	0.566
9	0.338	3.458	0.222	1.082	25.825	1.082
10	0.296	5.426	0.110	1.161	23.330	0.907
11	0.705	4.928	0.341	0.606	20.831	0.552
12	0.459	4.108	0.243	1.441	18.016	1.082
13	1.476	3.320	0.359	1.195	16.503	0.846
14	0.492	4.079	0	1.552	12.433	1.057
15	0.337	4.500	0.112	2.169	11.320	1.095
16	0.569	4.734	0.090	2.484	10.640	1.861
17	1.163	3.860	0	3.152	8.789	1.700
18	0.895	4.057	0.339	2.483	7.658	1.016
19	0.420	3.140	0.072	2.959	7.063	1.301
20	0.891	3.068	0.119	1.803	6.802	1.173
21	0.808	3.314	0.535	2.273	5.693	1.251
22	0.217	3.015	0.217	1.335	5.713	1.019
23	0.825	2.813	0.244	1.808	6.145	1.257
24	0.622	1.811	0.116	1.544	5.785	1.255
25	1.028	3.432	0.040	2.052	5.451	1.381
26	0.778	2.241	0	1.218	5.561	0.841
27	0.339	2.779	0.134	1.474	4.217	0.679
28	0.941	1.960	0.395	1.010	4.121	0.614
29	0.443	1.991	0.310	1.200	2.342	0.515
30	0.469	1.961	0.236	0.964	3.743	0.633
31	0.540	2.343	0.255	1.692	2.560	0.727
32	0.981	2.411	0.169	0.730	2.317	0.229
33	0.614	2.070	0.356	1.195	2.409	0.474
34	0.718	2.076	0.089	1.069	1.440	0.437
35	0.262	2.301	0	1.003	1.771	0.237
36	0.589	1.489	0.158	1.142	1.195	0.328
37	0.491	1.395	0.089	0.595	1.017	0.036
38	1.131	0.803	0.235	0.417	1.001	0.090
39	0.298	1.617	0.032	0.391	0.543	0.159
40	0.397	2.267	0.041	0.472	0.702	0.023
41	0.201	1.896	0.111	0.527	0.204	0.055
42	1.001	1.677	0.307	0.174	0.595	0.050
43	0.288	1.495	0.183	0.457	0.289	0.052
44	0.271	1.150	0.147	0.233	0.147	0
45	0	2.120	0	0	0	0

Table 7: Deviations from Optimality in Percent: K-Card Subgraph with Node Weights (Graphs and Grid Graphs with 30 Nodes)